DOI: 10.1109/RO-MAN57019.2023.10309457

A Framework for Realistic Simulation of Daily Human Activity

Ifrah Idrees[1,*], Siddharth Singh[2], Kerui Xu[2], Dylan F. Glas[2]

***Abstract*— For social robots like Astro which interact with and adapt to the daily movements of users within the home, realistic simulation of human activity is needed for feature development and testing. This paper presents a framework for simulating daily human activity patterns in home environments at scale, supporting manual configurability of different personas or activity patterns, variation of activity timings, and testing on multiple home layouts. We introduce a method for specifying day-to-day variation in schedules and present a bidirectional constraint propagation algorithm for generating schedules from templates. We validate the expressive power of our framework through a use case scenario analysis and demonstrate that our method can be used to generate data closely resembling human behavior from three public datasets and a self-collected dataset. Our contribution supports systematic testing of social robot behaviors at scale, enables procedural generation of synthetic datasets of human movement in different households, and can help minimize bias in training data, leading to more robust and effective robots for home environments.**

I. Introduction

Development of a commercial robot to coexist with people over the long term in a home environment is a challenging task. For robot behaviors which depend on spatial interaction with users, prediction of their locations, or long-term adaptation to user behavior, it can be difficult to evaluate performance effectively through on-device testing alone.

To support the development of behaviors like these for Amazon's Astro robot, we developed a simulation framework for scalable generation of daily human activity, enabling robot testing in a simulation environment with humans moving realistically throughout the home (Fig. 1). This approach is not specific to Astro, and could be used for any social robot or smart-home system designed for long-term deployment.

User simulation Simulation testing is needed when robot capabilities depend on user behavior patterns. For instance, if the robot is asked to deliver a message to a user and needs to search the home for them, simulation testing is needed to quantify the efficiency of the search and its success rate.

To be effective, these tests should be performed in a variety of simulated homes, at various times of day, and with realistic user behavior patterns. Regardless of whether the robot is meant to learn and adapt to user behaviors or follow fixed rules, realistic testing is necessary to robustly evaluate its effectiveness before it can be released to real users.

Avoiding bias in training data There has recently been a great deal of social awareness around the topic of bias in AI, both in popular media and in social robotics research [9, 14, 4]. While there are complex and subtle aspects to this topic, one common type of AI bias occurs when the training data are not representative of the eventual users of a system.

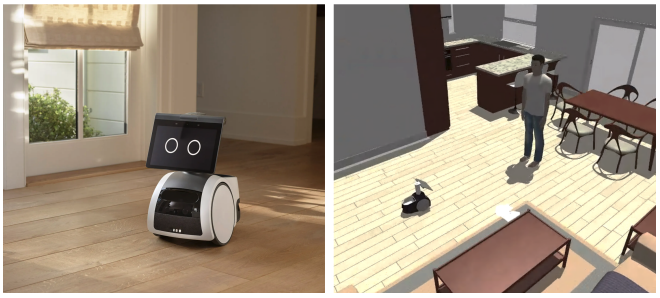

Fig. 1. Left: Amazon's Astro robot. Right: Gazebo-based simulation environment used in feature development and testing for Astro.

In the context of the current work, if the testing scenarios were all based on large homes with spacious floorplans, the robot might not perform well in smaller homes. Similarly, if the user behavior patterns in testing scenarios only represented working couples with children, the robot might not perform well in the home of a single retiree living alone.

To avoid this kind of bias, it is important to ensure that testing incorporates a diversity of floorplans and simulated user behaviors representing a broad range of possible users. Note that this work focuses on spatial and timing aspects of human activity but not computer vision, so we do not directly address common bias-related topics related to race, gender or skin tone.

Manual tuning For a commercial product, systematic testing is essential for prevention of regressions as features are developed over time. Quality Assurance (QA) teams develop thorough testing plans to cover both typical usage patterns and any anticipated edge case conditions which might cause failures. For this reason, the ability to hand-craft and tune test cases is important, providing QA with the ability to precisely specify user behavior for each test case.

Requirements The objective of this simulation framework is thus to satisfy the following requirements:

1) Manual control over simulated behaviors
2) Configurability for different personas or lifestyles
3) Day-to-day variation of activity timings
4) Execution on a variety of different floorplans

This paper presents a novel framework for generating configurable and variable schedules for daily human activity satisfying the above requirements, enabling testing of robot behaviors at scale in a simulation environment.

[1]Ifrah Idrees is affiliated with Dept. of Computer Science, Brown University, Providence, RI, USA ifrah_idrees@brown.edu

[2]Siddharth Singh, Kerui Xu, and Dylan F. Glas are affiliated with Amazon Lab126, 1100 Enterprise Way, Sunnyvale, CA, USA {hartsid, xkerui, dgglas}@amazon.com

*Work done during an Amazon Lab126 internship.

II. Related Work

A. Simulating human behavior

Simulation of human behavior is not new in HRI, and it is often an important element of social navigation research. Variants of the social force model [8] are often used for modeling crowds and multi-person scenarios. Systems such as SocNavBench [1] and SEAN [19] use playback of prerecorded trajectory patterns based on real data as a basis for evaluating robot navigational behavior. Kidokoro et al. used models of observed pedestrian subgoals and stochastically generated pedestrians to follow those routes [12], and Kaneshige et al. found that using simulation of pedestrian behavior in testing greatly improved the efficiency of robot behavior development in a shopping mall [11].

