Journal of Artificial Intelligence Research 58 (2017) 523-590 Submitted 08/16; published 03/17

Robots in Retirement Homes: Applying Off-the-Shelf Planning and Scheduling to a Team of Assistive Robots

Tony T. Tran TRAN@MIE.UTORONTO.CA
Department of Mechanical and Industrial Engineering
University of Toronto, Canada

Tiago Vaquero TVAQUERO@MIT.EDU
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, USA

Goldie Nejat NEJAT@MIE.UTORONTO.CA
Department of Mechanical and Industrial Engineering
University of Toronto, Canada

J. Christopher Beck JCB@MIE.UTORONTO.CA
Department of Mechanical and Industrial Engineering
University of Toronto, Canada

Abstract

This paper investigates three different technologies for solving a planning and scheduling problem of deploying multiple robots in a retirement home environment to assist elderly residents. The models proposed make use of standard techniques and solvers developed in AI planning and scheduling, with two primary motivations. First, to find a planning and scheduling solution that we can deploy in our real-world application. Second, to evaluate planning and scheduling technology in terms of the "model-and-solve" functionality that forms a major research goal in both domain-independent planning and constraint programming. Seven variations of our application are studied using the following three technologies: PDDL-based planning, time-line planning and scheduling, and constraint-based scheduling. The variations address specific aspects of the problem that we believe can impact the performance of the technologies while also representing reasonable abstractions of the real world application. We evaluate the capabilities of each technology and conclude that a constraint-based scheduling approach, specifically a decomposition using constraint programming, provides the most promising results for our application. PDDL-based planning is able to find mostly low quality solutions while the timeline approach was unable to model the full problem without alterations to the solver code, thus moving away from the model-and-solve paradigm. It would be misleading to conclude that constraint programming is "better" than PDDL-based planning in a general sense, both because we have examined a single application and because the approaches make different assumptions about the knowledge one is allowed to embed in a model. Nonetheless, we believe our investigation is valuable for AI planning and scheduling researchers as it highlights these different modelling assumptions and provides insight into avenues for the application of AI planning and scheduling for similar robotics problems. In particular, as constraint programming has not been widely applied to robot planning and scheduling in the literature, our results suggest significant untapped potential in doing so.

1. Introduction

The recent aging of global populations is unprecedented in human history and it is not expected that we will return to the younger population profiles of our ancestors (United Nations, 2002). The large increase in the aged population has had, and will continue to have, profound impact on social and economic facets of our society. Of particular concern is the welfare and well-being of the elderly as their physical, cognitive, and psychological requirements must be adequately met. However, without proportionately increasing the number of professional caregivers, the increase of the older population will lead to a strain on existing caregivers. To address the lack of human resources, human-robot interaction (HRI) and robot companionship have been proposed and shown to have positive results on the human psychological state (Banks, Willoughby, & Banks, 2008; Turkle, 2006). To that end, we have undertaken a long-term study on the deployment of intelligent human-like mobile robots in retirement homes to assist and interact with elderly residents (Booth, Tran, Nejat, & Beck, 2016; Louie, Vaquero, Nejat, & Beck, 2014; Louie, Li, Vaquero, & Nejat, 2014, 2015; Mohamed & Nejat, 2016).

The proposed research problem includes a myriad of technical challenges with respect to robot hardware, control, sensing and intelligence. Our focus in this paper is on an important part of the overall challenge: developing global decision making techniques which can plan and schedule the high-level tasks that a set of robots will perform during the day in a retirement home. The decisions that must be made are what tasks to perform, where and when to perform them, which residents are involved with these tasks, and which robot performs a particular task. It is, of course, critical to take into account the personal preferences, schedules, and requirements of each resident, creating a complex coordination problem where planned tasks must fit into the daily operations of a retirement home.

This paper aims to study whether a particular planning application can be modelled and solved using current planning and scheduling technologies. Such case studies serve as valuable feedback for researchers who focus on the theory and algorithms which form the core of planning and scheduling research. As we use off-the-shelf technology, this paper serves as a test of the extent to which two AI fields, domain-independent planning and constraint programming (CP), are progressing toward the "holy grail" (Freuder, 1997) of declarative problem solving. While this test, focusing on a single application, is far from definitive, it does provide needed insight as to where we are in this quest as well as towards the goal of solving similar planning and scheduling problems.

The main contributions of this paper are:

- The modelling of a complex multi-robot HRI problem for three different solving technologies: Planning Domain Definition Language (PDDL), New Domain Description Language (NDDL), and CP. The principles and practice of taking a real problem and developing a model are not often discussed and alternative models tend to not be explored in depth in the planning community. This work contributes to the knowledge needed for effective modelling;

- The modelling and solving study of a complex temporal planning problem that lies at the intersection of planning and scheduling;

- An investigation of alternative models in PDDL for timed events and multi-user actions;
- The introduction of one of the first applications of CP to a multi-robot planning and scheduling problem;
- Identification of particular model components for PDDL planners (timed initial literals, temporal constraints, and complex objective functions) and CP solvers (massive numbers of optional activities and complex objective functions) that are challenging for the technology.

The following section presents the motivation and background of our work. Details of the robot planning and scheduling application are presented in Section 3. Models developed for the three technologies are outlined in Section 4, followed by experimental results comparing the different technologies in Section 5. Lastly, in Section 6, discussions of our results are presented followed by ideas for future work and conclusions.

2. Motivation and Background

Our long-term project is the deployment of intelligent mobile robots in retirement homes to engage residents in stimulating recreational activities (Louie, Han, & Nejat, 2013; Louie, Li, Vaquero, & Nejat, 2014; Louie, Vaquero, Nejat, & Beck, 2014; Li, Louie, Despond, & Nejat, 2016). We have designed the robot known as Tangy to: 1) navigate using a laser range finder and 3D depth sensors, 2) detect users with 2D cameras, and 3) interact with users through speech, gestures, and a touch screen. While the implementation of the robot behaviors addresses robotics challenges (e.g., sensing, HRI, person and activity recognition), herein, we focus on the planning and scheduling of the daily activities of the socially assistive robots. These plans and schedules are to be generated prior to the start of a day and will be executed by the robots during the day.

We investigate the potential technologies to be implemented to generate daily plans and schedules. Due to external factors and uncertainties involved with human interaction, plans and schedules may fail. However, we leave replanning and rescheduling online for future work as being able to obtain a schedule first is a crucial step towards implementation.

We focus on two representative activities within the retirement home setting: *telepresence* sessions and *Bingo* games. In the former, the robot autonomously navigates to a user in his/her private room, prompts the user for a previously requested video call, starts the call and tracks the user during the session. For a Bingo game, the robot autonomously finds and reminds participants about the game prior to its start and then navigates to a specified location to conduct the game. During Bingo, the robot acts as the game facilitator, calling out numbers, verifying Bingo cards, prompting players to mark missed numbers and celebrating with winners. These two activities provide the framework necessary to represent any other single or multi-user activities relevant to our system. A centralized server will plan, schedule and monitor the daily tasks of the robots, while lower-level behaviors are planned and performed locally by each individual robot (Vaquero, Mohamed, Nejat, & Beck, 2015).

Planning and scheduling (P&S) is the joint problem of deciding what tasks to perform, when, and with what resources, to achieve a set of goals. At the low-level, if a robot is

given the goal of going to a resident's private room for a telepresence session, it has to plan a series of moves to navigate from its current location to the resident's room. In contrast, if a robot has requests for a number of different tasks with different residents, it needs to schedule these tasks, taking into account the profiles and preferences of the residents, the length of the tasks, and travel time between tasks. Our focus is on this high-level P&S problem.

Planning is a key component of intelligent behavior (Ghallab, Nau, & Traverso, 2004) and is primarily studied within Artificial Intelligence (AI). While initiated in robotics (Nilsson, 1984), planning research has broad applications including in autonomous rovers (Gaines et al., 2006; Jain et al., 2003), spacecraft and satellite control (Frank, 2008; Ghallab et al., 2004; Reddy et al., 2008), clinical decision support systems (Fdez-Olivares, Cózar, & Castillo, 2009), and advanced manufacturing (Vaquero, Tonidandel, de Barros, & Silva, 2006). The algorithmic foundation of AI planning is state-space search (Ghallab et al., 2004).

Scheduling is widely studied in both the AI and Operations Research communities (Pinedo, 2012). The emphasis in the literature has been on the combinatorial nature of a problem and the development of sophisticated optimization techniques. In general, robots have not received as much attention in the constraint-based scheduling literature as they have in the planning literature. Namely, most scheduling work focuses on robots in production lines (Dang, Nielsen, Steger-Jensen, & Madsen, 2013; Kats & Levner, 2011) and robot task scheduling (Zhang & Parker, 2012, 2013). In these works, all tasks must be processed and they do not lead to cascading effects on the actions of robots or require reasoning about causation.

The integration of planning and scheduling has been investigated over the past several years in such robotic applications as container transportation robots (Alami, Chatila, Fleury, Ghallab, & Ingrand, 1998), office assistant robots (Beetz & Bennewitz, 1998), planetary rovers (Estlin et al., 2007), hospital assistant robots (Pecora & Cesta, 2002), and eldercare robots (Cesta et al., 2011; Pineau et al., 2003). In these applications, single robot approaches are commonly studied.

With respect to HRI activities, existing work has mainly focused on automated reasoning about the schedule of a single user. For example, the Pearl robot (Pineau et al., 2003) uses the Autominder system (Pollack, 2005) to reason about an elderly person's current and planned activities to determine if and when reminders should be provided. The Autominder system has not been extended to consider multiple users. The Cobot robots (Coltin, Veloso, & Ventura, 2011) plan and schedule HRI activities, including semi-autonomous telepresence sessions, and office tasks based on requests from several users. However, the planning and scheduling are managed independently and the user schedules are not considered as constraints on the robots' tasks. Our previous work studied a similar system, but with a single robot and different restrictions on activities (Booth et al., 2016). All HRI activities in that work needed to be performed and the users associated with the activities were already known. In this work, interaction activities are optional and the identity of individual users invited to participate in group activities are decisions to be made. Although multiple user schedules have been considered in other non-robotic scheduling and optimization applications (e.g., energy conservation in buildings, Kwak, Varakantham, Maheswaran, Tambe, Jazizadeh, Kavulya, Klein, Becerik-Gerber, Hayes, and Wood, 2012), in this work we focus

on problems in which it is required to reason about the schedules of multiple users, limited resources, metric quantities, and both single- and multi-user HRI activities.

Our research work requires the combination of problem features that are often only individually considered in the literature. It is important for such an application problem that these features, which include optional activities, consideration of user schedules and preferences, and efficient deployment of a fleet of robots, are addressed. To date, such a real world problem has not been studied in the planning and scheduling literature. Our objective is to investigate the use of the existing planning and scheduling technology for our application problem, as well as developing an appropriate model that can provide the required daily schedules of a team of robots in a human-centred environment.

3. Problem Description

The problem of interest is creating a daily schedule for multiple robots in a retirement home environment. We define the main elements of the proposed problem: the environment in which the residents (users) and robots interact, the constraints, the goal and preferences. The constraints for the telepresence sessions and Bingo activities were obtained from meetings with directors, healthcare professionals, and residents from Toronto area retirement homes (Louie, Li, Vaquero, & Nejat, 2014, 2015). The parameters and preferences used herein can be changed as needed without a large impact on the models proposed in this paper.

3.1 The Retirement Home Environment

We consider a floor in a retirement home. The environment consists of rooms and hallways that are discretized as a set of locations, L, within which the users and robots interact. The distance between any two locations l and m, denoted as d_{lm}, is determined as part of the discretization of the retirement home.

3.2 The Users

A number of residents live in the retirement hone, represented by a set of users, U. Each user, $u \in U$, has his or her own user profile consisting of a private room at a location in L, and a schedule for the day. A user's schedule provides the availability and location of the user from 7:00 a.m. to 7:00 p.m. During this time, the user may or may not be available for interaction with a robot: a user u may be available between 10:00 a.m. and 11:00 a.m. at location l, but from 11:00 a.m. to 12:00 p.m. the same user can be at location m and unavailable for any interaction. All users have meal breaks for breakfast (8:00 a.m. to 9:00 a.m.), lunch (12:00 p.m. to 1:00 p.m.), and dinner (5:00 p.m. to 6:00 p.m.), during which no user-robot interactions are possible. Each user also has appointments during which no interaction can occur (e.g., art classes and family visits).

The user's profile also contains their preference for a minimum, att_min_u, and maximum, att_max_u, number of Bingo games. Furthermore, some users may have telepresence sessions that have been booked and must occur at some point during the day when the user is free.

3.3 The Robots

The environment has a set of assistive robots, R, that are responsible for performing the single-user activities (telepresence sessions) and multi-user activities (Bingo games). The robots also provide users with reminders prior to any Bingo games to which they are assigned. To perform these tasks, a robot must travel to the corresponding location (either the current location of the user or to the games room for a Bingo game). Once the robot and user are together at the scheduled time, the robot is then busy for the duration of the task and is not able to perform any other tasks.

While the robot is travelling and performing tasks, it consumes battery power at a rate dependent on the task being executed. The battery level, bl_i, of robot i must always stay between $bl\_min_i \leq bl_i \leq bl\_max_i$. To ensure that a robot's battery has sufficient energy, the robot can be scheduled to recharge its battery up to bl_max_i at a charging station. A constant recharging ratio of rr_i (V/min) is used to approximate the recharging process of robot i. Of course, to recharge, the robot must navigate to the location of a charging station.

A robot moves between locations at a constant velocity v_i and so the estimated time to move from a location l to m is $\frac{d_{lm}}{v_i}$. Moving consumes battery power with a constant rate of cr_move_i, the amount of battery consumed for moving one unit of distance. Each HRI activity has a different rate at which power is consumed: cr_telep_i, cr_remind_i, and cr_Bingo_i represent the consumption rate per minute of a robot i for telepresence sessions, reminders, and Bingo activities, respectively. The robot must always have sufficient battery power to return to a charging station.

3.4 Charging Stations

A set of charging stations, K, is considered for this problem. Each station $k \in K$ is available in one of the locations and is able to recharge any of the robots. There can be any number of charging stations per location. Each station has one docking spot and can charge at most one robot at a time. While a robot is docked, it cannot perform any other tasks. For a given level of charge, β, that is desired after a charging action, the charging duration is $CD(\beta) = \frac{\beta - bl_i}{rr_i}$.

3.5 Telepresence Sessions

A set of telepresence sessions, S, is required to be scheduled during the day. Each telepresence session $y \in S$ is characterized by the user u, the location l (in the user's private room), the duration dur_y (30 minutes), and the time window (or multiple non-overlapping time windows) in which a telepresence session may be held.

3.6 Bingo Games

A set of Bingo games, G, is to be scheduled during the day. Bingo games are optional activities that add value to the daily schedule for users. A Bingo game $g \in G$ is characterized by the location (i.e., the dedicated games room), the duration of the game dur_g (60 minutes), and the time window (or multiple non-overlapping time windows) when it can be played. For each Bingo game g that is played, the number of participants must be no less than

$p\_min_g = 3$ and no more than $p\_max_g = 10$. These users must be available during the game and each player must be reminded of the game by a robot 15 to 120 minutes before it begins. These times are chosen with the assumption that residents may require up to 15 minutes to travel to the games room and that a reminder any longer than 120 minutes prior to the game may be forgotten. The robot that reminds a user does not need to be the same robot that plays the Bingo game. The duration of a reminder is dur_remind_g (2 minutes) and can only be performed when a user is available. To remind a user, the robot must be in the same location as that user.

The group of participants of a game is not known a priori. For a given game, the group of players is a decision variable based on the residents' schedules and attendance preferences.

3.7 Input and Goal

The input of the problem is the sets of locations, L, users, U (with their profiles), charging stations, K, available robots, R (with their initial locations, velocity, and battery level and consumption details), and the requested telepresence sessions, S, and Bingo games, G, with their corresponding properties. The goal is to create a plan of robot tasks in which: 1) all the requested telepresence sessions are scheduled, and 2) the requested Bingo games and reminders are scheduled, if possible, given that user attendance preferences have to be satisfied. All robots must be at a location with a charging station at the end of the day. As a multi-objective optimization problem, we want to: 1) perform as many Bingo games as possible, 2) have as many users playing Bingo as possible, 3) provide reminders as close as possible to the game times, and 4) expend as little battery power as possible.

3.8 Problem Modifications

We propose various modifications to the problem to: 1) study how certain aspects of the problem affect each of the proposed approaches, and 2) obtain better, but still implementable, solutions through solving simplifications of the original problem. The aim is to isolate particular properties of the problem that may prove to be difficult to solve and thus to contribute insights into the strengths and weaknesses of the different technologies.

We consider five independent modifications:

- B: battery. When the battery is removed from the problem, we assume no robots run out of power;
- R: reminder time windows. When removed, the reminders can be performed at any time prior to the start of a Bingo game;
- P: participants. Bingo games may have a varying number of participants. When removed, we assume that all games have exactly four players;
- O: optional Bingo games. This modification requires all Bingo games to be played;
- F: complex objective function. When removed, we only consider user participation and ignore all other objectives.

Each modification can be added or removed independently. We denote the original problem as *BRPOF*, where all aspects of the problem are considered. The problem where

battery levels are ignored and Bingo games are not optional is *-RP-F*. Since there are five independent properties, there are 2^5 combinations. We only look at a subset of these problems, which we believe to exhibit more interesting and informative results: *BRPOF*, *-RPOF*, *B-POF*, *BR-OF*, *BRP-F*, *BRPO-*, and *B—F*. These problems represent the original problem, ones where each modification is made on its own, and the last problem where the modifications regarding the Bingo game and reminders simplifies the Bingo game properties.

Only *BRPOF*, *BR-OF*, *BRP-F*, and *BRPO-* provide *sound* solutions to the original problem. That is, a feasible solution to any of these problems is also a feasible solution to the original problem. The *BR-OF*, and *BRP-F* modifications can lead to infeasibility: no solution may exist with all Bingo games played or with exactly four participants per game. In the scenarios we tested, this is not the case and all modifications have non-empty feasible regions. However, none of these modifications is *complete*: the optimal solution of the original problem may not lie within the feasible region of the modified problem. A solution to *-RPOF* may result in a robot depleting its battery before completing all required tasks. *B-POF* and *B—F* can lead to a solution where a user is reminded outside of the reminder window. Although this is not as catastrophic as a depleted battery, the time windows are intended to be hard constraints. We test the models on problems *-RPOF*, *B-POF*, and *B—F* to observe how batteries and time windows affect the solvability of the models, even though these models are not sound.

The solution of each problem instance by each approach is evaluated a posteriori using the original objective function, which includes the presence of Bingo games, participation of users, delivery times, and energy usage.

4. Planning and Scheduling Technologies

In this section, we present nine models for the aforementioned robot problem: six PDDL-based planning models, a timeline planning and scheduling model, and two constraint programming models. Although other technologies can also be used to solve the problem of interest (e.g., mixed integer linear programming), we do not explore these options in this paper.[1] For each explored technology, we follow modelling techniques and conventions that are customary in the specific literature: we do not want to create a planning model and convert it to CP or vice versa but rather develop "native" models for each technology. Such an approach is the only fair way to test the respective technologies on our application while also bringing into focus fundamentally different modelling approaches: the action and state representation of AI planning vs. the constraint-based representation of CP. We also follow the standard and different model presentation styles of the two areas.

