Title:

ActivitySim: Large-scale Agent-Based Activity Generation for Infrastructure Simulation

Author(s):

Emmanuel Galli, CCS-3,
Stephan Eidenbenz, CCS-3,
Sue Mniszewski, CCS-3,
Christof Teuscher, Portland StateUniversity,
Leticia Cuellar, D-6

Intended for:

Agent-Directed Simulation (ADS'09), San Diego, CA, March 22-27, 2009 extra_author-1

Cuellar Leticia 217262 D-6

ActivitySim: Large-scale Agent-Based Activity Generation for Infrastructure Simulation

Emmanuel Galli, Stephan Eidenbenz, Sue Mniszewski, Christof Teuscher, Leticia Cuellar
{egalli, eidenben, smm, christof, leticia }@lanl.gov

30th October 2008

Abstract

We introduce ActivitySim, an activity simulator for a population of millions of individual agents each characterized by a set of demographic attributes that is based on US census data. ActivitySim generates daily schedules for each agent that consists of a sequence of activities, such as sleeping, shopping, working etc., each being scheduled at a geographic location, such as businesses or private residences that is appropriate for the activity type and for the personal situation of the agent. ActivitySim has been developed as part of a larger effort to understand the interdependencies among national infrastructure networks and their demand profiles that emerge from the different activities of individuals in baseline scenarios as well as emergency scenarios, such as hurricane evacuations. We present the scalable software engineering principles underlying ActivitySim, the socio-technical modeling paradigms that drive the activity generation, and proof-of-principle results for a scenario in the Twin Cities, MN area of 2.6 M agents.

1 Introduction

The United States' Department of Homeland Security aims to model, simulate, and analyze critical infrastructure and their interdependencies across multiple sectors such as electric power, telecommunications, water distribution, transportation, etc. Most infrastructure sectors rely on an underlying network that gets used by individual people and business entities, or alternatively speaking: there is a demand for the service that the network supplies. A non-exhaustive list of examples includes the phone network and the Internet that satisfy our the communication needs of the population, the road network that meets the demand for mobility, the electric power grid that satisfies our thirst for electricity. A full-fledged simulation capability that allows to run what-if scenarios as part of a course-of-action analysis requires the following, using the Internet as an illustrative example:

1. Accurate models of the *network topology*. For the Internet, the IP-level connectivity graph with capacity information satisfies this requirement.
2. Abstract models of the sector-specific *processes* on the network. For the Internet, we need models for the protocols used on the Internet (http, TCP, IP, email, Ethernet, 802.11, etc.).
3. Dynamic models of *demand* for the network service where changes in demand are an emergent property (as opposed to an input). For the Internet, we need a tool that provides realistic sets of Internet traffic sessions with origin, destination, and transmission size information. While some preliminary models for demand generation in this sense exist (e.g., call models such as described in [6]), this is a largely open research area.

Demand on networks is largely generated by people as part of their daily activities, such as driving to work, using energy to cook or to heat the house, using water and sewage systems, making phone calls, etc.[1] Thus, an accurate *model for the daily activities* of individuals is a pre-requisite for our simulation capability. An agent-based approach is the only modeling paradigm that allows us to generate demand shocks as an emergent property of the simulation. To stick to communication networks as an example, communication demand in emergency situations is different from a baseline demand because (i) individuals have evacuated in large numbers leading to a geographic demand shift, (ii) logistics, organization (such as organizing a return) and emotional turmoil leads to increased call volumes.

In this article, we describe our agent-based approach to activity modeling, called ActivitySim. Ac-

[1] There are cases, where demand is harder to attribute directly to activities of individuals, such as the water use in nuclear power plants. The point here is that a sizable fraction of demand is generated and directly attributable by individuals and individual households.

tivity modeling is ActivitySim is a scalable simulation tool. It relies on a synthetic, but statistically accurate population of the US that was obtained using disaggregation methods applied to US census data [2].