These approaches all focus on short-term or local behavior, often with anonymous pedestrians who appear and disappear. Our focus in this work is to model longer-term behavior of a persistent user (or users) in the home throughout the day.

B. Human schedule generation from data

Some work has focused on building generative models based on captured human behavior data. Francillette et al. developed a method for generating behavior trees from historical activity data [5]. However, this work focused on granular actions such as manipulation of cups and dishes, whereas our interest is in longer-term movement around the home. Elbayoudi et al. [3] proposed a system for simulating activities of daily living for a targeted population of older adults based on captured data using Hidden Markov Model and Direct Simulation Monte Carlo methods, which, like our work, was applied at the level of room-to-room transitions.

Patel et al. built generative models of daily life behavior from crowdsourced activity schedules, using Gaussian mixture models to capture average times and variability of daily activities [15] but did not model sequence dependencies or hard time constraints. Garcia-Ceja et al. [6] modeled daily life activities using conditional random fields. However, their focus was on estimation of human activity given accelerometer data, rather than generation of synthetic data.

None of the models described above were intended to be manually adjusted, whereas a primary aim of our work is to provide designers and testers with manual control over behaviors, although the ability to simulate behavior patterns which imitate observed data is also of interest to us.

C. Existing Human Simulators

Virtual Home [16] and Alfred [17] work well in generating probabilistic human motion patterns referred to as activity programs over multi-step, short-term tasks, but do not focus on modeling activity variation from day to day. To our knowledge, there does not exist work that focuses on generating varying daily activity schedules for different user profiles.

III. Our Approach

The elements of our proposed framework are shown in Fig. 2, A designer first creates a **schedule template** specifying a time sequence of abstracted activities for a simulated user's

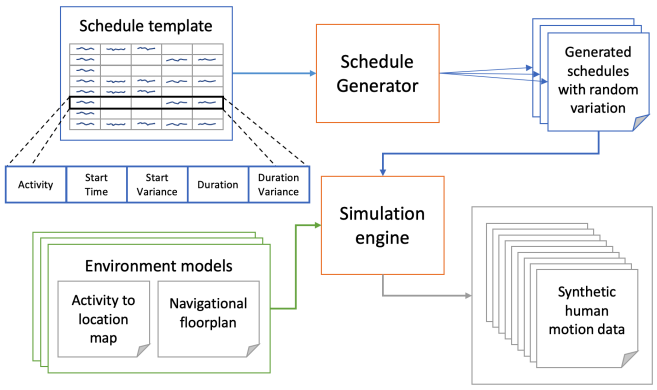

Fig. 2. Elements of the proposed simulation framework

day, including optional time constraints and ranges of variability for start times and activity durations. The **schedule generator** processes this template to generate a set of varying daily activity schedules. To support simulation in different homes, several **environment models** are defined, each containing a **navigational floorplan**, used for path planning in the simulator, and a corresponding **activity-to-location map** defining specific (x, y) coordinates for each activity. The generated schedules are executed by the **simulation engine** to move simulated users around the home according to their scheduled activities throughout the day, generating a diverse set of synthetic data scenarios containing day-to-day schedule variations and a variety of home environments.

Activities are mapped to specific locations in the home. For example, $cooking \rightarrow kitchen$. For each floorplan, we require an (x, y) location to be defined for each activity, allowing the floorplans to be interchanged, so the same schedule template can be used in different home layouts.

As the path-planning and obstacle-avoidance aspects of the human motion simulator are not novel, we will focus on the schedule generation aspect in this work. In the following subsections, we discuss the technical contribution in detail.

A. Problem Formulation

This section will describe the schedule generation algorithm. Let ST denote a schedule template consisting of an ordered sequence of n entries, where $ST = \{st_1, st_2, ..., st_n\}$, and each entry $st_i = (a_i, t^i_{start}, v^i_{start}, d^i, v^i_d)$, containing an activity a_i, a start time t^i_{start} with variability v^i_{start}, and duration d^i with variability v^i_d. For each entry, a_i is required, but $(t^i_{start}, v^i_{start})$ may be defined or left empty, and (d^i, v^i_d) may be defined or left empty.

Next, let S denote a generated schedule consisting of an ordered sequence of n entries, where $S = \{s_1, s_2, ..., s_n\}$, and each entry $s_i = (a_i, t^i_{start}, d^i, t^i_{end})$, containing an activity a_i, start time t^i_{start}, duration d^i, and end time t^i_{end}.

B. Iterative Bi-directional Constraint Propagation

Algorithm 1 describes our method for generating a concrete schedule instance S based on a schedule template ST.

1. Initializing Random Values The first step is to initialize S from the constraints provided by the designer in the

template. In this step, random variation is added to (t_{start}, d) from uniform distributions bounded by the specified ranges ($\pm v_{start}$, $\pm v_d$). For each entry s_i, t_{end} is initialized to $null$.