It is clear that exhaustively investigating the modelling space of a complex problem and a non-trivial representation framework is infeasible. While we have investigated numerous modelling choices, some not presented here due to their inferiority, it is possible, perhaps likely, that better models exist.[2] We return to this point in Section 6.4.

1. In a preliminary experiment, mixed integer linear programming was found to have poor performance.
2. Indeed, one of our objectives is to spur researchers to investigate alternative models for this problem.

4.1 PDDL-Based Planning

In order to test the capabilities of planners using PDDL (Ghallab et al., 1998) on the target problem (specifically, we base our models on PDDL2.2 with processes from PDDL+), we test six different models. For clarity of the exposition of these models, we first present one model in detail followed by the differences in the other five models.

It is important to distinguish the six different models and the seven problem modifications described in Section 3.8. Each of the six models can be used to accurately and equivalently represent each of the seven problem modifications. The only difference is in how aspects of the domain are represented. The modelling strategies alter the representation of the problem in PDDL while the problem modifications change the problem.

4.1.1 Domain Modelling

The itSIMPLE Knowledge Engineering tool (Vaquero, Silva, Ferreira, Tonidandel, & Beck, 2009; Vaquero, Silva, Tonidandel, & Beck, 2013) is used to model the proposed problem. itSIMPLE follows an object-oriented modelling approach using Unified Modelling Language (UML) (OMG, 2005) and generates a PDDL model. A UML diagram is presented in this section to help the reader visualize the resulting PDDL model. We also provide key PDDL action specifications to illustrate the main transition, state, resources and temporal constraints in the model.

Object Types and Fluents. A visualization of the modelled object types (classes), fluents and operators for our initial model variation is provided in the UML class diagram in Figure 1. The most important classes are: *Location*, *GamesRoom*, *ChargingStation*, *Robot*, *User*, *TelepresenceSession*, *BingoGame* and *Global*.

The *Location* and *GamesRoom* (a specialization of Location) represent the topology of the retirement home. The distance between locations (*distance*), and the distance between each available charging station and these locations (*distance_to_station*) are represented in the class *Global*. These two static variables provide the distances in meters for every pair (location, location) and (location, station) in the problem.

A games room is said to be *free* (fluent) when no game is taking place at the location. If a robot is performing a task in the games room the fluent *free* is set to false. All the other locations have no representation of their availability.

A *ChargingStation* is said to be *idle* (fluent) when no robot is docked for charging. In order to represent the physical location of a station, we use the fluent *available_at* to assign the station to a particular location object.

The class *User* has a set of properties to represent the user's location in the environment and the user's profile. The fluent *at* refers to the current location of the user who must be at one location at a time. The static variable *room* specifies the user's private room while the fluent *available* is used to represent the availability of the user during the day. This availability is translated into PDDL in the form of timed initial literals (TILs) (Edelkamp & Hoffmann, 2004) by assigning the *available* predicate to true or false in specific time intervals. We also represent the known locations of the user during the day with TILs by assigning the fluent *at* to the corresponding location based on the user's activity locations. The user's preferences on attending games are represented by the fluents *att_min* and *att_max*. The fluent *not_assigned_game* is used to list all the games to which a user has not

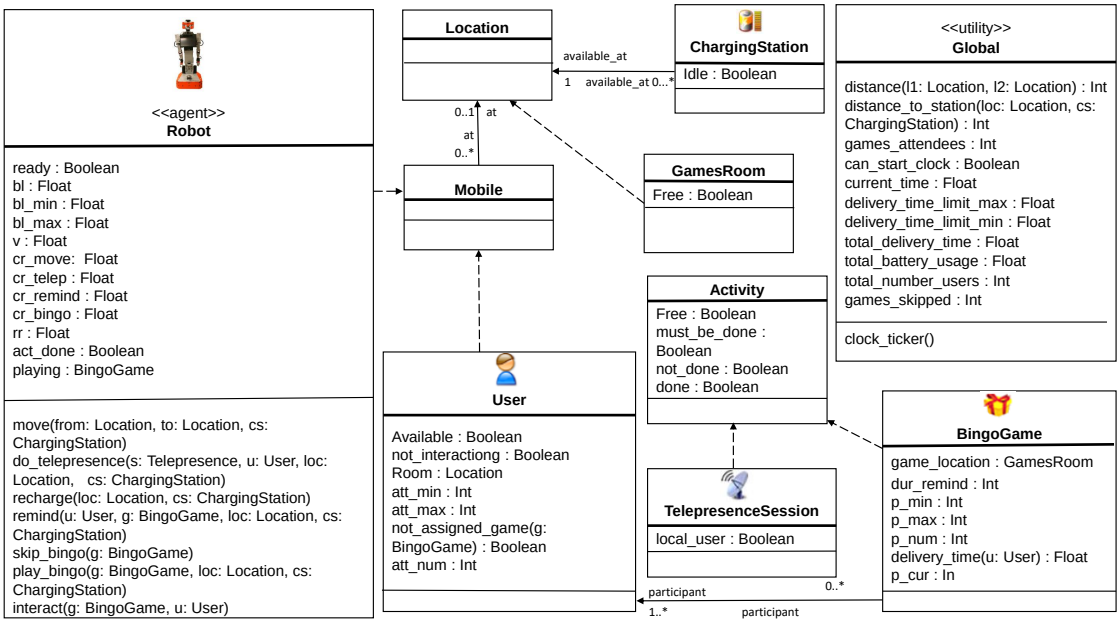

Figure 1: The UML Class Diagram of the first proposed problem model. Dashed lines represent an inheritance (e.g., Robot is a type of Mobile) and a solid line represents a relationship (e.g., a Mobile can be at a Location).

yet been assigned, and the fluent *participant* to specify the game to which the user has been assigned. We write *att_num* for the number of games planned for each user. Finally, when a user is interacting with a robot, the predicate *not_interacting* is set to false to prevent other robots from interacting with the same user.

The class *Robot* can also only be *at* (fluent) one location at a time and has all the properties (as fluents) detailed in the problem description (e.g., velocity, battery level, etc.). In addition, we have the fluents *ready*, *act_done*, and *playing*. A robot is *ready* when it is not engaged in any tasks and it is *playing* when it is performing a Bingo activity. The predicate *act_done* prevents a robot from going to a location and performing no action: a robot can only move to another location if it has completed a task in its current location.

The classes *TelepresenceSession* and *BingoGame* represent the HRI activities that need to be performed by the robots during the day. Both have the properties *dur*, to represent duration; *not_done* and *done*, to represent whether the task has been performed; and TILs *must_be_done_during*, to represent the time windows in which the task can be performed. In addition to the properties of the sessions and games introduced in the problem description, we have added the fluents *p_num* and *p_cur* to control the number of users reminded by the robots and the number of users playing the game, as well as *delivery_time* to control the time each user is reminded about the game.

Modelling the separation time between the delivery of a reminder and its associated Bingo game is done by using PDDL+ which includes *processes* (Fox & Long, 2006). A *process* (called *clock_ticker* in the class *Global*) models an exogenous activity that is triggered

for as long as a condition holds (in this case the fluent *can_start_clock*), regardless of the action selection process. This mechanism allows us to increment the fluent *current_time* 1 minute at a time, simulating the passage of time in discrete 1 minute intervals. If *current_time* is used in an action's precondition it will hold the exact start time of the action. We use this variable to record the time each user is reminded (fluent *delivery_time*) and also to check if the start time of a game is within the time constraints of the reminders.

The class *Global* also holds global variables including the maximum and minimum time for delivering reminders prior to the games (fluents *delivery_time_limit_min* and *delivery_time_limit_max*), the total time generated by adding all the lengths of the time intervals between the reminders and the game (fluent *total_delivery_time*), the total amount of battery power consumed by all robots (fluent *total_battery_usage*), the total number of games not played (fluent *game_skipped*), the total number of users attending games (*game_attendees*), and the number of target users (*total_number_users*). These variables are used to specify the cost function and are manipulated in the specification of the robot actions.

Operators. As shown in Figure 1, a robot has the following operators:

- *move* to a location
- *recharge* its battery
- *remind* a user
- *do_telepresence* with a user
- *play_Bingo* with a group of users
- *interact* with a player during the Bingo game
- *skip_Bingo* which removes the game from the request list.

We focus on the operators related to the Bingo activity given its modelling complexity. We present the details of the Bingo related operators (*remind*, *play_Bingo*, and *interact*) in Appendix A.

In the *remind* operator, a robot must be ready to perform the task and the user has to be available at the same location as the robot. As an effect of the operator, the user is set as a *participant* of the game. The time of the reminder is recorded in the fluent *delivery_time*, which will become a constraint (condition) for the Bingo operators.

In order to facilitate a game after the reminders, a robot has to first start the *play_Bingo* action, then it has to concurrently perform the *interact* action with each participant. The *play_Bingo* action can only finish when the robot has performed the *interact* action with all assigned players. The *interact* action is used to ensure that users are participating in the Bingo game for the duration of the game.

The passage of time in this model is managed through the PDDL+ process called *clock_ticker*. The PDDL code for *clock_ticker* in Appendix A shows how the fluent *current_time* gets updated in every tick of the planner's clock.

Goal and Objective Function. In the goal state, all sessions and games must be *done* (Bingo games can be either performed or skipped) and the user preferences on game attendance must be satisfied. We aim to minimize the following weighted cost function f:

$$f = 500(games\_skipped) + 1000(total\_number\_users - games\_attendees) + total\_delivery\_time + total\_battery\_usage, \quad (1)$$

where the weights are used to express preference on optimizing the number of games and players. In PDDL this cost function is represented as follows:

```
(:metric minimize
(+ (* 500 (games_skipped))
     (* 1000 (- (total_number_users) (games_attendees)))
     (total_battery_usage)
     (total_delivery_time)))
```

4.1.2 Alternative Modelling Strategies

The modelling possibilities for the target problem are numerous. We now present modelling strategies that result in five alternate PDDL models. We focus on requirements that are more complicated to model intuitively and efficiently in PDDL: the Bingo game activity requirements on the robot's interaction with participants and the constraint on the temporal separation between reminders and Bingo games. We essentially change the representation of these two aspects of the problem to obtain the alternative PDDL models. Altogether, we propose three strategies for the first aspect and two for the second resulting in six different models.

User-Bingo Interactions In the initial model, users play in Bingo games through the *interact* operation that is performed concurrently with the *play_Bingo* operation in order to individually model user interactions with the robot in the multi-user activity. We denote this initial strategy as *single*, as single users are considered for interaction. Two alternative strategies are presented that aim to explicitly model interactions with users in the *play_Bingo* operator. However, to explicitly represent the user interactions within the *play_Bingo* action, the number of users must be known. For example, if we assume exactly three participants the operator *play_Bingo3* may be defined as shown in Appendix A. The preconditions and effects of this operator are an amalgamation of both the *play_Bingo* and *interact* operators to include the user interactions in the *play_Bingo* operator. To improve the model further, symmetry breaking is used in the precondition (i.e., user i has an id less than user $i+1$) to reduce the available permutations.

As mentioned, we propose two alternatives to handle the user-Bingo interactions: *set-all* and *min-add*. The first, *set-all*, removes both the *play_Bingo* and *interact* operators and uses multiple *play_BingoX* operators where X is between 3 and 10 to account for all possible number of participants in a Bingo game. Essentially, this strategy sets all the participants in a Bingo game in the *play_Bingo* action. However, since we do not know the number of participants in advance, multiple operators with varying numbers of users are required. The second strategy, *min-add* is a combination of the two strategies that will only replace *play_Bingo* by *play_Bingo3*. By doing so, the Bingo games will start with

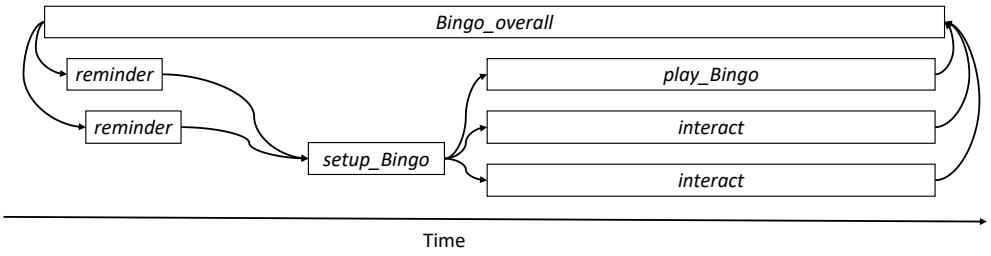

Figure 2: Example of a Bingo game with two participants. The *Bingo_overall* action encompasses all *reminder*, *setup_Bingo*, *play_Bingo*, and *interact* actions associated with the Bingo game. Here, the *setup_Bingo* action separates the reminders and the Bingo game to ensure that a minimum amount of time has passed. The length of the *Bingo_overall* action ensures that the separation of the reminders and the Bingo game is less than the maximum allowed time. We provide an intuitive representation of the influence of the preconditions and effects of each action through the use of precedence relationships (arrows) showing the relative ordering of actions.

the minimum number of players and then increase participation through *interact* actions with other users to add more players than the allowed minimum. The *min-add* strategy is therefore a combination of *single* and *set-all* by using the *play_Bingo3* action to facilitate a Bingo game with three users and potentially adding more participants using the *interact* action.

Separation Constraint The current PDDL model makes use of the process *clock_ticker* to keep track of the time passed between a reminder and a Bingo game. This strategy is denoted as *clock*. The proposed alternative, *envelope* is to make use of an encompassing larger action, *Bingo_overall* (detailed in Appendix A), that executes over all the Bingo related actions and must occur while any *remind*, *play_Bingo*, and *interact* actions are being executed (see Figure 2). This action acts as an envelope that spans all those actions to ensure the timing constraints are met and is similar to the model for durative sub-actions presented by Smith (2003). By setting the envelope action to be the appropriate duration (maximum delivery time plus duration of a Bingo game), a reminder cannot be separated from a Bingo game by more than the maximum delivery time. To ensure that reminders do not occur too close to a Bingo game, we introduce a new operator *setup_Bingo* with duration equal to the minimum separation time, which must occur between reminders and the Bingo game.

To use the proposed alternative strategy, new fluents are introduced. Each game can either be *Bingo_actions_ready* or *Bingo_actions_not_ready* based on whether *Bingo_overall* is being executed and any actions related to a Bingo game (*remind*, *play_Bingo*, and *interact*) has a prerequisite that a Bingo game has the fluent *Bingo_actions_ready* set as true. A fluent, *Bingo_game_ready*, is also required to enable the start of a Bingo game after *setup_Bingo*

has been performed. Finally, a *remind_enable* fluent is used to state when users can be reminded or not.

The *remind* operator is updated to require *remind_enable* to be true as a precondition and the *play_Bingo* requires *Bingo_game_ready* to be true. The *setup_Bingo* operator is presented in Appendix A.

Alternative Models The strategies proposed for both the user-Bingo interactions and the delivery time window constraints can be applied independently, resulting in six different models. One of the six models was already shown with single interaction actions being the sole method for users to play Bingo games and the use of processes to enforce separation constraints, *single-clock*. Table 1 presents an overview of the six models and the strategies they use.

Table 1: Alternative Models

User-Bingo Interaction	Separation Constraint	
	Clock Processes	Envelope
Single Interactions	*single-clock*	*single-envelope*
Set Min Then Add	*min-add-clock*	*min-add-envelope*
Set All Players	*set-all-clock*	*set-all-envelope*

4.1.3 Problem Modifications

The PDDL models discussed above correspond to the *BRPOF* problem definition. Here, we show the updates to the PDDL model to handle each of the five different modifications, *B*, *R*, *P*, *O*, and *F*.

B: Battery The removal of battery constraints in the model is straightforward. All fluents related to the battery are removed, specifically *bl*, *bl_min*, and *bl_max*. Any preconditions and effects that relate to any of these fluents are also removed so that the battery is no longer considered. In addition, the *recharge* operator is deleted from the model and the objective function is simplified to remove the battery component. These changes can be performed on all of the six models identically.

R: Remove Separation Constraints To handle the modification *R*, time constraints on the delivery time must be relaxed such that reminding a user at any time before a Bingo game is sufficient. To make this change, *delivery_time_limit_min* and *delivery_time_limit_max* must be updated to 0 and H, respectively, where H is the planning horizon minus the duration of a Bingo game. In the models that make use of *clock_ticker*, the separation time constraint in the precondition of *play_Bingo* and *interact* actions will change accordingly depending on the model. The models with the *Bingo_overall* operator will increase the duration of this action to extend over the entire planning horizon and will also remove the *setup_Bingo* operator. Since the *setup_Bingo* operator is no longer used, the fluent *Bingo_game_ready* is removed and a delete effect is added to *play_Bingo* to remove *enable_remind* to ensure that reminders still must occur before Bingo games.

P: Set Number of Participants We must treat the models differently to ensure that exactly four users participate in any game that is played. For the two models *single-clock* and *single-encompassing*, we can set $p\_min = p\_max = 4$ to force the planner to only consider plans where four *interact* actions are used for each Bingo game. The remaining four models will make use of only *play_Bingo4* operator and remove any other *play_Bingo* and/or *interact* operators.

O: Bingo Games are No Longer Optional We can ensure all games are played by removing the *skip_bingo* operator. The objective function can be simplified to remove the *games_skipped* component since all games will be played. This can be done for all six models.

F: Simplified Objective Finally, for modification *F*, one must only consider user participation of Bingo games in the objective. Furthermore, for the three models that use the *Bingo_overall* task to model the constraint on the separation time, the removal of the exact separation time required for use in the objective function means that it is possible to remove the clock processes entirely.

4.1.4 Modelling Issues and Limitations

An important challenge in modelling our problem in PDDL is dealing with time constraints between actions (i.e., time delays between reminder and Bingo games). To be able to model time constraints, we must model a clock as a process that is constantly incremented or make use of the larger encompassing task with required concurrency. If we use the processes, we must mark the start times of reminder activities and use this time to restrict the precondition of the Bingo games. Doing so allows the solver to verify that the two tasks are temporally consistent. However, few planners are able to handle this clock/process approach, limiting the solvers we can use.

Another issue is the flexibility in the number of participants of a Bingo game. In general, PDDL has the tools to model the Bingo attendees as a decision variable through the use of the feature *forall*, but the planners we tested do not seem to support *forall* along with other features that we need (e.g., *numeric, temporal, optimization,* etc.). Due to this limitation, we make use of two alternative approaches: 1) using required concurrency in which there is an action *play_bingo* that is a container for each individual interaction with a user and a robot, and 2) creating multiple duplicates of a single Bingo game, *play_BingoX*, one for each possible number of participants, *X*, in a game.

Finally, current STRIPS planners, in general, do not handle negative literals in preconditions. This leads to substantial redundancy as we must make use of many *not_* predicates to represent these negative preconditions.

4.2 Timeline-Based Planning and Scheduling

Timeline-based planning and scheduling differs from action-based planning as it represents the world in terms of a set of functions of time that describe how the world changes over a temporal interval (Cesta, Fratini, & Pecora, 2008).

The Extensible Universal Remote Operations Planning Architecture (EUROPA) system is a class library and tool set developed at the National Aeronautics and Space Administration (NASA) for building timeline-based planners and schedulers (Barreiro et al., 2012).