The ActivitySim modeling paradigm is based on a first-principle approach with respect to social modeling. Our main focus in this report is the model methodology, software implementation and scaling. Thorough validation and testing is reserved for future work.

ActivitySim is part of a family of simulation applications that follow the SimCore modeling paradigm. SimCore [4] is a scalable open-source discrete event-driven simulation engine. SimCore applications seamlessly integrate with each other by exchanging events. Other SimCore applications include sector-specific simulators, such as MIITS-NetSim, and the transportation simulator FastTrans as well as individual demand generation simulators such as SessionSim [3] for communication simulations. We give two examples of how SimCore applications work together: (i) ActivitySim provides input to SessionSim, which generates calls based on activities of agents. The SessionSim output (ie Internet sessions) are sent as events to MIITS -NetSim [8], which then routes these sessions over the network topology. (ii) Activities from ActivitySim that lead to location changes create a demand on the transportation network by generating a trip between locations. This can be fed as an event to FastTrans, which routes this trip over the transportation network and feeds back to ActivitySim at what time the trip was completed.

ActivitySim agents are *utility-driven*: each activity gives a certain amount of utility to an agent depending on how long the activity is being executed. Agents also have *priority functions* for activity types, where the priority of an activity intuitively increases usually with the time that has passed since the activity was last executed. Activity types have *constraints* that allow us to guide the timing of certain activities (such as work should happen during the business hours). As optional modules, we (i) allow agents to have *personality types* (guided by standard models from social sciences) and (ii) let agents guide their activity type selection by the *needs* that they want to satisfy. We describe the ActivitySim modeling in more detail in Section 3.

The main loop of an agent consists of planning and re-planning its scheduled activities and evaluation the resulting updated schedule with respect to the agent's objective function. The objective function takes into account predicted utility as well as priority and constraints violations. The schedule optimization step can be performed through your favorite optimization method, such as the gradient method, local search, simulated annealing, or taboo search. We describe the optimization loop in Section 3

We have implemented and tested ActivitySim on agent populations of up to 2.6M agents in a case for Twin Cities, MN. We present scaling results in Section 4.

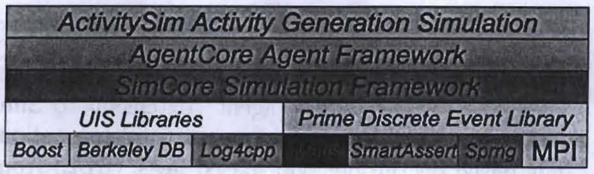

ActivitySim Activity Generation Simulation
AgentCore Agent Framework
SimCore Simulation Framework
UIS Libraries
Prime Discrete Event Library
Boost
Berkeley DB
Log4cpp
SmartAssert
Sprng
MPI

Figure 1: Overview of Software Architecture

2 The ActivitySim Architecture

ActivitySim is a C++ agent-based model that can run on workstations as well as high performance computing clusters. The supporting software architecture consists of agent and discrete event simulation (DES) frameworks, and libraries for graph processing, logging, partitioning, asserts, random number generation, and message passing (see Figure 1). More details follow.

2.1 The SimCore DES Framework

SimCore is a library for building large-scale distributed-memory, discrete event simulations (DES)[4] using the discrete event engine from the Parallel Real-time Immersive Modeling Environment (PRIME)[1] or passing events, event queue maintenance, and synchronization. It has previously been used for packet-level and session-level telecommunications network simulations[8] and fast queue-based transportation simulations, called FastTrans. The important concepts and classes within SimCore are Entity, Service, Info, and Profile. An Entity is a class that represents a simulation object such as a person, location, or facility. A Service is a class that is used to implement the behavior of an entity and operates like an event handler. Services are attached to Entities. An Info is a class that represents an event that can be scheduled and supplies additional data items and is processed by a Service. Infos are passed between Entities (more typically between the Services) to trigger an action. A Profile is a way of providing runtime specification of default parameter settings for different types of Entities, Services, and Infos.