The next step is to iteratively populate any uninitialized start and end times in S by alternating between two update steps: applying duration constraints, and applying adjacency constraints. Rules are also applied to eliminate gaps in the schedule and avoid time conflicts.

Algorithm 1 Schedule Generation Algorithm

Input Schedule template ST
Output Schedule S

procedure GENERATESCHEDULE(ST)
  ▷ Step 1: Apply random initialization to activities
  for each constraint $st_i \in ST$ **do**
    $s_i[a] \leftarrow st_i[a]$
    if $\exists(st_i[t_{start}], st_i[v_{start}])$ **then**
      $\Delta t \leftarrow \mathcal{U}(-st_i[v_{start}], st_i[v_{start}])$
      $s_i[t_{start}] \leftarrow st_i[t_{start}] + \Delta t$
    end if
    if $\exists(st_i[d], st_i[v_d])$ **then**
      $\Delta t \leftarrow \mathcal{U}(-st_i[v_d], st_i[v_d])$
      $s_i[d] \leftarrow st_i[d] + \Delta t$
    end if
  end for
  ▷ Step 2: Iteratively propagate constraints in S
  while $\exists s_i \ni \nexists s_i[t_{start}] \vee \nexists s_i[t_{end}]$ **do**
    for each entry $s_i \in S$ **do**
      APPLYDURATIONCONSTRAINTS(s_i)
    end for
    for each entry $s_i \in S$ **do**
      APPLYADJACENCYCONSTRAINTS(s_i)
    end for
    if no entries were updated in either step **then**
      FAIL($underconstrained$)
    end if
  end while
end procedure

2. Solving Duration Constraints (Algorithm 2) Because $s_i[t_{start}] + s_i[d] = s_i[t_{end}]$, we can determine the end time of any s_i which has start time and duration defined, or the start time for any s_i with end time and duration defined.

After each duration update, there may be a time conflict with an adjacent or non-contiguous activity (as there may be uninitialized schedule entries lying between two time-conflicted activities). Our algorithm looks ahead to identify these conflicts at the time a duration constraint is applied.

For example, if $s_i[t_{end}] > s_j[t_{start}] \ni j > i$, then we shorten s_i to end at $s_j[t_{start}]$. Any activities between them will be reduced to zero length and effectively deleted. This procedure is applied both forwards and backwards.

Arguably, such a conflict could be treated as a schedule resolution failure, but we considered it realistic to simulate a situation where a user "didn't get around to something" and skipped some activities to meet an upcoming time constraint.

Uninitialized start and end times are updated in this way for all applicable s_i.

Algorithm 2 Solving duration constraints

procedure APPLYDURATIONCONSTRAINTS(s_i)
  if $\exists s_i[t_{start}] \wedge \exists s_i[d] \wedge \nexists s_i[t_{end}]$ **then**
    $j = arg\,min\{s_j[t_{start}]$ where $j > i\}$
    $s_i[t_{end}] \leftarrow \min(s_i[t_{start}] + s_i[d], s_j[t_{start}])$
  else if $\nexists s_i[t_{start}] \wedge \exists s_i[d] \wedge \exists s_i[t_{end}]$ **then**
    $j = arg\,max\{s_j[t_{end}]$ where $j < i\}$
    $s_i[t_{start}] \leftarrow \max(s_i[t_{end}] - s_i[d], s_j[t_{end}])$
  end if
end procedure

3. Solving Adjacency Constraints (Algorithm 3) The next step is to update time constraints between adjacent activities. Unlike the discrete activities recorded in the datasets, which often have gaps between them, activities in our model represent persistent spatial locations, so we do not allow gaps between activities. Activities with uninitialized start times are thus defined to begin when the previous task ends, and activities with uninitialized end times conclude at the start time of the next activity. If a gap exists between two activities, the earlier activity is extended to end at the start time of the later activity.

Algorithm 3 Solving adjacency constraints

procedure APPLYADJACENCYCONSTRAINTS(s_i)
  if $\exists s_i[t_{end}] \wedge \exists s_{i+1}$ **then**
    if $\nexists s_{i+1}[t_{start}]$ **then**
      $s_{i+1}[t_{start}] \leftarrow s_i[t_{end}]$
    else if $s_{i+1}[t_{start}] > s_i[t_{end}]$ **then**
      Extend current activity to fill gap
      $s_i[t_{end}] \leftarrow s_{i+1}[t_{start}]$
    else if $s_{i+1}[t_{start}] < s_i[t_{end}]$ **then**
      FAIL($overconstrained$)
    end if
  end if
  if $\exists s_i[t_{start}] \wedge \exists s_{i-1}$ **then**
    if $\nexists s_{i-1}[t_{end}]$ **then**
      $s_{i-1}[t_{end}] \leftarrow s_i[t_{start}]$
    else if $s_{i-1}[t_{end}] < s_i[t_{start}]$ **then**
      Extend previous activity to fill gap
      $s_{i-1}[t_{end}] \leftarrow s_i[t_{start}]$
    else if $s_{i-1}[t_{end}] > s_i[t_{start}]$ **then**
      FAIL($overconstrained$)
    end if
  end if
end procedure

4. Exit criteria The iteration of constraint resolution continues until there are no schedule entries left with unresolved start or end times, indicating success.