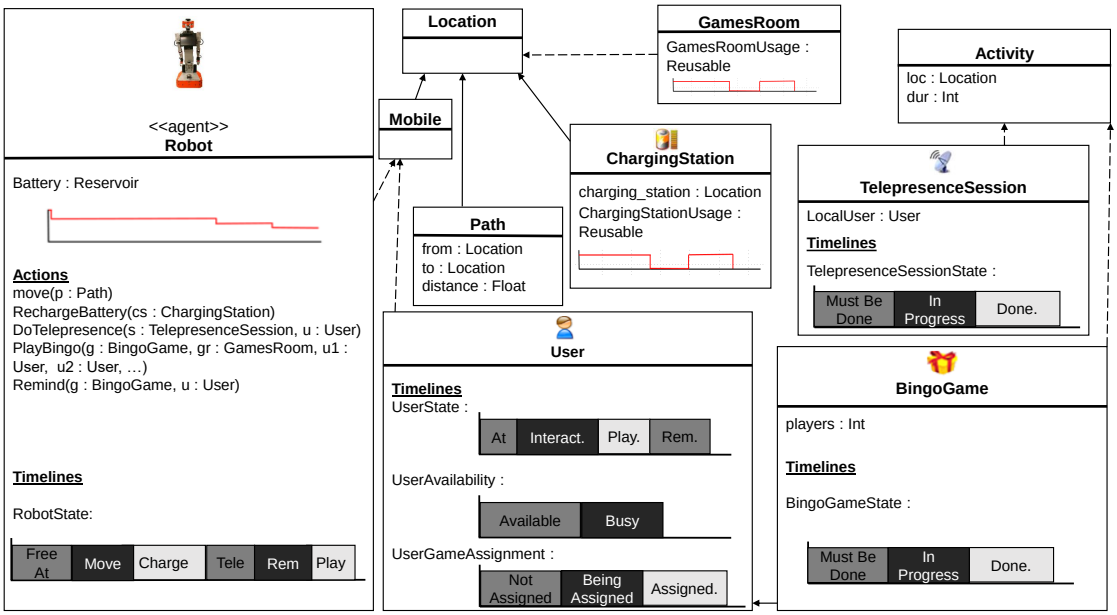

Figure 3: The UML Class Diagram of the proposed EUROPA model. Dashed lines represent an inheritance (e.g., Robot is a type of Mobile) and a solid line represents a relationship (e.g., a Mobile can be at a Location).

EUROPA represents a technology that appears to be a good fit for solving the problem of interest. Thus, we wish to explore this technology as a potential candidate. However, it is important to note here that we learned during our investigation of EUROPA that it was not made to fit within the model-and-solve paradigm that we wish to pursue, but rather requires customizing the solver and search heuristics.

EUROPA uses *New Domain Definition Language* (NDDL) as the main input modelling language. Like PDDL, NDDL uses state and activity descriptions. However, NDDL state variables are called timelines, temporally extended predicates that can be in a single state at any instant in time.

NDDL is object-oriented and so we represent most physical entities within the retirement home as objects. Figure 3 is the UML class diagram for the NDDL model. The objects used are very similar to those used in the PDDL model and so we present fewer details regarding each individual class, but emphasize the major difference between the PDDL and NDDL models: the addition of timelines and resources (reusable and reservoir) as first-class objects. We discuss the NDDL model here at a high-level. The encoding can be found in Appendix B.

4.2.1 The Environment

We use various classes to represent the static environment of the retirement home. Each of the rooms is an instance of a *Location* class and two such instances can be linked together by a *Path* which defines the distance between any two locations. A location may have a

ChargingStation that is represented as a reusable resource, *ChargingStationUsage*, to model the availability of the station over time.

4.2.2 Activities

Telepresence sessions and Bingo games are also represented with classes. Each telepresence session is associated with a particular user, location, and duration and has the timeline *TelepresenceSessionState* to indicate its state. The timeline has three values: *MustBeDone, InProgress*, and *Done*. At the beginning of the day, the state of the telepresence session is *MustBeDone*, indicating that the telepresence session has not yet been performed. Once a robot starts the action *DoTelepresence*, the *TelepresenceSessionState* timeline changes to *InProgress*. Upon completion of the telepresence session, the state changes to *Done*.

Bingo games have a particular location, duration and number of players associated with them. Similar to the *set-all* modelling strategy used for PDDL planning, we prescribe the number of players in the timeline-based model as a fixed value for the Bingo games. See Section 4.2.5 for details. Each Bingo game, like a telepresence session, has a timeline, *BingoGameState*, indicating the state of the Bingo game.

Recall that each activity has a time window in which the activity can be performed. These time windows are represented in the declaration of the initial state to indicate when the activity can be performed, similar to the representation of the user schedules.

4.2.3 Users

Users are represented by a *User* class. Each user has three associated timelines: *UserState*, *UserAvailability*, and *UserGameAssignment*. The *UserState* defines the state of the user, which can be either *At*, *Interacting*, *Playing*, or *BeingReminded*. While the user is not engaged with a robot, the *UserState* will be in the *At(l)* state, which is a parameterized state indicating that the user is at location *l*. *Interacting*, *Playing*, and *BeingReminded* are states indicating that the user is in a telepresence session, playing a Bingo game, and being reminded, respectively. The *UserAvailability* timeline indicates whether the user is *Busy* or *Available*. During the time a user is interacting with a robot or is at an appointment as per his/her personal schedule, the *UserAvailability* is *Busy*. The final user based timeline is *UserGameAssignment* which has three states: *NotAssigned*, *BeingAssigned*, and *Assigned*. Every user starts as *NotAssigned* and once a robot reminds a user, *UserGameAssignment* transitions to *BeingAssigned*. Upon playing a Bingo game, the *UserGameAssignment* will change to *Assigned* to indicate that the player has been assigned to and played in a game.

4.2.4 Robots

Robots are represented by a *Robot* class. Each robot has a *RobotState* timeline that indicates the state of the robot: *FreeAt, Moving, Charging, DoingTelepresence, PlayingGame*, and *Reminding*. Each robot also has a *Battery* resource, which is represented using a reservoir that can be consumed or replenished.

Robots have five actions: *Move, RechargeBattery, DoTelepresence, PlayBingo*, and *Remind*. Each action requires timelines to be in particular states and changes the state of the timelines. For example, *Move* changes the location of a robot from the current location at the start of the action, to a moving status during the action, and finally to the *destination*

location at the end of the action. *Move* requires the use of a *Path* between the current location and the destination location which provides the distance the robot must move, and therefore the duration of the movement and the battery usage. The Bingo related tasks for the NDDL model follow the same strategy as the *set-all* strategy of the PDDL models by handling all participation within the *PlayBingo* operator. The encoding of *PlayBingo* is presented in Appendix B, but we only present a three player Bingo game. Multiple *PlayBingo* games are used to allow the assignment of different numbers of players.

4.2.5 Modelling Issues and Limitations

We could not fully represent all aspects of the environment using EUROPA since our initial goal was to model and solve the problem without changing the solver code. While EUROPA is a very flexible and expressive package, significant effort and deeper knowledge is required to represent more complex components of our system.

The number of participants in a Bingo game is difficult to model since we must connect a participating user to the game. By leaving the number of users as a decision variable, we were not able to model which users were playing the games, while also ensuring that the number of participants in a game is within the required bounds. By fixing the number of users in a game to an appropriate size, we can model the interaction of users playing in Bingo games and ignore the provided bounds on the number of participants.

Furthermore, we are unable to represent an objective function. Although it is possible in EUROPA to optimize, it requires altering the solver. EUROPA has built-in backtracking, however, the backtracking rules and decision procedure must be coded to perform any optimization. Otherwise, EUROPA will return to a prior state by backtracking, but continue to make the same decision leading back to the state the system was in prior to backtracking. Without an objective function that allows the solver to reason about the optional Bingo games, we must also represent Bingo games like the telepresence sessions and make them mandatory; otherwise, Bingo games will not be played and only the telepresence sessions will be performed.

EUROPA leaves open many possibilities for those wishing to use timelines in their planning and scheduling problems. However, this flexibility comes at the cost of a more involved process while writing the code for modelling and solving the problem. Due to the requirement of a deeper understanding of the EUROPA architecture and code to fully express our problem, we limit the scope of the problem represented by timeline-based planning and scheduling in our study. Unfortunately, at this time, no solver exists that can handle NDDL in the model-and-solve approach, but rather, EUROPA is used as a domain-dependent solver with great benefits when employing customized search heuristics.

Another modelling language using a timeline approach, Action Notation Modelling Language (ANML), aims to combine strong notions of action and state (from PDDL), a variable/value model (from NDDL), and rich temporal constraints (from NDDL) (Smith, Frank, & Cushing, 2008). However, our investigation into ANML and the FAPE solver (Dvorak, Bit-Monnot, Ingrand, & Ghallab, 2014) led to the conclusion that, like NDDL, ANML is a good fit for representing the problem, but the FAPE solver has not yet been implemented

with efficient methods of handling TILs to be able to scale well for our problem of interest.[3] Therefore, we did not continue our study using ANML.

4.3 Constraint-Based Scheduling

Our target application combines characteristics that have typically been studied in planning with those more representative of scheduling problems. However, one of the major limitations of general constraint-based scheduling approaches is the requirement of a predefined set of tasks (Laborie, 2003). In planning, an operator dictates how the state of the world changes given its application. An operator is instantiated to create a ground action and the planner decides how many times to instantiate each operator and the sequence of actions required to reach a goal state. For example, to charge the battery on a robot, the planner has access to a charging operator that can be executed as many times as necessary. In contrast, typical constraint solvers require that we give, ahead of time, every potential charging task a distinct name. To address this issue, our general approach is to make use of optional tasks to model all the actions that the planner may choose to instantiate. This allows us to model a task but not necessarily execute it. Therefore, we must know the maximum number of possible charging tasks prior to scheduling in order to know how many optional tasks to create. The same must be done for the reminders and Bingo games since we do not know the assignment of users to a Bingo game or whether a game will be played or not a priori.

We introduce two constraint-based scheduling models in this section: one that is a single CP model and one that is a two-stage decomposition. The presentation of two CP models here is due to our preliminary results on the first CP model that showed promising directions for CP with minimal changes to include a simple decomposition that can perform significantly better.

Unlike PDDL and NDDL models that have a standardized format, CP solvers may vary widely. Thus, the CP models are presented using notation that is common in the CP literature. Below, we define the set of parameters relevant to our CP model.

Parameters:

M: set of reminder tasks,
M_{gu}: set of reminder tasks corresponding to user u and Bingo game g,
C: set of charging tasks,
CR_i: set of charging tasks corresponding to robot i,
CS_k: set of charging tasks corresponding to charging station k,
S: set of all telepresence session tasks,
G: set of all Bingo games,
A: set of all tasks $A = S \cup G \cup M \cup C$,
AU_u: subset of tasks in A that involve user u,
AR_i: subset of tasks in A that involve robot i,
$\bar{A}_j$: set of clone tasks of task a_j,
θ_j: Duration of task j, $j \in A \setminus C$.

3. Filip Dvorak, personal communication.

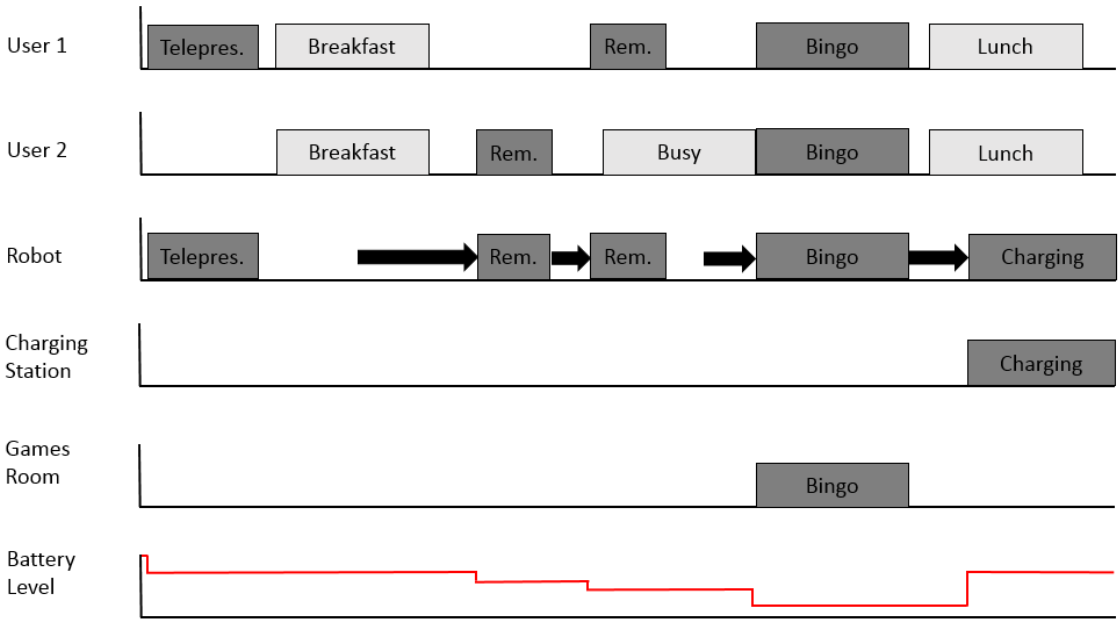

Figure 4: Gantt chart illustrating a sample schedule. Here, telepresence sessions and reminders are abbreviated as Telepres. and Rem., respectively.

4.3.1 Global-CP

Our first constraint-based scheduling model is a CP model we call Global-CP. We make use of IBM ILOG CPLEX CP Optimizer 12.6.2 (Laborie, 2009). Figure 4 is a Gantt chart that illustrates a sample morning schedule - a portion of a solution. Users, robots, charging stations, games rooms, and battery levels are resources that tasks use. Other than the battery level, all resources are unary capacity. The lighter shaded tasks are predefined appointments for a user and cannot be changed. The darker shaded tasks are those which the decision maker has control over and must be scheduled. Arrows in the robot's schedule represent the robot moving from location to location in order to perform the next task.

Variables and Domains. For each task $j \in A$, there is a corresponding interval variable a_j (Laborie, 2009). An interval variable is defined by a start time, end time, and size, which refers to the amount of battery power required to perform the task. Similar to the planning models, time in the CP model is represented in discrete 1 minute intervals and the battery power is able to take on real values. An interval variable can be either *absent* or *present*, which is indicated by the variable $presenceOf(a_j)$ equaling 0 or 1, respectively. If an interval variable is absent, it will not be considered by any constraint or expression. Each task has a number of *clone* tasks, which are required to model the alternative robots that can complete the tasks. For each task $j \in A$, there are $|R|$ additional tasks indexed by i and denoted by α_{ij}. Therefore, there are an additional $|A|$ sets of tasks denoted by $\bar{A}_j$, where $j \in A$ and $|\bar{A}_j| = |R|$.

For a telepresence session $j \in S$, the following domain restrictions apply:

$$presenceOf(a_j) = 1 \qquad \forall\{j \in S\} \quad (2)$$
$$length(a_j) = \theta_j \qquad \forall\{j \in S\} \quad (3)$$
$$forbid(a_j, calendar_j) \qquad \forall\{j \in S\}. \quad (4)$$

Constraint (2) enforces that each telepresence session must take place. Constraint (3) sets the duration of the telepresence session, which is known a priori. The entire duration of the task must be within the allowed time windows defined by $calendar_j$, a piecewise function over time equal to 1 during the times where both the telepresence session is allowed and the corresponding user is available. Otherwise, the function is equal to 0 and a task cannot be executed because of Constraint (4).

For tasks in G (Bingo games), the following domain restrictions apply:

$$presenceOf(a_j) \in \{0, 1\} \qquad \forall\{j \in G\} \quad (5)$$
$$length(a_j) = \theta_j \qquad \forall\{j \in G\} \quad (6)$$
$$forbid(a_j, calendar_j) \qquad \forall\{j \in G\} \quad (7)$$
$$p\_min_j \times presenceOf(a_j) \leq participants_j \leq p\_max_j \qquad \forall\{j \in G\}. \quad (8)$$

Constraint (2) is replaced with Constraint (5). Unlike telepresence sessions, Bingo games are optional. However, if a game is played, Constraint (8) provides bounds on the number of participants assigned to the game. Unlike the calendar for the telepresence sessions, we do not know which users will partake in the game. Therefore, the calendar will only consider the time windows when the Bingo game can be played and not the schedules of the user. We ensure user availability when we assign users to Bingo games (see below regarding *clone* Bingo games).

To model the reminder tasks, we must first define an upper bound on the number of possible reminders that can occur during a day. Since every user should be considered for each game, we have a set of reminder tasks M_{gu} for each Bingo game g and user u. The size of M_{gu} depends on the number of distinct locations at which a user u is available for reminders during the course of the day. The requirement for the multiple tasks representing the same reminder is necessary to model the movement of the robot to the various locations of the retirement home.[4] For these tasks $j \in M_{gu}$, the following domain restrictions apply:

$$presenceOf(a_j) \in \{0, 1\} \qquad \forall\{j \in M_{gu}, g \in G, u \in U\} \quad (9)$$
$$length(a_j) = \theta_j \qquad \forall\{j \in M_{gu}, g \in G, u \in U\} \quad (10)$$
$$forbid(a_j, calendar_j) \qquad \forall\{j \in M_{gu}, g \in G, u \in U\} \quad (11)$$
$$0 \leq del\_time_j \leq 120 \qquad \forall\{j \in M_{gu}, g \in G, u \in U\}. \quad (12)$$

Similar to the Bingo games, reminders are optional (in the sense that a reminder does not occur if a user does not play). However, we know the user that is being reminded for each task, so we can again update the task calendar to intersect the intervals of availability for games and the user. We further make use of a decision variable del_time_j to represent

4. For further details on the modelling requirements and limitations of CP, see Section 4.3.3.

the time between the delivery of the reminder and the start of a Bingo game. Although reminders are restricted to be between 15 and 120 minutes before a Bingo game starts, we allow the range of delivery times to be between 0 and 120 minutes to allow for the possibility that a reminder may not occur and the delivery time can be set to 0. We elaborate on the relationship of del_time_j and the reminder and Bingo game later in this section when we present the constraints of the model.

In order to link users to the Bingo games they play, we create a set of *clone* Bingo game tasks $\bar{G}$. $\bar{G}$ consists of $|G| \times |U|$ tasks that, like the Bingo games, are optional interval variables with the same domain restrictions shown in Constraints (5) to (8). However, Constraint (7) is changed to have a different $calendar_j$ function that is the intersection of the time window(s) available for the Bingo game and the user's personal schedule. We provide further details in the section on constraints (below) to illustrate how to link these interval variables to the Bingo games and assignment of users to the games.

The last set of tasks are the charging tasks C. Each charging task is associated with a specific robot. Therefore, C comprises of $|R|$ different sets denoted as CR_i, where $i \in R$. Each robot will have a set of charging tasks equivalent to the upper bound of the number of possible charging tasks. This upper bound is calculated by allowing the robot to charge before each potential task, the sum of the number of telepresence sessions, Bingo games, and maximum number of games each user is interested in playing. The following domain restrictions apply:

$$presenceOf(a_j) \in \{0, 1\} \qquad \forall\{j \in CR_i, i \in R\} \quad (13)$$

$$0 \leq length(a_j) \leq \frac{bl\_max_i}{rr\_i} \qquad \forall\{j \in CR_i, i \in R\}. \quad (14)$$

The charging tasks are optional tasks. Furthermore, the duration of a charging task may range from 0 time units to the maximum battery level of the robot divided by the recharging ratio of the robot, and is left as a decision for the solver.