2.2 The SimCore Agent Layer - AgentCore

AgentCore is a "reactive agent" extension to SimCore which adds classes for agent implementation. It is based on the behavior-based layer (BBL) from Muller's agent architecture[5]. The Agent class extends an Entity to include functionality (as methods) to perceive, think, and act and a Cognitor for processing rules or patterns of behavior. A Cognitor Service (known as the CognitorHandler) to the agent-part of a simulation object processes Think Infos (events) by calling the Agent's think method. An implementation would typically call its Cognitor's think method, but could certainly do more. Perceive means to gather the current values of the simulation world's facts or state variables. Think means to process the agent's production-rules or patterns of behavior by means of a Cognitor to perform actions and to cause other events to be scheduled. Act means to execute the actions determined by thinking. In implementations, the think method typically encompasses thinking and acting.

A pattern of behavior (POB) class has state, being active or inactive, has an activation condition based on the Agent's current facts, action code to execute if it is activated, a success condition that determines successful termination, and a failure condition that determines that a failure has occurred. A POB can have multiple execution steps. Each step can be interrupted or interruptions can occur between steps. Multiple POBs can be active at a time, but not all will be executing.

The Cognitor implements the control cycle model for thinking. An InterRap version is supplied (from Muller's agent architecture), though others can be added based on the beliefs, desires, and intentions (BDI) model, etc. InterRap processes POBs in its think method by managing completed POBs, checking for newly triggered POBs, and executing active POB steps.

Only Entities in a simulation that perform "intelligent" behavior should be agents, such as a Person entity changing and adapting their activity schedule. Other Entities such as locations and households do not need to be agents, though Agent-Entities and Entities can still interact through their Services.

2.3 ActivitySim

The *ActivitySim* agent-based simulation software provides daily activity schedule generation and execution for a synthetic population. It operates as a standalone model for population analysis, as well as coupled with other SimCore infrastructure models such as transportation and telecommunications to study inter-dependency effects in baseline and emergency scenarios. Persons, Locations, Households, and Zones comprise the entity types used to represent a model of a geographical area. A Person is also an agent that reasons about daily activity schedules. State (current activity and location), demographics, activity location choices, and a current schedule are all part of a Person. A Location tracks Persons as they participate in its' activities. A Household is associated with a Location, has aggregated income, and members (ex. family). A Zone is an aggregation of Locations used when selecting where an activity will take place.

The Persons, Locations, Households, and Zones are provided as input at runtime along with a specification of a set of activity types including utility and priority function parameters. An activity set can be very specific (ex. *sleep, personal care, lunch, dinner, leisure*) or more general (ex. *home, work, school, shopping, social recreation, daycare*). A Person's schedule consists of a sequence of these activities with start time, activity, location, and duration. A "Next Activity" POB triggered by a Think event causes each Person to reevaluate their activity schedule and add new activities as required (see Figure 2). A Person will "think again" during their next activity. In addition to the utility-based activity selection (described in Section 3), other methodologies can be added easily. Use of pre-generated schedules and random activity generation are also supported.

Single activity execution is an independent process from activity schedule generation. Each scheduled activity is executed as a sequence of four Person-level events and two Location-level events. A PersonDepart event causes a Person to complete their current activity and leave that Location (initiating a RemoveFrom Location event). A PersonTransport event allows a Person travel time between locations. A PersonArrive event places a person in the new activity at the new location (initiating a MoveTo Location event). Finally, a PersonDone event updates a Person's state per their new activity and location. When running on a parallel cluster, entities are distributed randomly across processors. The Person-level events are executed on the processor where the Person entity resides. Messaging is required between processors when "moving to" or "removing from" an activity at a Location.