The algorithm can fail to meet this condition if the schedule template is underconstrained. Such cases can be detected if no updates have been made to the schedule after a

round of applying both duration and adjacency constraints. In these cases, the algorithm fails and no schedule is produced.

In other cases, the schedule may be overconstrained. That is, a time conflict between two schedule entries cannot be resolved by adjusting a single start or end time. Rather than implementing complex logic to handle these cases, we chose to allow the algorithm to fail in these situations.

C. Validation of Schedule Template

Because failure of the system to generate a valid schedule would be problematic at run time, such as during automated testing, we provide a design-time tool that designers can use to validate the schedule templates they create. Our validation algorithm is able to identify the following errors:

- Chronological errors in the template
- Under-constrained conditions where the time constraints are insufficient to define a complete schedule
- Over-constrained conditions where overlapping activities result in unresolvable time conflicts

Because the process of schedule generation incorporates randomness, our validation algorithm generates schedules at the minimum and maximum extremes of the variance ranges. If it can be confirmed that there are no overlapping activities or underconstrained situations given these boundary values, then the template is considered valid. These validation steps are especially important for preventing failures during automated operations like regression testing or generation of large-scale synthetic datasets of user schedules.

D. Integration with the robot simulator

As indicated in Fig. 2, the generated schedules were used as one input into Astro's simulator, along with a floorplan and a mapping of activities to locations. The simulations were executed in a photo-realistic simulator integrated with Gazebo, with the human simulation module moving each user to the target destination using an A* path planner with local obstacle avoidance for their scheduled activities based on the simulation wall-clock time.

The resultant simulation can be used in two ways. First, the simulator can be used online with the robot software, to test and evaluate robot behaviors executed in relation to the simulated user(s). Alternatively, the output can be captured as a rosbag and added to a test corpus for offline replay.

IV. Validation: Use Case Scenarios

To demonstrate the expressive power of the framework, we present a set of use cases representing common schedule patterns found in daily life and describe how they can be represented in a schedule template using our framework, with examples shown in Fig. 3. Where applicable, we also note examples observed in the datasets studied later in Sec. V-C.

A. Scheduled activities

Some activities, like scheduled meetings and appointments, have fixed start times. In the datasets, aside from wakeup times, rigidly-scheduled activities were almost never seen. Much more common were loosely-scheduled activities

	Activity	Start time	Start variance	Duration	Duration variance
1	Shower	8:00 AM	0:20	0:10	0:05
2	Breakfast			0:20	0:05
3	Brush teeth			0:02	0:00
4	Work				
5	Lunch	12:00 PM	0:00	0:40	0:10
6	Work			4:00	0:30
7	Watch TV				
8	Cook dinner			1:00	0:15
9	Dinner	6:30 PM	0:15	0:45	0:15

Fig. 3. Example schedule template illustrating use case scenarios. Start variance, duration, and duration variance are expressed in hours and minutes.

where start times varying within a range. These activities, such as meal times, might be tentatively planned for a specific start time, but with the expectation that it may vary significantly.

The proposed framework supports this flexibility by allowing variance to be specified for the activity's start time. In Fig. 3, rows 1 and 9 show activities with variable start times, whereas row 5 is rigidly scheduled for 12:00 PM.

B. Variable-length activities

While some activities observed in the datasets, like brushing teeth or taking medicine, were fairly fixed in duration, many daily activities required a variable length of time to finish. This was observed in the datasets for activities such as meals and computer work. In the proposed framework, the duration variance can be used to control this variability. In Fig. 3, "brush teeth" (row 3) is fixed-duration, but most other entries have nonzero duration variance.

C. Sequences of activities

It is also common for activities to follow each other in a sequence. For example, every public dataset we examined included a sequential morning routine, such as showering, breakfast, and then brushing teeth. Each of these typically followed the previous activity rather than varying independently. In such cases, it does not make sense to express start times explicitly, but rather to chain together activities.

In the proposed framework, this can be achieved by setting the start time for the first activity, but specifying only duration and duration variance for the following activities. An example of this is shown in rows 1-3 in Fig. 3.

D. End time constraints

Sometimes an activity is constrained to a fixed end time. For example, preparation for a meal may need to be completed by a certain time, or a morning routine may need to be finished by the time a person needs to leave for work.

This type of pattern is made possible by the backward-propagation aspect of the algorithm. For example, consider a "cooking" activity followed by a "dinner" activity (*e.g.* rows 8-9 of Fig. 3). If "dinner" is given a fixed start time of 6:30pm and "cooking" is specified to have a duration of 1 hour but no fixed start time, then the end time of "cooking" would be constrained to the start of "dinner" and its start time would be pushed back to 5:30pm to fit the constraint.

E. Filling the time available

In other situations, an activity can be extremely flexible in its duration, expanding to fill the time available. Activities like watching TV or reading can fall into this category.

This kind of activity can be expressed by leaving both start time and duration blank, allowing the propagated constraints from other activities to implicitly define start and end times for these flexible activities, *e.g.* in rows 4 and 7 of Fig. 3.