Each task has an associated energy consumption decision variable, e_cons_j, $j \in A$ that represents the energy consumed to perform the particular task for the determined duration of that task and the energy required to move from the robot's previous location to the location of the task. The domain restriction of this variable is:

$$e\_task_j \leq e\_cons_j \leq e\_task_j + max\_dist_j \times max\_cr\_move \qquad \forall\{j \in A \setminus C\} \quad (15)$$

$$-bl\_max \leq e\_cons_j \leq max\_dist_j \times max\_cr\_move \qquad \forall\{j \in C\}. \quad (16)$$

The value e_task_j represents the minimum amount of energy required to process a task j. We use the minimum energy consumption over all robots (recall that robots have different rates of consumption) since the assignment of tasks to robots is not known a priori. As well, since the sequence of tasks is not known, the travelling distance of a robot is not known either. Therefore, we use the farthest location to the location of task j, max_dist_j, times the maximum consumption for moving over all robots, max_cr_move. In the case that the job j belongs to the set of charging tasks, the energy consumption has a domain that ranges between the maximum battery level over all robots, bl_max, and the maximum energy used for travelling to the charging location. The values are negative to signify a production of energy rather than a consumption. However, the value may be positive since travelling can

take up more energy than the charging task produces. Such a domain definition ensures that any possible value for e_cons_j is within the search space of the model.

In order to model user preferences about Bingo attendance, each user has a decision variable defining the number of games played during the day. The domain restriction on this variable is:

$$min\_att_u \leq games\_attended_u \leq max\_att_u \qquad \forall\{u \in U\}. \quad (17)$$

Finally, we have an auxiliary task signifying the start of the schedule for each robot $i \in R$, $\dot{a}_i$. These auxiliary tasks have domain:

$$presenceOf(\dot{a}_i) = 1 \qquad \forall\{i \in R\} \quad (18)$$

$$length(\dot{a}_i) = 0 \qquad \forall\{i \in R\} \quad (19)$$

$$start(\dot{a}_i) = 0 \qquad \forall\{i \in R\}. \quad (20)$$

Cumulative Functions: Cumulative functions are piecewise functions over time with discrete value changes made at the start and end times of interval variables (Laborie, 2009). Interval variables can have one of two effects on cumulative functions, a *step* effect or a *pulse* effect. A *step* effect can affect the cumulative function at the start or end of the interval variable and will permanently increase or decrease the cumulative function value. A *pulse* effect will increase (decrease) the cumulative function at the start of the interval variable, but then decrease (increase) the cumulative function by the same amount at the end. We make use of the cumulative function to represent various resources in the system such as: robots, users, charging station, and battery. Note that there is a difference between the use of calendars, which are static 0/1 step functions over time that are used to define the disjoint time-windows of tasks, and cumulative functions, which are dynamic piecewise functions that can change when tasks are performed.

For each robot $i \in R$, there is an associated cumulative function rc_i. Each task assigned to a robot will make use of the robot for the entire duration of the task. Upon completion of a task, the robot as a resource is released and can perform other tasks. Thus, tasks have a *pulse* effect on the robot adding 1 to the cumulative function:

$$rc_i = \sum_{j \in AR_i} pulse(\alpha_{ij}, 1) \qquad \forall\{i \in R\} \quad (21)$$

$$rc_i \leq 1 \qquad \forall\{i \in R\}. \quad (22)$$

Since a robot is a unary capacity resource, rc_i must never exceed 1.

A user is also considered as a unary capacity resource. For each user $u \in U$, we have an associated cumulative function uc_u, where all tasks pertaining to that user must also exhibit a *pulse* effect on the user resource:

$$uc_u = \sum_{j \in AU_u} pulse(a_{uj}, 1) \qquad \forall\{u \in U\} \quad (23)$$

$$uc_u \leq 1 \qquad \forall\{u \in U\}. \quad (24)$$

Here, we denote AU_u as the set of all interval variables (tasks) associated with a user u. These tasks include all telepresence sessions and reminder tasks for that particular user, as well as all the *clone* Bingo games in $\bar{G}$ for user u.

A charging station can only handle one robot at a time. Therefore, we have a charging station cumulative function chc_k for each station k where:

$$chc_k = \sum_{j \in CS_k} pulse(a_j, 1) \qquad \forall\{k \in K\} \qquad (25)$$

$$chc_k \leq 1 \qquad \forall\{k \in K\}. \qquad (26)$$

Robot battery levels are also a limited resource that we model with the cumulative function. Each robot $i \in R$ has a cumulative function re_i representing the battery level of the robot over time. Unlike the previous cumulative functions where the resource is released upon completion of a task, the battery level will remain changed after a task is completed. Therefore, we make use of the *step* effect. Given that a robot only performs one task at a time and charging cannot occur while other activities are being executed (unlike robots that can harvest energy (i.e., solar, vibration) during operations), the model can be represented with a step occurring either at the start or end of the interval variable. Although in reality battery power is consumed continuously during the event, modelling total consumption at the start or end of an interval variable will adequately represent our system since doing so ensures that there is sufficient energy to complete the task. We will use the *stepAtStart* effect to have energy consumption occur at the start of an interval variable:

$$re_i = \sum_{j \in A} stepAtStart(\alpha_{ij}, -e\_cons_j) \qquad \forall\{i \in R\} \qquad (27)$$

$$bl\_min_i \leq re_i \leq bl\_max_i \qquad \forall\{i \in R\}. \qquad (28)$$

Note that the size of the step will be the total energy consumption of the task. If the task is a consuming task, then e_cons_j is positive and we will subtract the amount consumed from the robot battery level. If the task is a charging task, e_cons_j is negative and we will add the amount of energy to the robot's battery. Constraint (28) provides bounds on the battery level of the robot at all times.

Interval Sequences: Interval sequences are defined on a set of interval variables whose values are constrained to form a total ordering (Laborie, 2009). Absent tasks are ignored in the sequence. Each robot $i \in R$ is associated with an interval sequence variable rs_i on the set of interval variables for tasks in AR_i. This variable has a value that is a permutation of all present variables for tasks of AR_i. The interval sequence variable contains all tasks that might be assigned to a robot including the auxiliary start task. Furthermore, each interval variable ar_j, $j \in AR_i$ in an interval sequence rs_i is given a non-negative integer type $T(rs_i, ar_j)$ that indicates the location of a task. A transition matrix, Δ_i, that represents the travel time between any two locations for a robot i can then be used in conjunction with the interval sequence variable to model the movement of the robot within the retirement home. More details are provided in the next section

Constraints. The CP model includes all possible tasks that may occur. In addition, each of the tasks has *clone* tasks that signify the assignment of the task to a robot. We link these tasks with an *alternative* constraint, which has the form $alternative(a_j, \bar{A})$ and is used to ensure that if a task a_j is present in the schedule, then exactly one other task from the set of

tasks $\bar{A}$ must also be present. Since we have our main set of tasks in A that must be linked to the cloned tasks in $\bar{A}_j$ to decide which robot executes a task, we use the constraint:

$$alternative(a_j, \bar{A}_j) \qquad \forall\{j \in A \setminus C\}. \tag{29}$$

Here, $\bar{A}_j$ represents the set of clone tasks, such that all tasks corresponding to clones of task a_j are contained in $\bar{A}_j$. Therefore, if the task $j \in A \setminus C$ is present, then one of the clone tasks must also be present. The clone tasks have a specified robot and will act as an assignment of the task to that robot. We do not need to consider the charging tasks C as each of these tasks pertain to a specific robot and no assignment is necessary.

We also restrict the values of a reminder task and set the delivery times based on the corresponding Bingo game. These constraints are:

$$presenceOf(a_j) \leq presenceOf(a_g) \qquad \forall\{g \in G, u \in U, j \in M_{gu}\} \tag{30}$$
$$start(a_j) \leq start(a_g) - 15 \qquad \forall\{g \in G, u \in U, j \in M_{gu}\} \tag{31}$$
$$start(a_j) \geq start(a_g) - 120 \qquad \forall\{g \in G, u \in U, j \in M_{gu}\} \tag{32}$$
$$del\_time_j = presenceOf(a_j) \times [start(a_g) - start(a_j)] \qquad \forall\{g \in G, u \in U, j \in M_{gu}\}. \tag{33}$$

Constraint (30) states that if a Bingo game is not played, the corresponding reminders are not executed. Constraints (31) and (32) are the delivery time constraints that ensure a reminder that is performed must be within the required time prior to a Bingo game. Constraint (33) sets the delivery time of a reminder to be the difference between the start of the Bingo game and the start of the reminder. If the reminder does not occur, because the user does not play in that game or a different reminder linked to another location is used, then the delivery time will be set to 0.

To ensure that a reminder occurs and a user is present at a Bingo game when a user is assigned to a game, we add the constraints:

$$\sum_{j \in M_{gu}} presenceOf(a_j) = presenceOf(a_{\bar{g}}) \qquad \forall\{g \in G, u \in U, \bar{g} = \bar{G}_{gu}\} \tag{34}$$
$$start(a_g) = start(a_{\bar{g}}) \qquad \forall\{g \in G, u \in U, \bar{g} \in \bar{G}_{gu}\} \tag{35}$$
$$length(a_g) = length(a_{\bar{g}}) \qquad \forall\{g \in G, u \in U, \bar{g} \in \bar{G}_{gu}\}. \tag{36}$$

Here, $\bar{G}_{gu}$ represents the clone Bingo game task for user u and Bingo game g. Constraints (35) and (36) then set the start time and length of the clone Bingo games to be equal to the actual Bingo games.

The number of users playing in a Bingo game is calculated as:

$$\sum_{u \in U} \sum_{j \in M_{gu}} presenceOf(a_j) = participants_g \qquad \forall\{g \in G\}. \tag{37}$$

Since Constraint (34) ensures the assignment of a user to a Bingo game when a reminder is made, we can use the reminders to count the participation of a user in a Bingo game. An alternative method to count the number of users in a Bingo games is to use the clone Bingo games. We found experimentally that there is no significant difference in performance when using either constraint.

In addition to the participation for each Bingo game, we must also calculate the number of games a user plays during the day. For each user $u \in U$, we use the reminder tasks again to count the user's participation in any of the Bingo games:

$$\sum_{g \in G} \sum_{j \in M_{gu}} presenceOf(a_j) = games\_attended_u \qquad \forall\{u \in U\}. \tag{38}$$

Lastly, we must handle the variables related to charging. To deal with the symmetry between the charging tasks, we have the constraint:

$$presenceOf(a_j) \leq presenceOf(a_{j'}) \qquad \forall\{k \in K, j, j' \in CS_k | j < j'\}. \tag{39}$$

To assign the energy consumption values, e_cons_j, we must know the sequence of tasks for a robot so that we can calculate the energy required to travel from the location of one task to that of the next task. We can handle the sequencing of tasks such that no two tasks are executed by a robot at the same time and the time between any two tasks is at least as much time as needed for the robot to travel between the two locations of the tasks. To do so, we use the *NoOverlap* constraint:

$$NoOverlap(rs_i, \Delta_i) \qquad \forall\{i \in R\}. \tag{40}$$

The *NoOverlap* constraint ensures that no tasks overlap each other. We define Δ_i as a square matrix with element $\Delta_i(l, h)$ being the time that is required for robot i to travel between locations l and h. Since rs_i maintains a vector of all tasks that robot i may perform and the corresponding locations between those tasks, *NoOverlap* will ensure that, based on the locations of the tasks, the time interval defined by the matrix Δ_i must occur between any two consecutive tasks. The energy consumption of a task is then:

$$\begin{aligned} e\_cons_j = {} & presenceOf(a_j) \times \Delta_i(prevLoc(rs_i, a_j), loc_j) \times cr_{move} \\ & + length(a_j) \times cr_j \qquad \forall\{i \in R, j \in AR_i\}. \end{aligned} \tag{41}$$

The function $prevLoc(rs_i, a_j)$ returns the location of the task directly before task a_j in the sequence of rs_i. Therefore, the first term of the right hand side is the energy required to move the robot between locations. The second term is the energy required to perform the task. Since the robot has different consumption rates for different tasks (telepresence sessions, Bingo games, reminders, charging), cr_j is used to define the particular rate of a task j.

Finally, at the end of the day, each robot should return to the charging station in order to charge its battery. We model this by using the constraint:

$$re_i = step(H, -bl\_max_i + bl\_min_i) \qquad \forall\{i \in R\}. \tag{42}$$

The *step* constraint makes a change at the time indicated in the first parameter, H, to the cumulative function re_i of $-bl\_max_i + bl\_min_i$. This represents a reduction in the battery level of a robot from the maximum charge to the minimum charge. By choosing a sufficiently

large H such that the daily schedule is completed and all robots have enough time to return to a charging station and recharge to a full battery level, we guarantee that the robot will always end the schedule at a charging station and the last task it will perform is a charging task. An example of a value for H can be,

$$H = maxT + maxD + |R| \times maxC \qquad \forall\{i \in R\}, \tag{43}$$

which is equal to the sum of the time at the end of the day, $maxT$, the maximum amount of time required for any robot to travel from anywhere in the retirement home to a charging station, $maxD$, and the maximum amount of time required to charge every robot in sequence, $|R| \times maxC$.

Objective Function. The objective function is:

$$\begin{aligned} minimize \sum_{g \in G} 500[1 - presenceOf(a_g)] + 1000 \left[|U| - \sum_{u \in U} games\_attended_u \right] \\ + \sum_{j \in M} del\_time_j - \sum_{j \in C} e\_cons_j. \end{aligned} \tag{44}$$

The objective is a multi-criteria objective that aims to maximize the total number of games played, participation in games, and minimize the total time between a reminder occurring and a Bingo game, and the total battery consumption.

4.3.2 Problem Modifications

The CP model presented above is for the *BRPOF* problem definition. Here, we show the updates to the CP model in order to handle each of the five different modifications, *B*, *R*, *P*, *O*, and *F*.

B: Battery Battery consideration can be removed by deleting Constraints (13)-(16), (25)-(28), (39), (41), and (42). The removal of the relevant battery related variables and cumulative functions is required. Finally, the objective function is updated to remove the criterion of minimizing battery usage.

R: Remove Separation Constraints To handle the modification *R*, time constraints on the delivery time must be relaxed such that reminding a user at any time before a Bingo game is sufficient. To make this change, Constraints (12), (31), and (32) are removed and the constraint,

$$start(a_j) \leq start(a_g) \qquad \forall g \in G, u \in U, j \in M_{gu} \tag{45}$$

is added to ensure reminders occur before Bingo games.

P: Set Number of Participants To set Bingo game participation to exactly 4, we just need to update $min\_att_u = max\_att_u = 4$.

O: Bingo Games are No Longer Optional We can ensure all games are played by updating Constraint (5) to force the presence of all Bingo game interval variables rather than allow them to be optional.

F: Simplified Objective Finally, for modification *F*, one must simply remove all components of the objective function other than the counter for the user participation.

4.3.3 Modelling Issues and Limitations

Our investigation of CP shows that modelling the problem requires many alternative tasks to represent a single task. Rather than having one interval variable to represent a task, clone tasks are needed to discern the robot that processes it and to model the possibly differing energy requirements of each robot. Thus, the number of tasks considered in the model will multiply based on the number of robots.

Another modelling difficulty is the representation of robots and users moving through the environment. Some activities have defined locations, but some HRI activities require the robot and user to be in the same location for interaction. Due to the limitations of CP for representing user movements over time, we introduce multiple interval variables of the same activity, each with a predetermined location. That is, if a user needs to be reminded for a Bingo game, a reminder task must be created for every potential location that user may be in. Without these additional tasks, we cannot model the travel time and energy costs of a robot. However, such a solution is not as elegant as the planning representation that uses actions, since all potential locations of robot tasks must be predetermined and considered in the CP model.

Finally, using the CP technology requires us to determine an upper bound on the number of each task that might occur. As mentioned earlier, we must know the maximum number of battery recharges and reminders that can be performed to generate the model. To ensure that we have a sound and complete model, the number of tasks to include may be grossly overestimated, leading to a model with many superfluous decision variables and slower runtimes.

4.3.4 Decomposed-CP

Smith, Frank, and Jónsson (2000) compared planning and scheduling, and indicated that one of the weaknesses of scheduling is the inability to adequately handle environments with cascading effects. An action with a cascading effect changes the system state and leads to requirements for one or more other actions. For example, if a user is to play a Bingo game, he or she must be reminded of the game. However, if a user is not participating in a Bingo game, the reminder should not take place. Due to such dependencies, the scheduling model becomes very large because we create alternative interval variables for every possible action. Based on preliminary results testing Global-CP, we propose a decomposition of the CP model, named Decomposed-CP, that attempts to improve upon the Global-CP model and better handle the cascading effects in this system.

The decomposition is comprised of two stages: a master problem and a sub-problem. The master problem is the Global-CP model for the *BRPO-* problem variant. Recall that *BRPO-* only simplifies the objective function. The sub-problem then uses the solution of the master problem to fix certain values of the schedule and solves the complete problem under these restrictions using CP. By choosing to fix the right decision variables in our sub-problem, we are able to reduce the difficulty of handling actions with cascading effects.

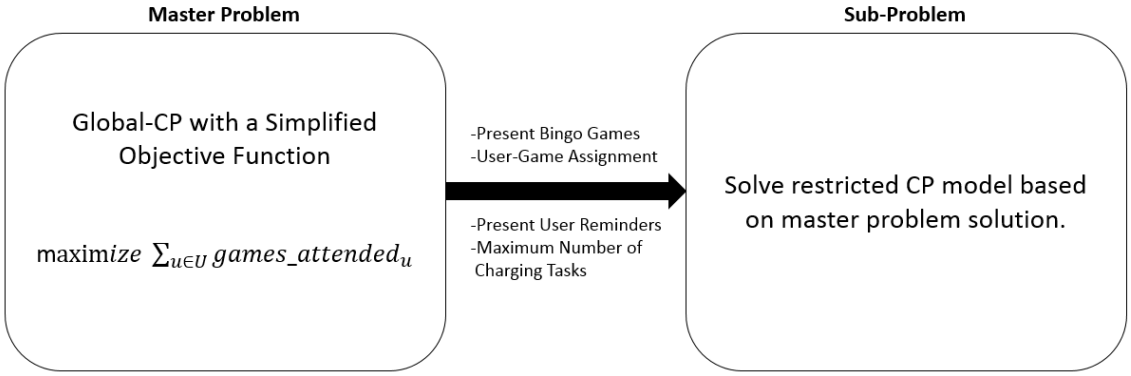

Figure 5: Brief overview of the Decomposed-CP model.

Figure 5 is an illustration of the two stages of the decomposition and shows the problem components that are passed from master problem solution as restrictions to the schedule in the sub-problem.

The proposed decomposition only considers the maximization of user participation in Bingo games in the master problem objective function. That is:

$$maximize \sum_{u \in U} games\_attended_u. \tag{46}$$

The minimization of the number of games played, battery consumption, and delivery times of reminder tasks are ignored in the master problem, but the constraints regarding the battery and delivery times are still enforced. The solution to the master problem will be a valid solution to the complete robot scheduling problem, however, the schedule may be of poor quality since most of the objective function is ignored.

The solution of the master problem gives us an assignment of users to Bingo games that will be used in the sub-problem. Games that are not played in the master problem are removed and not considered in the sub-problem and games that were played in the master problem are available to be played in the sub-problem with the same players, but are not fixed to start at any specific time. We further set the upper bound for the number of charging tasks per robot to the total number of charging tasks across all robots in the master solution. We choose this upper bound to allow for some flexibility in changing the schedule of recharging robots, while guaranteeing that a feasible schedule exists. The objective function of the sub-problem is the original objective function. Note that this decomposition potentially loses global optimality since the optimal assignment of games and the number of recharges from the master problem might not be optimal when the other parameters of the objective function are considered.