F. Comparisons with other approaches

There are many ways to specify schedules for simulations, and there are benefits and drawbacks to any method. We believe that our method has some advantages over other approaches, as follows.

A naive approach might specify exact start and stop times for tasks. Our approach provides control over variance of start times and durations, an important consideration for data which will be used to train machine learning systems, where variability in the distribution of data is needed for robustness.

Some approaches use programmatic logic to control task scheduling. While such approaches are extremely flexible, they are also complex. The method proposed here contains a small set of parameters which are still sufficient to express the task categories listed above. Arguably, this can be simpler for designers to manage, tune, and debug than a representation based on complex sets of rules and conditions.

Other approaches are stochastic, e.g. using probability distributions to specify when tasks should be scheduled. One drawback of such approaches is that it is difficult to ensure the sequential execution of dependent sequences of activities, which our proposed framework can provide.

V. Evaluation: Emulating captured data

Although our proposed method is primarily intended for developing handcrafted schedules, we also evaluated how effectively it could be used to create schedule data in imitation of an example dataset.

For this evaluation, we performed comparisons with several public datasets as well as one we captured on our own, evaluating how closely we were able to emulate the activity patterns in those datasets using our framework by measuring similarity between the schedules generated by our template and schedules from the original dataset.

A. Levenshtein distance as a similarity metric

To measure similarity between sequences, we used a similarity metric based on Levenshtein distance. Similar techniques have often been used to make quantitative comparisons between human motion trajectories [10, 2, 7, 18].

In our comparison, we first discretize each of the sequences to be compared into a state-chain representation, where states are sampled at regular 1-minute intervals and assigned the value of the action being performed at that time step. The Levenshtein distance, denoted by $L(S_1, S_2)$, between action state sequences S_1 and S_2 of length n is calculated as the minimum number of single-element edits required to transform S_1 into S_2.

A measure of similarity between the state chains can be obtained by normalizing the Levenshtein distance and subtracting it from 1, as follows:

$$sim_{Lev}(S_1, S_2) = 1 - \frac{L(S_1, S_2)}{n}$$

B. Comparison method

The datasets being compared consisted of collections of schedules, each containing several examples of daily schedules collected for one user. To compare **cross-similarity** between two collections of daily schedules, C_1 containing m schedule examples and C_2 with n examples, pairwise comparisons using sim_{Lev} were performed between the individual schedules, and the results were averaged.

$$sim_{cross}(C_1, C_2) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} sim_{Lev}(C_1^i, C_2^j)}{m \cdot n}$$

To help interpret the meaning of these similarity values, **self-similarity** scores were also computed within each collection of schedules. The self-similarity metric provides an approximate (although not strict) upper bound to the similarity that could be achieved when attempting to imitate a given collection of schedules. For a collection of schedules C containing n examples, average self-similarity over all combinations was computed as follows.

$$sim_{self}(C) = \frac{2}{n(n-1)} \sum_{1 \le i < j \le n} sim_{Lev}(C^i, C^j)$$

C. Datasets for comparison

We evaluated the ability of our framework to emulate examples from public datasets of daily activity schedules, as well as a dataset of space transitions in daily activity that we captured ourselves.

1) Public datasets: The following three public datasets were used in this comparison. When necessary, an activity designated "other" was added to fill gaps between activities.

HOMER Dataset: This dataset includes simulated activity schedules generated based on self-reported schedule information collected via Amazon Mechanical Turk. We used the "test" data for household 0, consisting of 12 activities tracked over 10 days. [15]

ADL Dataset: A dataset containing manually-labeled activities of daily living performed by two users on a daily basis in their own homes. We used data from user 1, comprising 10 activities tracked over 14 days. [13]

GENEActiv Dataset: This is a smartwatch accelerometer dataset. We used the manually-annotated activities for user 1, featuring 7 activities tracked over 11 days. [6]

2) Self-collected data: The above public datasets tracked daily activities, but reported activities were sparse and did not exactly correspond to locations. For our robot applications, our primary interest is in activities representing location changes within the home. Thus, we collected an additional dataset ourselves, tracking a single participant (one of the

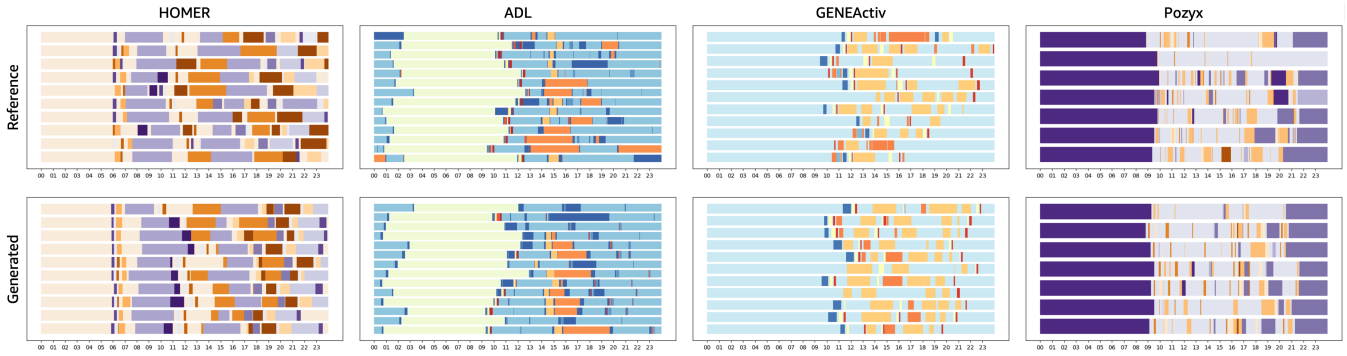

Fig. 4. Results of manual fitting of schedule templates to reference datasets. Each graph shows activities over a 24-hour period (activity names omitted for readability). The upper graph of each pair represents the original reference data, and the lower graph shows data generated by our algorithm.

authors of this paper) across 7 discrete regions of the home (8 states, including "not tracked"), over 7 days of daily activity.

Pozyx Dataset: In this dataset, the participant's real-time positioning data were collected using the Pozyx Real-Time Localizing System[1], which performs position tracking of portable ultra-wideband radio emitter tags worn by participants using the Time Differences of Arrival (TDoA) method with multiple fixed antennas. For each day of data collection, the participant started the tracking system between 9-10am and the positioning data were continuously generated for a collection window of 11 hours at frequency of 1Hz. The participant's location was tracked at room-level resolution, and we applied a low-pass filter to remove noisy space transitions with durations less than 1 minute.

As it is synthetically generated, the HOMER data is the least realistic dataset in our study. The GENEActiv and ADL datasets represent real activity data, but it was manually reported and only captures a few discrete actions. The Pozyx data represents raw space transitions in the home, so it is the noisiest data; however, it most closely represents the kind of data we plan to use with our framework.

D. Experiment

For each dataset, we manually created a schedule template to emulate the behavior patterns with our framework.

For each reference dataset to be evaluated, we compared three conditions: a random **baseline** (10 schedules), the cross-similarity score between the **generated** data (10 schedules) and reference data (7-14 schedules), and the **self-similarity** score within the reference data.

1) Baseline: As a baseline for this comparison, we created an algorithm that triggers a random user activity every 30 minutes. Although this is a naive approach, it is not unreasonable as a first-pass design for a simulator to run in a newly-designed environment when models of user behavior are not available.

2) Generated schedules: For this comparison, we manually created schedule templates designed to imitate the target datasets as closely as possible. Although an automated method for doing this would be useful, we leave that for future work. The goal of this evaluation is to demonstrate the expressive power of the proposed framework, which can be done via manual fitting.

For activities occurring around consistent times, t_{start}, v_{start}, d, and v_d were defined. For sequences of transient activities which seemed to be more dependent on each other than on a fixed time, only d and v_d were defined. Often, "default" states such as *other* or *Spare time* filled the time between other activities, so these were specified with all constraint fields empty, allowing them to fill the time between other activities. Intermittent activities were modeled by defining $v_d > d$, as this would cause the duration to sometimes go below zero, effectively deleting the activity on some days. Finally, templates were iteratively refined until the generated data appeared similar to the reference data.

3) Self-similarity: Whereas the baseline provided a practical lower bound for similarity measurements, we approximated an upper bound by computing a self-similarity score for each dataset as described in Sec. V-B, representing the degree of day-to-day variation in that dataset.

E. Results

Fig. 4 shows the results of our manual fit to each of the four datasets. This visualization was a helpful tool during the development of the templates, and it provides a broad sense of the degree of similarity which was achieved.

The quantitative similarity scores are shown in Fig. 5. In all cases, the similarity between the generated data and reference data was much higher than the random baseline, and comparable to the self-similarity within the reference set itself. The best match was to the GENEActiv dataset, with a 66.0% similarity score, slightly higher than the 65.6% reference data self-similarity and much better than the 16.4% similarity in the random baseline case. The match with lowest similarity was with the HOMER dataset, although the 45.3% similarity still beat the 43.0% self-similarity within that dataset and was much higher than the 10.2% similarity with the baseline case. The greatest improvement over the baseline was observed in the ADL dataset, with 65.9% similarity vs 11.9% in the baseline. From these results we conclude that the expressive power of the proposed framework was sufficient to closely approximate each of these datasets.

[1]https://www.pozyx.io/

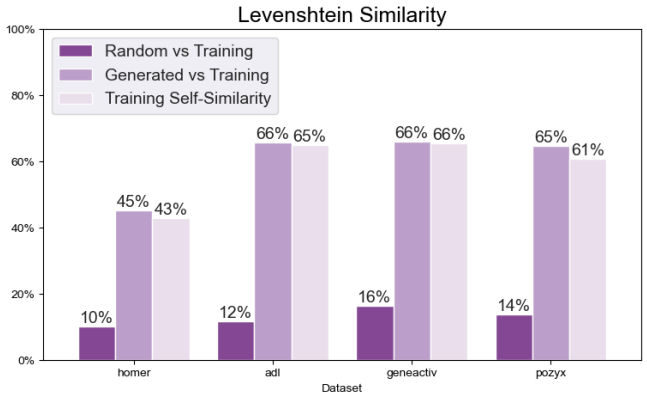

Fig. 5. Similarity scores for each dataset using the Levenshtein distance-based similarity metric, where higher similarity indicates a closer approximation of the real schedule. In all cases the similarity between the generated data and reference data was much higher than the random baseline, and comparable to the self-similarity within the reference set itself.