When using the Decomposed-CP model, a decision must be made as to when to switch from solving the master problem to the sub-problem. The most straight-forward approach is to solve the master problem to optimality and then solve the sub-problem with the remaining time. In practice, we found that this approach works well. However, if the

master problem is too difficult, it is possible that all the computation time will be spent on the master problem. Alternatively, one could set a time limit for the master problem that is shorter than the total time limit and switch when either the master problem has been solved to optimality or when a time-limit has been reached; whichever occurs first. We choose to use this latter strategy with at most half of the total time limit available for solving the master problem. However, for the experiments we present in Section 5, the optimal master problem solutions are all found and proved before the time limit is reached.

Based on our preliminary results with Global-CP on problem variation *BRPO-*, we know that the Decomposed-CP model will find solutions significantly faster than the Global-CP model for the original problem and for most variations. Global-CP has difficulties with the complex objective function while the simplified objective function is tractable for the problem sizes found in our application. By first solving the master problem, we find a schedule with a high quality on the most important objective (user participation) that is used to restrict decisions in the sub-problem to reduce the cascading effects of actions and limit the number of optional interval variables while optimizing the original objective function.

4.3.5 Modelling Issues and Limitations

Our decomposition may not find the optimal solution if the master problem assignment does not result in the optimal Bingo game assignments with at least as many charging tasks as necessary to achieve the optimal solution for the complete problem. However, if obtaining solutions quickly with some emphasis on solution quality is important, the Decomposed-CP model may be a valuable technique to apply. Recent work on an approximate logic-based Benders decomposition method (Burt, Lipovetzky, Pearce, & Stuckey, 2015) presents a similar style of decomposition to ours to solve a mining application problem.

Although we believe that using a sophisticated decomposition technique, such as logic-based Benders decomposition (Hooker & Ottosson, 2003) or branch-and-check (Thorsteinsson, 2001), could lead to stronger performance and the guarantee of optimal solutions given sufficient time, the work is non-trivial in comparison to our proposed decomposition. Furthermore, from our experiments, it is not clear that the problem structure allows for a decomposition such as logic-based Benders to be used, given the difficulty of finding optimal solutions for even relaxed constraint-based scheduling problems.

5. Experimental Study

We consider a retirement home environment in which residents undertake several activities in different locations (e.g., TV room, private room, garden, dining hall) during a day. We assume that each user has four one-hour non-interruptible activities (e.g., physiotherapy, doctor's appointment, family visit, nap), in addition to the meal times, during which he/she cannot be disturbed. Other, interruptible, activities (e.g., walk in the garden, read in a common area) allow robot interactions. At least one interruptible activity is assumed for each user. We analyze the proposed models for five full-day scenarios in this environment (7am-7pm), see Table 2. These scenarios represent the requirements of the retirement home, but with varying number of users and robots from a fairly small retirement home to ones that are comparable to the actual retirement homes we are working with. Possible

Table 2: The number of objects in the five scenarios.

Scenario	Users	Robots	Telepresence	Bingo
1	5	2	2	1
2	10	2	4	2
3	15	3	6	3
4	20	3	8	4
5	25	4	10	5

	7am-8am	8am-9am	9am-10am	10am-11am	11am-12pm	12pm-1pm	1pm-2pm	2pm-3pm	3pm-4pm	4pm-5pm	5pm-6pm	6pm-7pm
User 1	personal care	BREAKFAST	exercise class		Garden	LUNCH	art class				DINNER	movie night
User 2				physiotherapy			Tv room	nurse appoint.		music class		personal care
User 3	personal care		family visit				art class					movie night
User 4	personal care			walk				physiotherapy	Tv room			movie night
User 5	personal care		exercise class	Garden	physiotherapy					music class		

Figure 6: Example user schedules over a single day. Blue tiles indicate when a user is busy with a personal activity, red tiles are meal times, green tiles represent the interruptible activities, and white tiles are leisure periods of time when the users are in their own personal rooms and are available to interact with robots..

user schedules were obtained from healthcare professionals at our collaborative retirement homes. Figure 6 is an example schedule for five users over the course of a single day.

In all scenarios, the telepresence sessions and Bingo games are 30 and 60 minutes long, respectively, with time windows from 8am-7pm.[5] Reminders are 2 minutes long. For all models, we use a discrete representation of time in increments of 1 minute. Each game has a minimum of three participants and a maximum of ten participants. Every user is willing to attend at most one Bingo game during the day (i.e., $att\_min = 0$, $att\_max = 1$). All robots have the following property values, estimated based on the Tangy robot: $bl\_min = 0$, $bl = bl\_max = 20$, $v = 20m/min$, $rr = 0.5$, $cr\_move = 0.04$, and $cr\_telep = cr\_remind = cr\_Bingo = 0.1$.

We run each model on the five scenarios using a 64-bit Ubuntu Linux machine with 12 GB of memory. We use the OPTIC planner (Benton, Coles, & Coles, 2012) to solve the PDDL planning model. Preliminary experiments tested five different planners: COLIN (Coles, Coles, Fox, & Long, 2012), LPG-td (Gerevini, Saetti, Serina, & Toninelli, 2004), OPTIC (Benton et al., 2012), POPF (Coles, Coles, Fox, & Long, 2010), and SGPlan (Hsu & Wah, 2008). Of the five planners, only COLIN, OPTIC, and POPF were able to find feasible plans for the smallest scenario tested.[6] OPTIC was found to be the best performing solver of the five. The CP models are solved using IBM ILOG CPLEX CP Optimizer 12.6.2. A one-hour timeout was used for each model in each scenario. We measure the runtime, the number of users attending a game, and the solution quality based on the objective function of problem *BRPOF*.

5. Bingo games are not allowed before 8:00 a.m. as such an early Bingo game is undesirable, even though the daily schedule starts at 7:00 a.m..
6. LPG-td and SGPlan cannot handle PDDL+ processes and required concurrency, while POPF and COLIN support required concurrency but not processes.

Table 3: Performance of the proposed models on problem *BRPOF*. The best results over all six PDDL planning models for each scenario is presented. A (-) indicates that no solution was found.

Model	Scenario	Runtime (s)		Participants		Objective Value	
		first	last	first	last	first	last
PDDL Planning	1	0.74	2,033.72	3	4	2,147.10	1,124.11
	2	0.68	2,062.94	0	0	11,012.37	11,012.23
	3	57.16	1,960.02	0	0	16,518.65	16,518.63
	4	2.18	23.84	0	0	22,024.92	22,024.92
	5	7.24	89.46	0	0	27,557.30	27,554.54
Global-CP	1	0.08	9.26	0	5	5,623.00	192.00
	2	1.96	33.84	6	9	4,549.00	1,243.00
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
Decomposed CP	1	0.05	0.23	5	5	486.00	192.00
	2	0.08	67.49	6	10	5,039.00	847.00
	3	0.36	37.87	4	15	12,296.00	1,213.50
	4	0.41	2,567.80	10	20	12,107.00	1,430.50
	5	0.37	3,382.83	4	25	23,494.50	1,929.00

For a given problem modification, all solvers search an equivalent solution space and objective function. During the one-hour time limit, the first and last solution found using each of the models was recorded. The only objective function that is not calculated based on the full objective is for problem variation *-RPOF*, where the battery usage criteria is ignored. Thus, the objective function used in problem variation *-RPOF* removes the battery usage and only includes the user participation, Bingo games played, and delivery time components. As such, the objective value for *-RPOF* will seem better than it should be if the same schedule were to be applied to other problem variations as there will be an increase in cost from battery usage (assuming that the schedule is also feasible when battery consumption is considered).

Table 3 presents our results for the five scenarios using the different solvers on problem *BRPOF*. We provide the best results, based on the final objective value score, over the six different PDDL planning models for each of the five scenarios individually. That is, the PDDL results represent the virtual best over all PDDL models. PDDL planning is able to find feasible plans for all scenarios, however, Bingo games are only played in Scenario 1. CP Optimizer has problems finding even feasible solutions for larger scenarios when using the Global-CP model. However, for the scenarios where solutions are found, Bingo games are played and the user participation is high. CP Optimizer with the Decomposed-CP model is

able to find high quality solutions for all scenarios and is consistently the best performing method.[7]

In the rest of this section, we present our experiments for each methodology. We show the performance of these approaches under various problem modifications and discuss the different approaches in more detail.

5.1 PDDL-Based Planning

Table 4 presents the results of running the OPTIC planner on all six models on problem *BRPOF*. Overall, we see that good quality solutions, meaning high user participation in Bingo games, are only found for the smallest scenario. All models that make use of the larger action that acts as an envelope over all Bingo related activities are unable to find feasible solutions for Scenarios 2-5. Introducing the required concurrency between the *Bingo_overall* action and *remind*, *setup_Bingo*, and *play_Bingo* gives the planner difficulty as the problem size increases. Thus, the *envelope* modelling strategy does not scale well.

We also see worse performance from *set-all* when compared to *single* and *min-add*. The problem with *set-all* is the large number of grounded *play_Bingo* actions in the larger scenarios. Although *min-add* also has many grounded actions, the total number of grounded *play_Bingo* actions is significantly fewer. Only the *single* modelling strategy will scale the number of grounded *interaction* actions linearly with the number of users and Bingo games. The other alternatives will require $\binom{N}{X}$ grounded actions for each *play_BingoX* operator, where N is the total number of users. Thus, even for strategy *min-add* with $X = 3$, scenarios with large N will be intractable as the planner fails to find a feasible solution within the time limit.

Table 5 illustrates the planning model performances on the *BRPO-* problem, where the objective function has been simplified. This problem provides insights into the behavior of the planning solver and shows the best performance we are able to obtain from the planner over all problem modifications. The planning models generally behave as in Table 4, except for *single-envelope*. The larger scenarios are still not solved, but high quality solutions are found for Scenarios 1-3. The change that allows for better performance here is the removal of the *clock-ticker* process completely, since *BRPO-* is not concerned with the delivery time in the objective function and the *envelope* modelling strategy handles separation constraints by using the *Bingo_overall* action. As we expect, grounding many actions for *min-add* and *set-all* is still very difficult for the larger scenarios, but under the *single-envelope* model, it is possible to obtain solutions for up to the medium-sized scenarios.

Full results for all six PDDL models and seven problem modifications can be found in Appendix C in Tables 9 - 14. In general, we observe that OPTIC has a difficult time finding solutions with active Bingo games. Removal of battery consideration and separation constraints helps the planner, but does not lead to a significant improvement. At best, these problem modifications led to Bingo games being played for Scenarios 1 and 2 rather than just Scenario 1. Interestingly, we found that for most models, the planner has a harder time when Bingo games are forced to be played and in fact, more often than not, fails to find

7. Recall that due to the limitations of the timeline-based planner and scheduler as outlined in Section 4.2.5, we are not able to model the problem fully and therefore no solutions are obtained for the problem.

Table 4: Performance of PDDL planning on the *BRPOF* problem. A (-) indicates that no solution was found.

Problem	Scenario	Runtime (s) first	Runtime (s) last	Participants first	Participants last	Objective Value first	Objective Value last
single-clock	1	0.04	786.12	0	3	5,506.13	2,070.75
	2	0.18	0.88	0	0	11,019.38	11,016.66
	3	0.84	9.38	0	0	16,530.31	16,525.73
	4	2.18	23.84	0	0	22,044.46	22,043.11
	5	7.24	89.46	0	0	27,557.30	27,554.54
min-add-clock	1	0.06	2,950.88	0	3	5,506.13	2,066.51
	2	0.68	2,062.94	0	0	11,012.37	11,012.23
	3	57.16	1,960.02	0	0	16,518.65	16,518.63
	4	143.36	143.36	0	0	22,024.92	22,024.92
	5	-	-	-	-	-	-
set-all-clock	1	0.74	2,033.72	3	4	2,147.10	1,124.11
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
single-envelope	1	0.84	1,287.07	3	3	2,152.33	2,077.94
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
min-add-envelope	1	0.84	1,287.07	3	3	2,152.333	2,077.94
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
set-all-envelope	1	0.76	943.56	3	3	2,152.33	2,077.94
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-

any solutions when Bingo games cannot be skipped, even though the same model is able to find a solution with a Bingo game when the *skip_Bingo* operator is included.

When Bingo games are restricted to have exactly four participants, the planner is unable to find a solution for Scenario 1 with any Bingo games except for the *single-envelope* model. Lastly, the problem *B—F* can help the performance as the solver is able to find plans with

Table 5: Performance of PDDL planning on the *BRPO-* problem. A (-) indicates that no solution was found.

Problem	Scenario	Runtime (s) first	Runtime (s) last	Participants first	Participants last	Objective Value first	Objective Value last
single-clock	1	0.06	75.94	0	3	5,612.46	2,761.64
	2	0.18	0.18	0	0	11,019.38	11,019.38
	3	0.82	0.82	0	0	16,530.31	16,530.31
	4	2.18	2.18	0	0	22,044.46	22,044.46
	5	7.04	7.04	0	0	27,557.30	27,557.30
min-add-clock	1	0.06	2,459.74	0	3	5,506.13	2,161.64
	2	0.82	0.82	0	0	11,019.38	11,019.38
	3	0.82	0.82	0	0	16,518.65	16,518.65
	4	146.72	146.72	0	0	22,044.46	22,044.46
	5	7.04	7.04	0	0	27,557.30	27,557.30
set-all-clock	1	0.46	602.40	3	4	2,226.30	1,617.03
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
single-envelope	1	0.26	1,297.10	3	5	2,138.96	530.36
	2	1,190.36	1,190.36	9	9	2,680.88	2,680.88
	3	3,406.20	3,406.20	15	15	1,089.24	1,089.24
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
min-add-envelope	1	0.22	1,269.80	3	4	2,261.10	1,620.54
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
set-all-envelope	1	0.42	721.80	3	4	2,261.14	1,620.54
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-

two games in Scenario 2 for the *single-clock* model. However, the other models do not see any improvements.

5.2 Constraint Programming

In this section, experimental results of both the Global-CP and the Decomposed-CP models using CP Optimizer are presented.

5.2.1 Global-CP

Table 6 presents the results from running the CP solver on the different problem modifications using the Global-CP model. For problem *BRPOF*, CP is only able to find solutions for Scenarios 1 and 2. With more than 10 users and 2 robots, CP cannot obtain feasible solutions within the one-hour time limit. CP can only obtain solutions to bigger problems (Scenarios 3-5) when we remove batteries (B), the reminder time bounds (R), or the complex objective function (F). Of those three properties, the reminder time bounds (R) has a much smaller effect as *B-POF* can only solve up to Scenario 3. CP performs well on *-RPOF*, *BRPO-*, and *B—F*. In all three of these problems, CP obtains solutions for all five scenarios.

Problems *BRP-F* and *BRPO-* provide some interesting and unexpected results. We assumed that *BRP-F* would lead to better performance since the development of constraint-based scheduling technology has not been heavily focused on dealing with as many optional tasks as are present in our model. Although a majority of optional tasks still exist in the *BRP-F* problem, we expected a significant improvement of performance, which was not observed in our experiment. In fact, for Scenario 1, we see that CP Optimizer takes twice as long to find the best found solution for problem *BRP-F* compared to *BRPOF*. This result suggests that optional tasks are not as hard to handle for CP Optimizer as we initially assumed. In contrast, changing the objective function to only consider user participation leads to CP finding solutions quickly (within fractions of a second) and with maximum user participation. This is against expectation as one of the main advantages of CP is its strong inference methods, which we believe the removal of the complex objective function would not affect. More specifically, we would have expected potentially stronger performance when altering the objective function in regards to optimizing schedules, but the fact that even finding satisficing schedules became significantly easier was not expected.

Table 6 gives us insight into the problem components that are difficult for the CP model. It is clear that battery level considerations and complex objective functions lead to the majority of the issues when trying to find even feasible solutions. When either of these are removed, CP does not have trouble quickly obtaining solutions with everyone playing Bingo games.

5.2.2 Decomposed-CP

Table 7 provides the performance of the decomposition model for the different problem modifications. The first solution presented is the first feasible schedule found by the first stage of the Decomposed-CP model. Note that the first stage represents only a simple objective function. For example, if the problem to be solved is *BRP-F*, the master problem would be *BRP–* and the subproblem is *BRP-F*. Furthermore, recall that the first stage is a sound model and therefore produces globally feasible solutions.

The decomposition is able to find schedules with the maximum number of possible participants for every scenario and in general to do so very quickly. Note that for problem variations *B—F* and *BR-OF*, the maximum number of participants is limited by the restriction on the number of users in a game, which is less than the total number of users. These results were expected after observing the performance of Global-CP, since, as we men-

Table 6: Performance of Global-CP on all tested problem modifications. A (-) indicates that no solution was found.

Problem	Scenario	Runtime (s)		Participants		Objective Value	
		first	last	first	last	first	last
BRPOF	1	0.08	9.26	0	5	5,623.00	192.00
	2	1.96	33.84	6	9	4,549.00	1,243.00
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
-RPOF	1	0.08	0.15	5	5	223.00	178.00
	2	0.11	48.73	10	10	807.00	276.00
	3	0.46	337.67	14	15	2,352.00	359.00
	4	0.91	3,040.75	20	20	2,006.00	450.00
	5	1.66	2,870.58	24	25	4,646.00	550.00
B-POF	1	0.08	41.75	5	5	1,745.00	192.00
	2	0.36	256.49	8	10	4,885.00	346.00
	3	2,230.32	2,230.32	15	15	5,176.00	5,176.00
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BR-OF	1	0.11	2,297.28	4	4	1,447.00	1,129.50
	2	0.49	925.55	8	8	2,817.00	2,179.50
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRP-F	1	0.40	18.83	5	5	499.00	192.00
	2	0.87	1,399.25	8	10	2,937.00	305.50
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRPO-	1	0.05	0.05	5	5	486.00	486.00
	2	0.08	24.22	6	10	5,039.00	847.00
	3	0.36	37.87	4	15	12,296.00	2,013.50
	4	0.41	1,076.5	10	20	12,107.00	2,522.00
	5	0.37	103.33	4	25	23,494.50	3,033.50
B—F	1	0.27	28.88	4	4	1,387.50	1,129.50
	2	0.18	2,022.39	8	8	4,144.00	2,262.00
	3	25.83	3,590.05	12	12	5,366.50	3,428.00
	4	1,162.26	1854.93	16	16	7,309.00	4,589.00
	5	1,813.50	3501.50	20	20	9,473.50	5,960.50

Table 7: Performance of the Decomposed-CP model on all tested problem modifications. The first solution that is recorded is based on the solution found from the first stage of the Decomposed-CP model.