Cases where the cross-similarity scores exceeded the self-similarity scores likely indicate that the generated schedules had lower variability than the reference schedules and were hence closer to the "average" schedule than the original data. This is expected, as the original schedules included day-to-day variations of activity sequence that could not be reproduced using our method.

Qualitatively, we visually judged the generated sequences to closely resemble the reference data. Some anomalies in the reference data were not reproducible, but overall patterns appeared highly similar to the reference data in three key ways: First, several activity sequences such as morning and bedtime routines were successfully reproduced. Second, the lengths of individual activities and the relative distributions of activities through the day were roughly accurate. Finally, activity selection appeared to happen at the correct times of day and in the right sequence, with things like showers, meal times, commutes, and taking medication (activities tracked in the public datasets) occurring around the correct times.

VI. Discussion

A. Fitting Templates to Data

For the most part, it was not difficult to develop schedule templates to approximate a given input dataset. For activities which seemed to be associated with specific times of the day, such as waking up or eating meals, we used absolute start times with some variance. For activity sequences, we omitted start times and specified only duration and duration variance. For activities which occurred on some days but not others, we set the duration variance to be greater than the duration, resulting in the activity occasionally having zero duration (although this is hacky - it is not possible to model activities of fixed duration in this way). However, despite this flexibility, some phenomena were still difficult to model, such as the following examples.

1) Activity pattern changes: In the ADL dataset, the user ate lunch at home on some days, but went out for lunch on others. As noted above, our framework can handle a single activity occasionally being omitted, but it has no way to model a switch like that, where the "lunch" activity is sometimes replaced by a "leaving" activity with an earlier start time, and the two never occur together. In such cases, one possible solution could be to create multiple templates for different daily patterns, and have the schedule generator select randomly among them.

2) Stochastic vs scheduled activities: The Pozyx dataset was the most difficult for us to fit. In part this is because much of the activity appeared to be driven by random events, like going to the restroom or getting a glass of water, rather than scheduled activities that occur at a certain time or in a certain sequence. Rather than trying to account for such random events within a calendar schedule, we believe it could make sense to model them as a parallel phenomenon. The relative balance between scheduled and stochastic events seems likely to depend the individual user's lifestyle.

B. Limitations and Future Work

1) Similarity metrics: One limitation of using Levenshtein distance is the fact that it is based on total number of insertions or deletions. This means that this metric will have less sensitivity to short-duration activities than to longer-running activities, which can bias the scoring. However, we observed that as we improved our hand-tuned schedule templates over time, the similarity metric indeed converged towards the self-similarity score, indicating that the directionality of the metric was consistent with what we were evaluating, even if there might be some bias in its absolute magnitude.

Although the Levenshtein distance metric was included to provide a quantitative evaluation for this paper, in practice we found the graphical plots of daily schedules to provide much more useful feedback about the quality and nature of the fit, both for tuning the schedule templates and for validating the quality of the match through qualitative inspection.

2) Automating template fit: We expect that this template fitting process could be automated, but it is not trivial to do so, due to the insertion and removal of discrete schedule entries, the existence of optional fields, and the asymmetric nature of activities where the duration variance exceeds the duration. Due to these complications, we leave automation of this process for future work.

3) Simulating interaction between robot and human: As the data generated by this framework are meant for robot testing, one of the most important considerations is how to simulate interactions between users and the robot. Our view is that human-robot interactions need to be modeled separately in the simulator. The proposed framework can produce realistic locations and activity contexts for a simulated human, but any simulation of interaction with the robot needs to explicitly account for the user's motivation for interacting and what behaviors they will perform - for example, will the user only interact when the robot is in the same room, or will they leave the room to search for the robot?

4) Multiple users: The framework was designed to implicitly support multiple simulated users. By specifying unique locations for each user's activities, such as assigning

each user a different seat in the dining room or living room, the users can be simulated to coexist in the same home. However, the random variation added to their activity schedules is applied independently, meaning that activities cannot really be coordinated between the simulated users. For simulating situations such as a family coming together for a meal, this may be important to address in the future.

C. Differences between academia and industry

1) Intellectual property: We were unfortunately unable to share the code of our simulation framework as open-source, due to its proprietary nature. While perhaps more common in industry, such situations can also occur in academia, depending on funding sponsors or licensing restrictions. Despite these constraints, we believe it is still possible to share useful knowledge and findings.

2) Primary use cases: While academia rewards "interesting" research topics, research work in industry is driven by business goals. The framework presented here was developed for manual design and adjustment of schedule templates for testing. If this were a purely academic paper, it might be more likely to focus on automated or learning-based methods of generating schedule templates, which are interesting topics but not necessarily driven by such practical needs. Thus, we can probably expect research work from industry to trend towards the pragmatic, moreso than the theoretically interesting. This difference could be beneficial, introducing a useful diversity of perspectives into the academic discussion.

3) Numerical performance vs practical utility: In a typical HRI study, there would be a focus on p-values and human subjects experiments. From an industry perspective, our main focus is on understanding the practical utility and expressive capability of the framework to model a variety of scenarios, and we seek to understand its strengths and limitations rather than focusing on numerical results.

VII. Conclusion

In this work we have introduced a framework for simulating daily user activity at scale for testing and developing commercial robots. The framework enables manual tuning of daily schedules for developing both typical-use and corner-case tests for testing and quality assurance. Our case studies demonstrated the framework's expressive capability, and we also evaluated the approach's potential to emulate example data using both public and internally-captured datasets.

Overall, this work makes a significant contribution to the field of social robotics by providing a systematic approach for testing robot behaviors related to daily user activity. By helping to avoid training data bias, our approach has the potential to make robot behavior effective for a broad range of user households, making it a valuable tool for future research and development in social robotics.

ACKNOWLEDGMENT

We thank Tarun Morton, Aarthi Raveendran, and Abraham Dauhajre for their support of this research. This work was supported in part by the Office of Naval Research under Grant award N00014-21-1-2584.

References

[1] Abhijat Biswas, Allan Wang, Gustavo Silvera, Aaron Steinfeld, and Henny Admoni. Socnavbench: A grounded simulation testing framework for evaluating social navigation. *ACM Transactions on Human-Robot Interaction*, 2022.

[2] Simone Calderara, Uri Heinemann, Andrea Prati, Rita Cucchiara, and Naftali Tishby. Detecting anomalies in people's trajectories using spectral graph analysis. *Computer Vision and Image Understanding*, 115(8):1099–1111, 2011.

[3] Abubaker Elbayoudi, Ahmad Lotfi, Caroline Langensiepen, and Kofi Appiah. Modelling and simulation of activities of daily living representing an older adult's behaviour. In *Proc. 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, pages 1–8, 2015.

[4] Mary Ellen Foster and Jane Stuart-Smith. Social robotics meets sociolinguistics: Investigating accent bias and social context in hri. In *Companion of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, pages 156–160, 2023.

[5] Yannick Francillette, Bruno Bouchard, Kévin Bouchard, and Sébastien Gaboury. Modeling, learning, and simulating human activities of daily living with behavior trees. *Knowledge and Information Systems*, 62 (10):3881–3910, 2020.

[6] Enrique Garcia-Ceja, Ramon F Brena, Jose C Carrasco-Jimenez, and Leonardo Garrido. Long-term activity recognition from wristwatch accelerometer data. *Sensors*, 14(12):22500–22524, 2014.

[7] Marc Hanheide, Annika Peters, and Nicola Bellotto. Analysis of human-robot spatial behaviour applying a qualitative trajectory calculus. In *Proc. 2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pages 689–694. IEEE, 2012.

[8] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[9] Tom Hitron, Noa Morag Yaar, and Hadas Erel. Implications of ai bias in hri: Risks (and opportunities) when interacting with a biased robot. In *Proc. 2023 ACM/IEEE International Conference on Human-Robot Interaction*, pages 83–92, 2023.

[10] Takayuki Kanda, Dylan F Glas, Masahiro Shiomi, and Norihiro Hagita. Abstracting people's trajectories for social robots to proactively approach customers. *IEEE Transactions on Robotics*, 25(6): 1382–1396, 2009.

[11] Yuya Kaneshige, Satoru Satake, Takayuki Kanda, and Michita Imai. How to overcome the difficulties in programming and debugging mobile social robots? In *Proc. 2021 ACM/IEEE International Conference on Human-Robot Interaction*, pages 361–369, 2021.

[12] Hiroyuki Kidokoro, Takayuki Kanda, Dražen Brščić, and Masahiro Shiomi. Simulation-based behavior planning to prevent congestion of pedestrians around a robot. *IEEE Transactions on Robotics*, 31(6): 1419–1431, 2015.

[13] Fco Javier Ordónez, Paula De Toledo, and Araceli Sanchis. Activity recognition using hybrid generative/discriminative models on home environments using binary sensors. *Sensors*, 13(5):5460–5477, 2013.

[14] Maria Teresa Parreira, Sarah Gillet, Katie Winkle, and Iolanda Leite. How did we miss this? a case study on unintended biases in robot social behavior. In *Companion of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, pages 11–20, 2023.

[15] Maithili Patel and Sonia Chernova. Proactive robot assistance via spatio-temporal object modeling. In *6th Annual Conference on Robot Learning (CoRL)*, 2022.

[16] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs, 2018. URL `https://arxiv.org/abs/1806.07011`.

[17] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. URL `https://arxiv.org/abs/1912.01734`.

[18] M Alex Syaekhoni, Chanseung Lee, and Young S Kwon. Analyzing customer behavior from shopping path data using operation edit distance. *Applied Intelligence*, 48:1912–1932, 2018.

[19] Nathan Tsoi, Mohamed Hussein, Jeacy Espinoza, Xavier Ruiz, and Marynel Vázquez. Sean: Social environment for autonomous navigation. In *Proc. 8th International Conference on Human-Agent Interaction (HAI)*, November 2020.