Problem	Scenario	Runtime (s)		Participants		Objective Value	
		first	last	first	last	first	last
BRPOF	1	0.05	0.23	5	5	486.00	192.00
	2	0.08	67.49	6	10	5,039.00	847.00
	3	0.36	37.87	4	15	12,296.00	1,213.50
	4	0.41	2,567.80	10	20	12,107.00	1,430.50
	5	0.37	3,382.83	4	25	23,494.50	1,929.00
-RPOF	1	0.26	0.29	5	5	473.00	178.00
	2	0.01	0.09	0	10	11,000.00	364.00
	3	0.04	170.31	0	15	16,500.00	678.00
	4	0.06	1.06	0	20	22,000.00	783.00
	5	0.08	12.64	0	25	27,500.00	723.00
B-POF	1	0.27	0.49	5	5	1,397.00	192.00
	2	0.06	36.16	6	10	4,471.00	642.50
	3	0.89	608.66	3	15	13,188.00	1,587.00
	4	0.58	2,739.24	6	20	16,493.00	1,614.00
	5	0.39	2,179.80	6	25	21,522.00	1,978.00
BR-OF	1	0.26	0.76	4	4	1,373.00	1,129.50
	2	0.06	7.95	4	8	6,517.00	2,264.50
	3	0.14	76.15	4	12	12,130.00	3,471.00
	4	0.22	838.98	8	16	13,784.00	4,826.00
	5	0.44	562.93	4	20	23,075.00	6,381.50
BRP-F	1	0.34	0.18	5	5	503.00	192.00
	2	0.47	291.48	8	10	2,937.00	305.50
	3	0.75	3,559.85	9	15	6,298.00	627.00
	4	1.05	2,468.11	12	20	8,564.00	817.00
	5	1.66	2,880.74	15	25	11,090.00	2,242.50
BRPO-	1	0.05	0.05	5	5	486.00	486.00
	2	0.08	24.22	6	10	5,039.00	847.00
	3	0.36	37.87	4	15	12,296.00	2,013.50
	4	0.41	1,076.5	10	20	12,107.00	2,522.00
	5	0.37	103.33	4	25	23,494.50	3,033.50
B—F	1	0.60	12.89	4	4	2,530.00	1,172.00
	2	0.08	3,249.35	8	8	4,720.00	2,454.50
	3	0.49	767.91	12	12	4,285.00	3,771.00
	4	30.31	3,470.31	16	16	8,933.00	4,933.00
	5	4.03	2,398.54	20	20	10,408.00	6,381.50

tioned earlier, the first stage of the decomposition is just the Global-CP applied to problem *BRPO-*. Once a partial schedule is found with decisions made regarding the presence of a Bingo game and its players, the problem becomes significantly easier as many actions with cascading dependencies are eliminated. There is a large reduction in the number of charging tasks (about 98%) and reminder tasks (between 80% and 96%). As we saw with the Global-CP model, battery level considerations make the problem hard and removal of just that aspect of the problem led to substantial performance improvements.

5.3 Best Performance Results

Table 8 presents the best performance of each approach when considering all the *sound* problem modifications: *BRPOF*, *BR-OF*, *BRP-F*, and *BRPO-*. The quality of the solutions generated are compared using the objective function for the original problem. We choose the problem that leads to the best results for each method and compare these results. Furthermore, for the planning models, since there are six different models, we choose what we consider the best performing model based the ability to obtain better objective values (*single-envelope*) under the best performing problem modification. For the planning solver and Global-CP, the best performance is found in problem *BRPO-*, whereas Decomposed-CP works best with *BRP-F*.

Each of the three methods is able to produce schedules for the robots for the system sizes that we tested, but if we choose the planning model that finds the most user participation in Bingo games, the planner is unable to find feasible solutions for the larger scenarios. The planning solver was found to perform worse than the two CP approaches even for scenarios where solutions are found. Global-CP, with a simpler objective function, is overall better than planning, also with a simpler objective function, except in Scenario 3. By abstracting the objective and ignoring certain components that are considered of lesser importance, Global-CP becomes a much more attractive option. The Decomposed-CP model, which has the Global-CP strengths on the *BRPO-* problem by design, is the superior choice as the components ignored in the objective initially are re-introduced into the sub-problem.

6. Discussion

Our overall project requires research and development on multiple fronts. This study considers the issue of which technology to adopt for the planning and scheduling components of our system. Based on the results that we obtained, CP is currently the more suitable technology within the confines of our modelling paradigms.

While all three technologies can be further improved with intelligent modelling choices and search guidance, we believe the results we present demonstrate the baseline performance one achieves when following standard modelling approaches for the different technologies along with current state-of-the-art solvers. Unfortunately, none of the technologies tested can satisfactorily solve the problem without some compromises. The issues we found indicate the research directions needed to be further explored in order to adequately handle our application as well as ones similar to it.

Table 8: Performance of the proposed models using the best modifications. A (-) indicates that no solution was found.

Model	Scenario	Runtime (s) first	Runtime (s) last	Participants first	Participants last	Objective Value first	Objective Value last
Planning (BRPO-) (*single-envelope*)	1	0.26	1,297.10	3	5	2,138.96	530.36
	2	1,190.36	1,190.36	9	9	2,680.88	2,680.88
	3	3,406.20	3,406.20	15	15	1,089.24	1,089.24
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
Global-CP (BRPO-)	1	0.05	0.05	5	5	486.00	486.00
	2	0.08	24.22	6	10	5,039.00	847.00
	3	0.36	37.87	4	15	12,296.00	2,013.50
	4	0.41	1,076.5	10	20	12,107.00	2,522.00
	5	0.37	103.33	4	25	23,494.50	3,033.50
Decomposed-CP (BRP-F)	1	0.34	0.18	5	5	503.00	192.00
	2	0.47	291.48	8	10	2,937.00	305.50
	3	0.75	3,559.85	9	15	6,298.00	627.00
	4	1.05	2,468.11	12	20	8,564.00	817.00
	5	1.66	2,880.74	15	25	11,090.00	2,242.50

6.1 PDDL-Based Planning

As noted in our discussion of the limitations and issues with PDDL planning, representation of temporal constraints is a challenge. While we made substantial use of the calendar in CP, enforcing temporal consistency with the temporal representation in PDDL (e.g., TILs) is difficult and the current approaches do not have an intuitive representation as compared to simple temporal networks and scheduling models (Cushing & Kambhampati, 2007; Marzal, Sebastia, & Onaindia, 2014). In addition Cushing and Kambhampati (2007) state that most temporal planners are incomplete due to restrictions of action start times to a small set of time points called *decision epochs*. Nonetheless, using *decision epochs* drastically improves computational performance and allows improved techniques from classical planners to be directly used for temporal planners. It is clear from our results that the use of the *clock* process to capture the temporal separation has significant negative impact on performance. Thus, improvement on handling processes more efficiently is a fruitful direction.

The performance of and accessibility to model features in planners can be improved. One example is the unavailability of the *forall* feature in conjunction with other features, greatly limiting our modelling options. This is an issue noted in the literature for most state-of-the-art planners (Bajada, Fox, & Long, 2014). Even with the features we could use, like *optimization*, we found that the planner was generally not able to significantly improve solutions. Benton et al. (2012) discuss the limitations of optimization in planners and propose the OPTIC planner for optimization of continuous models. Our experiments illustrate that planners still have much to gain from additional efforts in this area. Most

of our final solutions found using OPTIC were similar in quality to the first solution found and those plans were of very poor quality. To address problems similar to ours, planners must develop improved abilities to solve problems with temporal reasoning and concurrency (Cushing & Kambhampati, 2007; Jiménez, Jonsson, & Palacios, 2015), numeric values (Coles, Coles, Fox, & Long, 2013; Ivankovic, Haslum, Thiébaux, Shivashankar, & Nau, 2014), and optimization (Benton et al., 2012).

6.2 Timeline-Based Planning and Scheduling

We believe EUROPA presents a step in the right direction in temporal representation as the NDDL language provides a more intuitive representation of temporal characteristics via timelines. As mentioned in Section 4.2, ANML is an alternative language similar to NDDL that makes use of timelines. However, both modelling languages share the same problem of having minimal support from solvers capable of handling the full set of language features. While both are strong candidates for use, the current barrier for timeline-based planning and scheduling to be competitive with PDDL-based planning and constraint-based scheduling is the state of current solvers. With additional efforts towards the development of solvers, timeline-based approaches will be a useful technology for problems similar to ours.

6.3 Constraint-Based Scheduling

While the decomposition-based CP model delivered satisfactory performance on our test set, there is still room for improvement. Unlike the planning representation, it is not currently possible to model operators that may be instantiated multiple times. Laborie (2003) notes this issue when comparing the planning and scheduling literature. Beck and Fox (2000) extended propagation algorithms to handle alternative activities that do not necessarily have to be executed in a schedule. Laborie and Rogerie (2008) introduced a framework for reasoning about conditional *time-interval* variables that we used to address the issue of planning operators. However, this approach is not scalable and is a limitation in how we can currently model operators with CP. A valuable direction for researchers in constraint-based scheduling is the development of a method to model and solve problems where these high-level, reoccurring tasks exist. By being able to dynamically add tasks during problem solving, the size of the model can be substantially reduced (Barták, 2002). Yet, significant progress towards being able to competently handle reoccurring tasks for complex environments in CP has not been made. The closest work that we are aware of is the CPT planner, which combines partial order causal link branching with CP to obtain a strong pruning mechanism that dynamically introduces actions in the CP model (Vidal & Geffner, 2006). However, this functionality came at the cost of substantial customization of the underlying CP solver.

CP also lacks the expressivity of planning and timeline-based planning and scheduling. For example, the move action of the robot is represented in the CP models as transition times between actions. However, given that the distance traveled by a robot is dependent on the location of the previous and next actions and that action locations can depend on the time at which they are executed (e.g., users move around during the day), there is no straightforward or efficient manner to model robot movement. We are able to model such

movement by creating alternative tasks for each activity that corresponds to the potential locations of a user, but this approach will not scale well and is only possible because our problem does not include a large number of possible user locations. For similar robot scheduling problems in a larger environment, the representation for robot movement used here may result in much poorer performance.

Furthermore, the current state of CP solvers does not as easily allow for experimentation between different solvers like PDDL-based planning and timeline-based planning and scheduling that have a standardized modelling language. Each CP solver uses a different modelling language to represent the same model. MiniZinc, a CP modelling language supported by many CP solvers, addresses this problem, but has some compromises since each CP solver has a wide range of capabilities that are not fully supported by MiniZinc (Nethercote et al., 2007). Although this issue does not affect the solver performance, it does make the process of switching to other solvers more problematic in CP than with the alternative technologies.

Finally, it was interesting to see that the complex objective function was so difficult for the CP solver. We believe a deeper analysis and understanding of how CP behaves under particular models, objectives, and constraints would be useful. The current state-of-the-art in CP optimization techniques is either hybridization with linear programming (Heinz, Ku, & Beck, 2013a; Heinz, Schulz, & Beck, 2013b; Laborie & Rogerie, 2016) or cost-aware constraints (Focacci, Lodi, & Milano, 2002; Simonis & Hadzic, 2011; Régin, 1999). Insights into the behavior of these techniques for optimization can help improve our model to better handle our application.

6.4 The Effect of Modelling

Although we investigated multiple PDDL and CP models for our application, it is impossible to exhaust the complete model space to establish the best possible model for each technology. It is clear that modelling decisions can greatly impact solver performance, but the choice of formulation is non-trivial. This problem is further confounded as the performance of a model can depend on the solver that is chosen, requiring a user to have a strong understanding of a specific solver's inner workings in order to create efficient models. These issues are true for all technologies we study. Neither AI planning nor CP has developed solver-independent and formal ways of comparing alternative models for the same problem in the way, for example, that the study of polyhedral theory has for mixed-integer linear programming (Bertsimas & Weismantel, 2005; Schrijver, 1998). Nonetheless, to make progress on solving interesting problems, modelling decisions must be made and conclusions drawn from experimental studies.

There is not much work that deeply explores the study and development of modelling approaches and strategies in either PDDL planning or CP scheduling. The current approach to modelling in planning and scheduling is based strongly around personal expertise and trial and error. Given the impact that different models or model refinements can have on the performance of solvers, it is important that one develops a strong model in order to be able to draw proper conclusions (Riddle, Holte, & Barley, 2011; Vaquero, Silva, & Beck, 2010). It would be of substantial value to users of these technology to have studies that develop principles and practices of modelling. There is a particular research area

dedicated to the modelling techniques and principles for planning and scheduling problems: Knowledge Engineering for Planning and Scheduling (KEPS), a workshop that has been at the International Conference on Automated Planning and Scheduling since 2008. Yet, to the authors' knowledge, a detailed body of work does not exist that explores the impact of modelling in planning and scheduling.

6.5 AI Planning vs. Constraint Programming

Within the narrow context of our robot application, the results in Section 5.3 suggest that CP is the more promising approach to our problem. However, it is not our purpose, nor would it be appropriate, to make general conclusions about the relative problem solving abilities of the technologies that we investigated. At the most, given the very sparse application of CP technology to robot planning and scheduling problems and the contrastingly larger application of AI planning (e.g., the series of PlanRob workshops and robotics tracks at the *International Conference on Automated Planning and Scheduling*), our results suggest that CP is an interesting technology to investigate for this domain.

A more intriguing comparison arises from the question of the knowledge permitted in a model. In domain-independent planning, the modeller seeks to represent the "physics" of a problem without representing what has been generically termed "search control knowledge" (Bonet & Geffner, 2001; Hoffmann & Nebel, 2001). Although domain-independent planning comprises of the mainstream of AI planning, domain-configurable planning, which uses a domain-independent search engine with domain control knowledge, also exists in the literature. For example, TLPlan (Bacchus & Kabanza, 2000) and TALPlan (Kvarnström & Doherty, 2000) make use of control rules to prune the search space while hierarchical task network planners such as O-Plan (Tate, Drabble, & Kirby, 1994) and SHOP2 (Nau et al., 2003) use domain-specific knowledge for decomposing tasks into subtasks. It is also worth mentioning the work done by the planning and learning community who are looking to learn search control knowledge for planning automatically (de la Rosa, Jiménez, Fuentetaja, & Borrajo, 2011; Krajanský, Hoffmann, Buffet, & Fern, 2014) (see also the learning track of the international planning competition). The justification for the restriction of search control knowledge is that human planners are able to solve problems without being given such domain-specific rules. If we are truly seeking to develop an artificially intelligent planner, then supplying such rules defeats our purpose as well as requiring extra effort by the modeller.

In contrast, the primary tools for modelling in CP are precisely the addition of search control knowledge in the form of redundant constraints, dominance rules, and dual variables (Smith, 2006). For example, from a domain-independent planning perspective, the derivation of an upper bound on the number of recharging actions of a robot (see Section 4.3) so that we can even *represent* the problem is search control knowledge. As noted above, the CP community also has the goal of declarative problem solving (Freuder, 1997), but has taken a different path, allowing modellers to add search control knowledge with the justification being that through the study of such knowledge, as initially derived by modellers, we will eventually be able to automate its derivation. Such automated modelling and model reformulation studies form an active part of the CP research (Frisch, Harvey, Jefferson, Martínez-Hernández, & Miguel, 2008; Nightingale & Rendl, 2016), see the series of Mod-

Ref workshops at the *International Conference on Principles and Practice of Constraint Programming*.

It is beyond our purposes here to go further into this contrast which has historically been a subject of debate in the AI planning community (Hendler, Tate, & Drummond, 1990; Hewitt, 1971; Wilkins, 1984). However, returning to our application-driven motivation, *from the narrow perspective of solving application problems*, the domain-independent AI planning field would seem to be operating at a self-imposed disadvantage compared to CP by maintaining domain independence. Of course, if the restriction helps to more quickly understand and achieve the broader goals of truly intelligent agents, the disadvantage may be worthwhile.

7. Future Work

There are a number of avenues of subsequent work arising from our study.

7.1 Advanced Decomposition Models

We considered various PDDL planning models and a CP decomposition model. Decompositions are commonly found in the constraint-based scheduling community but less so in the planning literature. It would appear valuable to incorporate decomposition into the PDDL planning models. However, from the empirical results, it is not apparent how one should decompose the problem for planning. Using a similar decomposition as CP would not greatly improve performance as the best performance over all models tested for PDDL planning was only able to find solutions with user participation for up to Scenario 3. By using such a decomposition, only poor quality solutions will be generated for Scenarios 4 and 5. Alternatively, one could explore a hybrid decomposition where the master problem is solved by CP as in Decomposed-CP and the sub-problem with planning. One could potentially see the benefits of being able to use the *set-all* modelling strategy, but without the explosion of grounded actions as user participation would be fixed in the sub-problem.

7.2 Disturbances

Beyond this study, we plan to consider a more complex system where external robot- or user-related events cause disturbances that prevent a daily schedule from being completed properly. For example, a robot-related disturbance can be a failure to arrive at a planned destination due to obstacles blocking its path or unexpectedly low battery level. User-related disturbances exist since a person may not be located where the robot believes him/her to be. Additionally, he/she may decide not to participate in an HRI activity and so we must alter the schedule accordingly. Disturbance consideration is essential for ensuring that robots can operate in a dynamic environment and that users will have a positive experience with and confidence in the robots. In this current study, disturbances could be handled naively by generating a new plan starting at the time of any disturbance using the models presented in this paper. However, a deeper study is required to fully understand and apply the models developed here to the application problem where disturbances are a serious concern. We hope to build techniques to identify if and when the current schedule is no longer executable due to disturbances. Once such disturbances are identified, schedule repair, and in extreme

cases replanning and rescheduling, must be done so that the robots can continue to operate in the environment. We hope to evaluate and extend work in the planning literature that looks at generating plans to handle environments with exogenous events (Fritz & McIlraith, 2008; Muise, Beck, & McIlraith, 2013; Muise, Belle, & McIlraith, 2014).

To repair, replan, and/or reschedule, we will again explore the planner and CP technologies in addition to other technologies. For example, Markov decision processes have been used by the planning community (Feldman & Domshlak, 2014; Ross, Pineau, Paquet, & Chaib-Draa, 2008), but were not used in this paper. Due to the complexity of our environment, the state representation (that must include temporal and numeric states) would lead to a very large state-space. However, given pre-generated schedules, it may be interesting to see if there is potential to use Markov decisions processes to replan when disruptions occur.

8. Conclusion

We propose nine models using three technologies for the multiple robot, retirement home environment: six PDDL-based planning models, a timeline-based planning and scheduling model, and two constraint programming models. The many properties of the problem together create a complex problem to solve. Not only must the robots manage their battery power, but they must also choose sequences of tasks such that an efficient path is followed. In addition, tasks have time constraints and users have their own personal schedules. Based on numerical experiments, we find that CP, in particular a decomposition model using CP, is the most suitable technology to use in our system.

CP is better equipped than PDDL planning for handling optimization, but can struggle to find feasible schedules for larger problems. Although we found that CP performs favourably when considering optimization, it is interesting to note that we discovered the best approach is to only consider a simple objective function. Although the large number of optional tasks used in the CP model is also a culprit for the poor performance, we suspect for similar problems with complex objective functions, it is easier and likely just as effective to decompose a CP model to handle the objective function in stages rather than to try and remove or reduce the optional activities. Of course, it may be necessary to introduce both strategies in order to adequately deal with even more difficult problems.

Our study of the various PDDL-based planning models suggest the importance of understanding modelling decisions. The tests of different methods to represent interaction within Bingo games shows that one must be conscious of the inclusion of required concurrency and the number of grounded actions generated by a particular model. Decisions that introduce a trade-off between these aspects can be the difference between a planning model that easily finds feasible solutions, but with poor quality, and a model that does not find feasible solutions for larger problems, but has improved quality for smaller problems.

Due to the complex reasoning that must be applied to handle all the aspects of the problem, no solution technique is the perfect choice. This paper illustrates some of the limitations of the current techniques in planning and scheduling to deal with the problem of interest as well as the difficulties in making comparisons between these technologies. All aspects of this problem have been studied in the two fields of research, but together, these problem characteristics prove difficult for the available solvers. We believe that this

problem provides an interesting application that tests the existing planning and scheduling technology and hope that this paper might spur research on the unsolved challenges we have identified.

Acknowledgements

The authors would like to thank the Natural Sciences and Engineering Research Council of Canada, Dr. Robot Inc., and the Technology Evaluation in the Elderly Network (which is supported by the Government of Canada through the Networks of Centres of Excellence). We also thank the anonymous reviewers for their thoughtful feedback and suggestions during the paper review. Tiago Vaquero would like to thank the Keck Institute for Space Studies for supporting his work and his team at the Massachusetts Institute of Technology and the California Institute of Technology for the valuable discussions and insights.

Appendix A. PDDL Details

We provide PDDL code for some of the classes representing the static environment, the operators related to Bingo games, and the clock process. Depending on the modelling strategy used, different operators are combined. For example, *remind*, *play_Bingo* and *interact* are used for *single-clock*, whereas *min-add-clock* replaces *play_Bingo* with *play_Bingo3*.

```
(:durative-action remind
    :parameters (?self - Robot ?u - User ?g - BingoGame ?loc - Location
                 ?cs - ChargingStation)
    :duration (= ?duration (dur_remind ?g))
    :condition
      (and
        (over all (at ?self ?loc))
        (over all (at ?u ?loc))
        (over all (available ?u))
        (at start (ready ?self))
        (at start (at ?self ?loc))
        (at start (at ?u ?loc))
        (at start (available ?u))
        (at start (not_interacting ?u))
        (at start (not_done ?g))
        (at start (< (p_num ?g) (p_max ?g)))
        (at start (not_assigned_game ?u ?g))
        (at start (< (att_num ?u) (att_max ?u)))
        (at start (>= (bl ?self) (+ (+ (* (dur_remind ?g)
          (cr_remind ?self)) (* (distance_to_station ?loc ?cs)
          (cr_move ?self))) (bl_min ?self))))
      )
    :effect
      (and
        (at start (not (ready ?self)))
```

```
      (at start (not (not_interacting ?u)))
      (at end (ready ?self))
      (at end (not_interacting ?u))
      (at start (increase (p_num ?g) 1))
      (at start (participant ?g ?u))
      (at start (not (not_assigned_game ?u ?g)))
      (at start (increase (att_num ?u) 1))
      (at end (act_done ?self))
      (at start (decrease (bl ?self) (* (dur_remind ?g)
                 (cr_remind ?self))))
      (at start (assign (delivery_time ?g ?u) (current_time)))
      (at start (increase (total_battery_usage)
          (* (dur_remind ?g) (cr_remind ?self))))
    )
)

(:durative-action play_Bingo
 :parameters (?self - Robot ?g - BingoGame ?loc - GamesRoom
              ?cs - ChargingStation)
 :duration (= ?duration (dur ?g))
 :condition
   (and
     (at start (at ?self ?loc))
     (over all (at ?self ?loc))
     (at start (ready ?self))
     (at start (must_be_done_during ?g))
     (over all (must_be_done_during ?g))
     (at start (game_location ?g ?loc))
     (at start (not_done ?g))
     (at start (<= (p_num ?g) (p_max ?g)))
     (at start (> (p_num ?g) (- (p_min ?g) 1)))
     (at end (= (p_cur ?g) (p_num ?g)))
     (at start (>= (bl ?self) (+ (+ (* (dur ?g) (cr_Bingo ?self))
         (* (distance_to_station ?loc ?cs) (cr_move ?self)))
         (bl_min ?self))))
     (at start (free ?loc)))
 :effect
   (and
     (at start (not (ready ?self)))
     (at end (ready ?self))
     (at end (done ?g))
     (at start (not (not_done ?g)))
     (at start (playing ?self ?g))
     (at end (not (playing ?self ?g)))
     (at end (act_done ?self))
```

```
    (at start (decrease (bl ?self) (* (dur ?g) (cr_Bingo ?self))))
    (at start (increase (total_battery_usage) (* (dur ?g)
              (cr_Bingo ?self))))
    (at start (increase (games_attendees) (p_num ?g)))
    (at start (not (free ?loc)))
    (at end (free ?loc))))

(:durative-action interact
 :parameters (?self - Robot ?g - BingoGame ?u - User)
 :duration (= ?duration (- (dur ?g) 1))
 :condition
   (and
     (at start (playing ?self ?g))
     (over all (playing ?self ?g))
     (at start (available ?u))
     (over all (available ?u))
     (at start (not_interacting ?u))
     (at start (participant ?g ?u))
     (at start (>= (delivery_time_limit_max)
         (- (current_time) (delivery_time ?g ?u))))
     (at start (<= (delivery_time_limit_min)
         (- (current_time) (delivery_time ?g ?u)))))
 :effect
   (and
     (at start (increase (p_cur ?g) 1))
     (at start (not (not_interacting ?u)))
     (at end (not_interacting ?u))
     (at start (increase (total_delivery_time)
       (- (current_time) (delivery_time ?g ?u))))))

(:durative-action play_Bingo3
 :parameters (?self - Robot ?g - BingoGame ?loc - GamesRoom ?u1 - User
    ?u2 - User ?u3 - User ?cs - ChargingStation)
 :duration (= ?duration (dur ?g))
 :condition
   (and
     (at start (at ?self ?loc))
     (over all (at ?self ?loc))
     (at start (ready ?self))
     (at start (must_be_done_during ?g))
     (over all (must_be_done_during ?g))
     (at start (game_location ?g ?loc))
     (at start (not_done ?g))
     (at start (<= (p_num ?g) (p_max ?g)))
     (at start (> (p_num ?g) (- (p_min ?g) 1)))
     (at end (= (p_cur ?g) (p_num ?g)))
```

```
    (at start (>= (bl ?self) (+ (+ (* (dur ?g) (cr_Bingo ?self))
        (* (distance_to_station ?loc ?cs) (cr_move ?self)))
        (bl_min ?self))))
    (at start (free ?loc)))
    (at start (available ?u1))
    (over all (available ?u1))
    (at start (not_interacting ?u1))
    (at start (participant ?g ?u1))
    (at start (>= (delivery_time_limit_max) (- (current_time)
        (delivery_time ?g ?u1))))
    (at start (<= (delivery_time_limit_min) (- (current_time)
        (delivery_time ?g ?u1))))
    (at start (available ?u2))
    (over all (available ?u2))
    (at start (not_interacting ?u2))
    (at start (participant ?g ?u2))
    (at start (>= (delivery_time_limit_max) (- (current_time)
        (delivery_time ?g ?u2))))
    (at start (<= (delivery_time_limit_min) (- (current_time)
        (delivery_time ?g ?u2))))
    (at start (available ?u3))
    (over all (available ?u3))
    (at start (not_interacting ?u3))
    (at start (participant ?g ?u3))
    (at start (>= (delivery_time_limit_max) (- (current_time)
        (delivery_time ?g ?u3))))
    (at start (<= (delivery_time_limit_min) (- (current_time)
        (delivery_time ?g ?u3))))
    (at start (not (= ?u1 ?u2)))
    (at start (not (= ?u1 ?u3)))
    (at start (not (= ?u2 ?u3)))
    (at start (< (id ?u1) (id ?u2)))
    (at start (< (id ?u2) (id ?u3)))
    (at start (assigned ?u1 ?g))
    (at start (assigned ?u2 ?g))
    (at start (assigned ?u3 ?g))
:effect
  (and
    (at start (not (ready ?self)))
    (at end (ready ?self))
    (at end (done ?g))
    (at start (not (not_done ?g)))
    (at start (playing ?self ?g))
    (at end (not (playing ?self ?g)))
    (at end (act_done ?self))
```

```
        (at start (decrease (bl ?self) (* (dur ?g) (cr_Bingo ?self))))
        (at start (increase (total_battery_usage) (* (dur ?g)
                  (cr_Bingo ?self))))
        (at start (increase (games_attendees) 3))
        (at start (not (free ?loc)))
        (at end (free ?loc)))
        (at start (increase (p_cur ?g) 3))
        (at start (not (not_interacting ?u1)))
        (at end (not_interacting ?u1))
        (at start (increase (total_delivery_time)
            (+ (- (current_time) (delivery_time ?g ?u3))
            (+ (- (current_time) (delivery_time ?g ?u2))
            (- (current_time) (delivery_time ?g ?u1))))))
        (at start (not (not_interacting ?u2)))
        (at end (not_interacting ?u2))
        (at start (not (not_interacting ?u3)))
        (at end (not_interacting ?u3))
        (at start (increase (p_num ?g) 3)))

(:durative-action Bingo_overall
     :parameters (?g - BingoGame)
     :duration (= ?duration (+ (dur ?g) (delivery_time_limit_max))
     :condition
       (and
         (at start (not_done ?g))
         (at start (not_Bingo_actions_ready ?g)))
     :effect
       (and
         (at start (not (not_Bingo_actions_ready ?g)))
         (at start (Bingo_actions_ready ?g))
         (at end (not (Bingo_actions_ready ?g)))
         (at start (remind_enable ?g))))

(:durative-action setup_Bingo
     :parameters (?g - BingoGame)
     :duration (= ?duration (delivery_time_limit_min))
     :condition
       (and
         (at start (Bingo_actions_ready ?g))
         (over all (Bingo_actions_ready ?g))
         (at start (remind_enable ?g))
         (at start (not_done ?g)))
     :effect
       (and
         (at start (not (remind_enable ?g)))
         (at end (Bingo_game_ready ?g))))
```

```
    (:process clock_ticker
     :parameters ()
     :precondition
       (can_start_clock)
     :effect
       (increase (current_time) (* #t 1.0))
    )
```

Appendix B. NDDL Details

We provide NDDL code for the *Move* and *PlayBingo* actions. The code illustrates the requirements and effects of the actions and how these actions interact with the state of the system.

```
class Location {
    string name; }


class Path {
    string name;
    Location from;
    Location to;
    float distance; }

class ChargingStation {
    Location charging_station;
    ChargingStationUsage charging_station_usage; }

class ChargingStationUsage extends Reusable {
    string profileType;
    string detectorType;

    ChargingStationUsage() {
        super (1, 0);
        profileType = "GroundedProfile";
        detectorType = "GroundedFVDetector"; }}

class TelepresenceSession {
    string name;
    TelepresenceSessionState status;
    User localuser;
    Location location;
    int dur; }

class TelepresenceSessionState extends Timeline {
    predicate MustBeDone {}
    predicate InProgress {}
    predicate Done {} }
```

```
class BingoGame {
    string name;
    BingoGameState status;
    Location location;
    int dur;
    int players; }

class BingoGameState extends Timeline {
    predicate MustBeDone {}
    predicate InProgress {}
    predicate Done {} }

class User {
    string name;
    int deliveryTime;
    UserState state;
    UserAvailability availability;
    UserGameAssignment assignment; }

class UserState extends Timeline {
    predicate At {Location location; }
    predicate Interacting {Robot robot; }
    predicate Playing {Robot robot; BingoGame game;}
    predicate BeingReminded {Robot robot;} }

class UserAvailability extends Timeline {
  predicate Available {}
  predicate Busy {} }

class UserGameAssignment extends Timeline {
  predicate NotAssigned {}
  predicate BeingAssigned {}
  predicate Assigned {BingoGame game;} }

class Robot {
    string name;
    RobotState status;
    Battery battery;
    float speed;
    int cr_move;
    int recharge_rate;
    int cr_telep;
    int cr_bingo;
    int cr_reminder;

    // Actions
```

```
    action Move {
        Path path;
        Location destination; }

    action RechargeBattery {
        ChargingStation station; }

    action DoTelepresence {
        TelepresenceSession session;
        User user; }

    action PlayBingo {
        BingoGame game;
        User user;
        User user2;
        User user3;
        GameRoomUsage gameRoom;
        neq(user,user2);
        neq(user,user3);
        neq(user2,user3); }

    action Remind {
        BingoGame game;
        User user; }}

class RobotState extends Timeline {
    predicate FreeAt {Location location;}
    predicate Moving {Location destination;}
    predicate Charging {Location location; }
    predicate DoingTelepresence {TelepresenceSession session; User user;}
    predicate Reminding {BingoGame game; User user;}
    predicate PlayingGame {BingoGame game;} }

class Battery extends Reservoir {
    string profileType;
    string detectorType;

    Battery(int _ini, int _min, int _max) {
        super(_ini, _min, _max);
        profileType="GroundedProfile";
        detectorType = "GroundedFVDetector"; }}

Robot::Move {
    met_by(condition object.status.FreeAt _from);
    eq(_from.location,path.from);
    eq(destination, path.to);
```

```
    meets(effect object.status.FreeAt _to);
    eq(_to.location, destination);
    neq(_from.location, destination);
    eq(effect object.status.Moving _moving);
    eq(_moving.destination, destination);
    float dura, dist, vel, energy_use, c_move;
    dist == path.distance;
    vel == object.speed;
    c_move == object.cr_move;
    dist == dura * vel;
    energy_use == dura * c_move;
    starts(effect object.battery.consume cons);
    eq(cons.quantity, energy_use);
    eq(dura,duration); }

Robot::PlayBingo {
    met_by(condition object.status.FreeAt _robotFreeAtStart);
    met_by(condition game.status.MustBeDone _gameStart);
    eq(game.location,_robotFreeAtStart.location);
    contained_by(condition user1.availability.Available _user1Available);
    contained_by(condition user2.availability.Available _user2Available);
    contained_by(condition user3.availability.Available _user3Available);
    met_by(condition user1.assignment.Assigned _user1Assigned);
    met_by(condition user2.assignment.Assigned _user2Assigned);
    met_by(condition user3.assignment.Assigned _user3Assigned);
    eq(_user1Assigned.game, game);
    eq(_user2Assigned.game, game);
    eq(_user3Assigned.game, game);
    eq(effect object.status.PlayingGame _interactingRobot);
    eq(_interactingRobot.game, game);
    meets(effect object.status.FreeAt _robotFreeAtEnd);
    eq(_robotFreeAtStart.location,_robotFreeAtEnd.location);
    eq(effect user1.state.Interacting _interactingUser1);
    eq(_interactingUser1.robot, object);
    eq(effect user2.state.Interacting _interactingUser2);
    eq(_interactingUser2.robot, object);
    eq(effect user3.state.Interacting _interactingUser3);
    eq(_interactingUser3.robot, object);
    meets(effect user1.state.At _user1AtEnd);
    eq(_robotFreeAtStart.location,_user1AtEnd.location);
    meets(effect user2.state.At _user2AtEnd);
    eq(_robotFreeAtStart.location,_user2AtEnd.location);
    meets(effect user3.state.At _user3AtEnd);
    eq(_robotFreeAtStart.location,_user3AtEnd.location);
    equals(effect game.status.InProgress _gameProgress);
```

```
meets(effect game.status.Done _gameDone);
eq(_gameDone.end, Horizon);
eq(GameRoomUsage.uses use_room);
int user1Delivery, user2Delivery, user2Delivery;
user1Delivery == _interactingRobot.start - user1.deliveryTime;
user2Delivery == _interactingRobot.start - user2.deliveryTime;
user3Delivery == _interactingRobot.start - user3.deliveryTime;
user1Delivery >= 15;
user1Delivery <= 120;
user2Delivery >= 15;
user2Delivery <= 120;
user3Delivery >= 15;
user3Delivery <= 120;
eq(game.dur, use_room.duration);
eq (use_room.quantity, 1);
int _cr_bingo, dura, energy_use_bingo;
_cr_bingo == object.cr_bingo;
dura == game.dur;
energy_use_bingo == dura * _cr_bingo;
starts(effect object.battery.consume cons);
eq(cons.quantity, energy_use_bingo);
eq(game.dur, duration); }
```

Appendix C. Detailed Results

Detailed results for all the PDDL planning models tested on each of the problem variants are presented here. The first and last feasible solutions found within a one-hour time limit is recorded.

Table 9: Performance of PDDL planning on all tested problem modifications for the *single-clock* model. A (-) indicates that no solution was found.

Problem	Scenario	Runtime (s)		Participants		Objective Value	
		first	last	first	last	first	last
BRPOF	1	0.04	786.12	0	3	5,506.13	2,070.75
	2	0.18	0.88	0	0	11,019.38	11,016.66
	3	0.84	9.38	0	0	16,530.31	16,525.73
	4	2.18	23.84	0	0	22,044.46	22,043.11
	5	7.24	89.46	0	0	27,557.30	27,554.54
-RPOF	1	0.02	345.02	0	3	5,500.00	2,051.23
	2	0.12	306.00	0	4	11,000.00	7,208.00
	3	0.34	0.34	0	0	16,500.00	16,500.00
	4	0.92	0.92	0	0	22,000.00	22,000.00
	5	1.94	1.94	0	0	27,500.00	27,500.00
B-POF	1	0.04	168.42	0	3	5,506.13	2,070.41
	2	0.18	0.78	0	0	11,019.38	11,016.66
	3	0.74	7.24	0	0	16,530.31	16,525.73
	4	1.90	16.62	0	0	22,044.46	22,043.11
	5	6.10	66.33	0	0	27,557.30	27,554.54
BR-OF	1	0.04	761.99	0	0	5,508.49	5,506.78
	2	0.18	0.88	0	0	11,019.38	11,016.66
	3	0.84	9.02	0	0	16,530.31	16,525.73
	4	2.16	22.88	0	0	22,044.46	22,043.11
	5	7.16	92.06	0	0	27,557.30	27,554.54
BRP-F	1	-	-	-	-	-	-
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRPO-	1	0.06	75.94	0	3	5,612.46	2,761.64
	2	0.18	0.18	0	0	11,019.38	11,019.38
	3	0.82	0.82	0	0	16,530.31	16,530.31
	4	2.18	2.18	0	0	22,044.46	22,044.46
	5	7.04	7.04	0	0	27,557.30	27,557.30
B—F	1	0.10	163.72	4	4	1,013.33	1,013.22
	2	4.38	391.42	8	8	2,526.20	2,526.09
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-

Table 10: Performance of PDDL planning on all tested problem modifications for the *min-add-clock* model. A (-) indicates that no solution was found.

Problem	Scenario	Runtime (s)		Participants		Objective Value	
		first	last	first	last	first	last
BRPOF	1	0.06	2,950.88	0	3	5,506.13	2,066.51
	2	0.68	2,062.94	0	0	11,012.37	11,012.23
	3	57.16	1,960.02	0	0	16,518.65	16,518.63
	4	143.36	143.36	0	0	22,024.92	22,024.92
	5	-	-	-	-	-	-
-RPOF	1	0.04	1173.57	0	3	5,500.00	2,047.51
	2	0.42	0.42	0	0	11,000.00	11,000.00
	3	19.68	19.68	0	0	16,500.00	16,500.00
	4	53.18	53.18	0	0	22,000.00	22,000.00
	5	1.94	.94	0	0	27,500.00	27,500.00
B-POF	1	0.06	1,453.80	0	3	5,506.13	2,142.00
	2	0.94	1,995.34	0	0	11,012.37	11,012.23
	3	55.64	1,717.98	0	0	16,518.65	16,518.62
	4	144.08	144.08	0	0	22,024.92	22,024.92
	5	6.10	66.33	0	0	27,557.30	27,554.54
BR-OF	1	0.04	761.99	0	0	5,508.49	5,506.13
	2	0.18	0.88	0	0	11,019.38	11,016.66
	3	0.84	9.02	0	0	16,530.31	16,525.73
	4	2.16	22.88	0	0	22,044.46	22,043.11
	5	7.16	92.06	0	0	27,557.30	27,554.54
BRP-F	1	-	-	-	-	-	-
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRPO-	1	0.06	2,459.74	0	3	5,506.13	2,161.64
	2	0.82	0.82	0	0	11,019.38	11,019.38
	3	0.82	0.82	0	0	16,518.65	16,518.65
	4	146.72	146.72	0	0	22,044.46	22,044.46
	5	7.04	7.04	0	0	27,557.30	27,557.30
B—F	1	-	-	-	-	-	-
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-

Table 11: Performance of PDDL planning on all tested problem modifications for the *set-all-clock* model. A (-) indicates that no solution was found.

Problem	Scenario	Runtime (s)		Participants		Objective Value	
		first	last	first	last	first	last
BRPOF	1	0.74	2,033.72	3	4	2,147.10	1,124.11
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
-RPOF	1	6.22	829.26	3	5	1,201.69	294.01
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
B-POF	1	0.56	753.48	3	4	2,147.10	1,178.79
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BR-OF	1	-	-	-	-	-	-
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRP-F	1	0.58	1633.62	3	4	2,147.10	1,124.11
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRPO-	1	0.46	602.40	3	4	2,226.30	1,617.03
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
B—F	1	7.10	7.10	4	4	1,451.96	1,451.96
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-

Table 12: Performance of PDDL planning on all tested problem modifications for the *single-envelope* model. A (-) indicates that no solution was found.

Problem	Scenario	Runtime (s)		Participants		Objective Value	
		first	last	first	last	first	last
BRPOF	1	0.84	1,287.07	3	3	2,152.33	2,077.94
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
-RPOF	1	0.20	2,874.84	3	5	2,138.20	347.51
	2	500.76	500.76	9	9	1,963.17	1,963.17
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
B-POF	1	0.60	2,581.60	3	4	2,194.73	1,230.67
	2	500.76	500.76	9	9	2,326.43	2,326.43
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BR-OF	1	0.20	0.20	4	4	1,213.43	1,213.43
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRP-F	1	0.84	1,513.16	3	4	2,138.93	1,195.61
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRPO-	1	0.26	1,297.10	3	5	2,138.96	530.36
	2	1,190.36	1,190.36	9	9	2,680.88	2,680.88
	3	3,406.20	3,406.20	15	15	1,089.24	1,089.24
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
B—F	1	0.56	0.56	4	4	1,313.44	1,313.44
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-

Table 13: Performance of PDDL planning on all tested problem modifications for the *min-add-envelope* model. A (-) indicates that no solution was found.

Problem	Scenario	Runtime (s)		Participants		Objective Value	
		first	last	first	last	first	last
BRPOF	1	0.84	1,287.07	3	3	2,152.333	2,077.94
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
-RPOF	1	0.98	65.62	3	3	2,161.24	2,126.01
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
B-POF	1	1.12	249.46	3	3	2,147.10	2,071.93
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BR-OF	1	-	-	-	-	-	-
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRP-F	1	1.04	1,013.14	3	3	2,152.33	2,077.94
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRPO-	1	0.22	1,269.80	3	4	2,261.10	1,620.54
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
B—F	1	-	-	-	-	-	-
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-

Table 14: Performance of PDDL planning on all tested problem modifications for the *set-all-envelope* model. A (-) indicates that no solution was found.

Problem	Scenario	Runtime (s)		Participants		Objective Value	
		first	last	first	last	first	last
BRPOF	1	0.76	943.56	3	3	2,152.33	2,077.94
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
-RPOF	1	4.28	3,021.76	3	4	2,109.80	1,102.98
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
B-POF	1	0.78	3336.24	3	4	2,147.10	1,168.81
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BR-OF	1	-	-	-	-	-	-
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRP-F	1	1.24	769.30	3	3	2,152.33	2,077.94
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
BRPO-	1	0.42	721.80	3	4	2,261.14	1,620.54
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-
B—F	1	-	-	-	-	-	-
	2	-	-	-	-	-	-
	3	-	-	-	-	-	-
	4	-	-	-	-	-	-
	5	-	-	-	-	-	-

References

Alami, R., Chatila, R., Fleury, S., Ghallab, M., & Ingrand, F. (1998). An architecture for autonomy. *The International Journal of Robotics Research*, *17*(4), 315–337.

Bacchus, F., & Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, *116*(1), 123 – 191.

Bajada, J., Fox, M., & Long, D. (2014). Temporal plan quality improvement and repair using local search. In *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS)*, Vol. 264, pp. 41–50. IOS Press.

Banks, M. R., Willoughby, L. M., & Banks, W. A. (2008). Animal-assisted therapy and loneliness in nursing homes: use of robotic versus living dogs. *Journal of the American Medical Directors Association*, *9*(3), 173–177.

Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T., & Smith, D. (2012). EUROPA: A platform for AI planning, scheduling, constraint programming, and optimization. In *Proceedings of the 4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)*.

Barták, R. (2002). Visopt shopfloor: On the edge of planning and scheduling. In *Proceedings of the Principles and Practice of Constraint Programming (CP)*, pp. 587–602. Springer.

Beck, J. C., & Fox, M. S. (2000). Constraint-directed techniques for scheduling alternative activities. *Artificial Intelligence*, *121*(1), 211–250.

Beetz, M., & Bennewitz, M. (1998). Planning, scheduling, and plan execution for autonomous robot office couriers. In *Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments, volume Workshop Notes*, pp. 98–02.

Benton, J., Coles, A. J., & Coles, A. I. (2012). Temporal planning with preferences and time-dependent continuous costs.. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 2–10.

Bertsimas, D., & Weismantel, R. (2005). *Optimization over integers*, Vol. 13. Dynamic Ideas Belmont.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1), 5–33.

Booth, K. E., Tran, T. T., Nejat, G., & Beck, J. C. (2016). Mixed-integer and constraint programming techniques for mobile robot task planning. *IEEE Robotics and Automation Letters*, *1*(1), 500–507.

Burt, C., Lipovetzky, N., Pearce, A., & Stuckey, P. (2015). Approximate uni-directional benders decomposition. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence Workshop on Planning Search and Optimization (PlanSOpt)*, pp. 1–8.

Cesta, A., Cortellessa, G., Rasconi, R., Pecora, F., Scopelliti, M., & Tiberio, L. (2011). Monitoring elderly people with the robocare domestic environment: Interaction synthesis and user evaluation. *Computational Intelligence*, *27*(1), 60–82.

Cesta, A., Fratini, S., & Pecora, F. (2008). Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Science*, *18*(2), 231–271.

Coles, A. J., Coles, A. I., Fox, M., & Long, D. (2012). Colin: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research*, *44*(1), 1–96.

Coles, A. J., Coles, A. I., Fox, M., & Long, D. (2013). A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of Artificial Intelligence Research*, *46*, 343–412.

Coles, A. J., Coles, A., Fox, M., & Long, D. (2010). Forward-chaining partial-order planning.. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 42–49.

Coltin, B., Veloso, M. M., & Ventura, R. (2011). Dynamic user task scheduling for mobile robots.. In *Automated Action Planning for Autonomous Mobile Robots*, pp. 1–9.

Cushing, W. A., & Kambhampati, S. (2007). When is temporal planning really temporal. In *Proceedigns of the 20th International Joint Conference On Artificial Intelligence (IJCAI)*, pp. 1852–1859.

Dang, Q.-V., Nielsen, I., Steger-Jensen, K., & Madsen, O. (2013). Scheduling a single mobile robot for part-feeding tasks of production lines. *Journal of Intelligent Manufacturing*, *25*(6), 1–17.

de la Rosa, T., Jiménez, S., Fuentetaja, R., & Borrajo, D. (2011). Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence Research*, *40*, 767–813.

Dvorak, F., Bit-Monnot, A., Ingrand, F., & Ghallab, M. (2014). A flexible anml actor and planner in robotics. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Planning and Robotics (PlanRob)*.

Edelkamp, S., & Hoffmann, J. (2004). PDDL2.2: the language for the classical part of the 4th international planning competition. *4th International Planning Competition (IPC04), at ICAPS04*.

Estlin, T., Gaines, D., Chouinard, C., Castano, R., Bornstein, B., Judd, M., Nesnas, I., & Anderson, R. (2007). Increased mars rover autonomy using AI planning, scheduling and execution. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4911–4918. IEEE.

Fdez-Olivares, J., Cózar, J. A., & Castillo, L. (2009). Oncotheraper: Clinical decision support for oncology therapy planning based on temporal hierarchical tasks networks. In *Knowledge Management for Health Care Procedures*, Vol. 5626, pp. 25–41. Springer.

Feldman, Z., & Domshlak, C. (2014). Simple regret optimization in online planning for markov decision processes. *Journal of Artificial Intelligence Research*, *51*, 165–205.

Focacci, F., Lodi, A., & Milano, M. (2002). Optimization-oriented global constraints. *Constraints*, *7*(3-4), 351–365.

Fox, M., & Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, *27*, 235–297.

Frank, J. (2008). An intelligent agent for autonomous lunar exploration. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Scheduling and Planning Application*, pp. 1–8.

Freuder, E. C. (1997). In pursuit of the holy grail. *Constraints*, *2*(1), 57–61.

Frisch, A. M., Harvey, W., Jefferson, C., Martínez-Hernández, B., & Miguel, I. (2008). Essence: A constraint language for specifying combinatorial problems. *Constraints*, *13*(3), 268–306.

Fritz, C., & McIlraith, S. A. (2008). Planning in the face of frequent exogenous events. In *Online Poster Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 14–18.

Gaines, D. M., Estlin, T., Chouinard, C., Castano, R., Castano, A., Bornstein, B., Anderson, R. C., Judd, M., Nesnas, I., & Rabideau, G. (2006). Opportunistic planning and execution for planetary exploration. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 1–3.

Gerevini, A., Saetti, A., Serina, I., & Toninelli, P. (2004). LPG-TD: a fully automated planner for PDDL2.2 domains. In *In Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS) International Planning Competition abstracts*. Booklet of the system demo section.

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL—The Planning Domain Definition Language..

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

Heinz, S., Ku, W.-Y., & Beck, J. C. (2013a). Recent improvements using constraint integer programming for resource allocation and scheduling. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 12–27. Springer.

Heinz, S., Schulz, J., & Beck, J. C. (2013b). Using dual presolving reductions to reformulate cumulative constraints. *Constraints*, *18*(2), 166–201.

Hendler, J. A., Tate, A., & Drummond, M. (1990). Ai planning: Systems and techniques. *AI magazine*, *11*(2), 61.

Hewitt, C. (1971). Procedural embedding of knowledge in planner. In *Proceedings of the 2nd International Joint Conference on Artificial intelligence (IJCAI)*, pp. 167–182.

Hoffmann, J., & Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Hooker, J. N., & Ottosson, G. (2003). Logic-based benders decomposition. *Mathematical Programming*, *96*(1), 33–60.

Hsu, C.-W., & Wah, B. W. (2008). The SGPlan planning system in IPC-6. In *In the booklet of the International Planning Competition (IPC), International Conference on Planning and Scheduling (ICAPS)*.

Ivankovic, F., Haslum, P., Thiébaux, S., Shivashankar, V., & Nau, D. S. (2014). Optimal planning with global numerical state constraints. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 145–153.

Jain, A., Guineau, J., Lim, C., Lincoln, W., Pomerantz, M., Sohl, G., & Steele, R. (2003). Roams: Planetary surface rover simulation environment. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, pp. 1–8.

Jiménez, S., Jonsson, A., & Palacios, H. (2015). Temporal planning with required concurrency using classical planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*. In press.

Kats, V., & Levner, E. (2011). Parametric algorithms for 2-cyclic robot scheduling with interval processing times. *Journal of Scheduling*, *14*(3), 267–279.

Krajanský, M., Hoffmann, J., Buffet, O., & Fern, A. (2014). Learning Pruning Rules for Heuristic Search Planning. In *21st European Conference on Artificial Intelligence*, Proceedings of the 21st European Conference on Artificial Intelligence (ECAI), pp. 483–488, Prague, Czech Republic.

Kvarnström, J., & Doherty, P. (2000). Talplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, *30*(1-4), 119–169.

Kwak, J.-y., Varakantham, P., Maheswaran, R., Tambe, M., Jazizadeh, F., Kavulya, G., Klein, L., Becerik-Gerber, B., Hayes, T., & Wood, W. (2012). Saves: A sustainable multiagent application to conserve building energy considering occupants. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Vol. 1, pp. 21–28.

Laborie, P., & Rogerie, J. (2016). Temporal linear relaxation in IBM ILOG CP Optimizer. *Journal of Scheduling*, *19*(4), 391–400.

Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. *Artificial Intelligence*, *143*(2), 151–188.

Laborie, P. (2009). IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 148–162. Springer.

Laborie, P., & Rogerie, J. (2008). Reasoning with conditional time-intervals.. In *Proceedings of the Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pp. 555–560.

Li, J., Louie, W.-Y. G., Despond, F., & Nejat, G. (2016). A user-study with tangy the bingo facilitating robot and long-term care residents. In *Proceedings of the IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE.

Louie, W.-Y. G., Han, R., & Nejat, G. (2013). Did anyone say BINGO: A socially assistive robot to promote stimulating recreational activities at long-term care facilities. *Journal of Medical Devices*, *7*(3), 030944.

Louie, W.-Y. G., Li, J., Vaquero, T., & Nejat, G. (2014). A focus group study on the design considerations and impressions of a socially assistive robot for long-term care. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pp. 237–242. IEEE.

Louie, W.-Y. G., Li, J., Vaquero, T., & Nejat, G. (2015). Socially assistive robots for seniors living in residential care homes: User requirements and impressions. In *Human-Robot Interactions: Principles, Technologies and Challenges*, pp. 75–108. Nova Science Publishers, Inc.

Louie, W.-Y. G., Vaquero, T., Nejat, G., & Beck, J. C. (2014). An autonomous assistive robot for planning, scheduling and facilitating multi-user activities. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5292–5298. IEEE.

Marzal, E., Sebastia, L., & Onaindia, E. (2014). On the use of temporal landmarks for planning with deadlines. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 172–180.

Mohamed, S. C., & Nejat, G. (2016). Autonomous search by a socially assistive robot in a residential care environment for multiple elderly users using group activity preferences. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Planning and Robotics (PlanRob)*, pp. 58–66.

Muise, C., Beck, J. C., & McIlraith, S. A. (2013). Flexible Execution of Partial Order Plans With Temporal Constraints. In *Proceedings of the International Joint Conference On Artificial Intelligence (IJCAI)*, pp. 2328–2335.

Muise, C., Belle, V., & McIlraith, S. A. (2014). Computing contingent plans via fully observable non-deterministic planning. *Models and Paradigms for Planning under Uncertainty: a Broad Perspective*, 2322–2329.

Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, *20*, 379–404.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). Minizinc: Towards a standard cp modelling language. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 529–543. Springer.

Nightingale, P., & Rendl, A. (2016). Essence' description 1.6. 4. *arXiv preprint arXiv:1601.02865*.

Nilsson, N. J. (1984). Shakey the robot. Tech. rep., DTIC Document.

OMG (2005). OMG unified modeling language specification. Version 2.0.

Pecora, F., & Cesta, A. (2002). Planning and scheduling ingredients for a multi-agent system. In *Proceedings of UK PLANSIG02 Workshop*, Vol. 371, pp. 135–148.

Pineau, J., Montemerlo, M., Pollack, M., Roy, N., & Thrun, S. (2003). Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, *42*(3), 271–281.

Pinedo, M. L. (2012). *Scheduling: theory, algorithms, and systems.* Springer.

Pollack, M. E. (2005). Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment. *AI magazine*, *26*(2), 9–24.

Reddy, S. Y., Iatauro, M. J., Kürklü, E., Boyce, M. E., Frank, J. D., & Jónsson, A. K. (2008). Planning and monitoring solar array operations on the ISS. In *Proceedings of the Scheduling and Planning Applications Workshop (SPARK), ICAPS*, pp. 1–8.

Régin, J.-C. (1999). Arc consistency for global cardinality constraints with costs. In *Proceedings of the Principles and Practice of Constraint Programming (CP)*, pp. 390–404. Springer.

Riddle, P. J., Holte, R. C., & Barley, M. W. (2011). Does representation matter in the planning competition?. In *Proceedings of the 9th Symposium on Abstraction, Reformulation, and Approximation, SARA*.

Ross, S., Pineau, J., Paquet, S., & Chaib-Draa, B. (2008). Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, *32*, 663–704.

Schrijver, A. (1998). *Theory of linear and integer programming.* John Wiley & Sons.

Simonis, H., & Hadzic, T. (2011). A resource cost aware cumulative. In *Recent Advances in Constraints*, pp. 76–89. Springer.

Smith, B. (2006). Modelling. In Rossi, F., van Beek, P., & Walsh, T. (Eds.), *Handbook of Constraint Programming*, chap. 11, pp. 377–406. Elsevier.

Smith, D. E. (2003). The case for durative actions: A commentary on pddl2.1. *Journal of Artificial Intelligence Research*, *20*, 149–154.

Smith, D. E., Frank, J., & Cushing, W. (2008). The ANML language. *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, 1–8.

Smith, D. E., Frank, J., & Jónsson, A. K. (2000). Bridging the gap between planning and scheduling. *The Knowledge Engineering Review*, *15*(1), 47–83.

Tate, A., Drabble, B., & Kirby, R. (1994). O-plan2: an open architecture for command, planning and control. In *Intelligent Scheduling*, pp. 213–239. Morgan Kaufmann.

Thorsteinsson, E. S. (2001). Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *Proceedings of the Principles and Practice of Constraint Programming (CP)*, pp. 16–30. Springer.

Turkle, S. (2006). A nascent robotics culture: new complicities for companionship. In *American Association for Artificial Intelligence Technical Report Series AAAI.*

United Nations (2002). *World population ageing, 1950-2050.* No. 207 in ST/ESA/SER.A. New York: United Nations. Department of Economic and Social Affairs.

Vaquero, T., Mohamed, S. C., Nejat, G., & Beck, J. C. (2015). The implementation of a planning and scheduling architecture for multiple robots assisting multiple users in a retirement home setting. In *Proceedings of the AAAI'15 Workshop on Artificial Intelligence Applied to Assistive Technologies and Smart Environments*, pp. 1–6.

Vaquero, T. S., Silva, J. R., Tonidandel, F., & Beck, J. C. (2013). itSIMPLE: towards an integrated design system for real planning applications. *The Knowledge Engineering Review*, *28*(02), 215–230.

Vaquero, T. S., Silva, J. R., & Beck, J. C. (2010). Improving planning performance through post-design analysis. In *Proceedings of ICAPS 2010 workshop on Scheduling and Knowledge Engineering for Planning and Scheduling (KEPS)*, pp. 45–52.

Vaquero, T. S., Silva, J. R., Ferreira, M., Tonidandel, F., & Beck, J. C. (2009). From requirements and analysis to PDDL in itSIMPLE3.0. In *Proceedings of the 3rd International Competition on Knowledge Engineering for Planning and Scheduling (ICAPS)*, pp. 54–61.

Vaquero, T. S., Tonidandel, F., de Barros, L. N., & Silva, J. R. (2006). On the use of UML.P for modeling a real application as a planning problem.. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 434–437.

Vidal, V., & Geffner, H. (2006). Branching and pruning: An optimal temporal pocl planner based on constraint programming. *Artificial Intelligence*, *170*(3), 298–335.

Wilkins, D. E. (1984). Domain-independent planning representation and plan generation. *Artificial Intelligence*, *22*(3), 269–301.

Zhang, Y., & Parker, L. E. (2012). Task allocation with executable coalitions in multi-robot tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3307–3314. IEEE.

Zhang, Y., & Parker, L. E. (2013). Multi-robot task scheduling. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2992–2998. IEEE.