Model output consists of configurable logging of details of entity and service creation, along with simulation progression. Event files show the individual event details for each Person and Location during activity selection and execution. Output is selectable for Person schedules and/or counts of Persons per activity for each Location at a single time or at regular

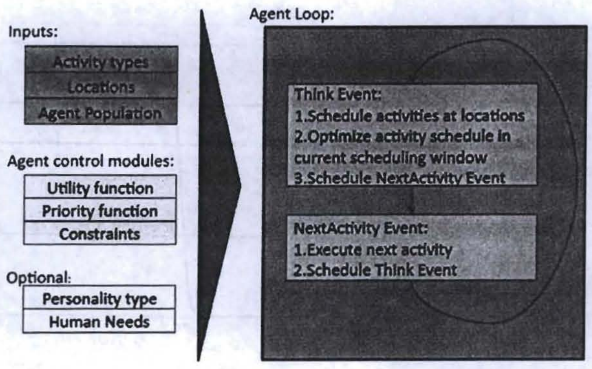


Figure 2: ActivitySim Inputs, Modules, and Think Loop

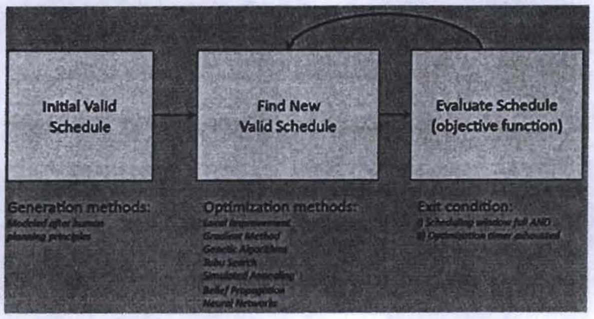


Figure 3: Optimization Loop

intervals.

3 ActivitySim Modeling Paradigms

We explain the ActivitySim modeling philosophy along the main building blocks as illustrated in Figure 2. Recall that ActivitySim takes as input a population of agents and a set of locations. Each resulting ActivitySim agent is characterized by a set of demographic attributes, such as age, gender, social status, personality type and home location. ActivitySim creates agents based on data sets derived from US census data (see Section 4 for details). Input locations are physical locations, characterized through latitude/longitude coordinates that represent either a business location (obtained from standard business data sources such as the Dunn&Bradstreet database) or a private residence.

As part of the main loop of an agent, it re-plans its scheduled activities during a Think-event that is executed at the end of an activity or at any time during an activity. In a Think-event, an agent optimizes its planned future schedule and evaluates it with respect to the its objective function. The objective function is influenced by the notions of takes utility functions, priority functions, constraints, needs function, and personality types, which we will explain towards the end of this section. Figure 3 illustrates the optimization loop that forms the Think-event in more detail. The schedule optimization step can be performed through your favorite optimization method, such as the gradient method, local search, simulated annealing, or taboo search.

The utility-driven activity selection model uses the modules of utility, priority and constraints with additional (optional) modules of needs and personality types that impact the schedule optimization loop shown in Figure 3. The notion of utility functions for activity selection builds on earlier work [7]. Generally an individual tries to optimize just a little set of activities while he/she has just a general idea of what he will do in the next days. So agents try to improve their current schedules, deciding often at last minute which activities they will perform. To grasp this concept intuitively, consider that we have a generic idea of what we will do in the next days (we know that we will go to sleep at the end of the day) but we can not say with great precision at what time we will perform it. Moreover an individual generally can schedule some particular activities that will happen in the far future (vacation, checkup, meeting) but will optimize every particular activity only when she/he is really close to the event. Inspired by these concepts, we have created a general model where given an initial schedule, every agent tries to optimize its schedule according to his personal characteristics. Starting with a basic set of already scheduled activities, the agent searches for a valid better schedule inside a limited future time window, which we call the sliding window. The new schedule is then evaluated by an objective function and if the new schedule is better then previous, the process is iterated. So the general algorithm can be described as follows:

1. *Starting Schedule*: A starting schedule is selected or it is the scheduled calculated in a previous step;

2. *Local Optimization*: Schedule is modified within the sliding window according to local optimization rules;

3. *Validity Check*: All activities that violate constraints are deleted from schedule;

4. *Schedule Evaluation*: Objective function evaluates the total schedule. If the objective function value is greater than the previous schedule, it becomes the new starting schedule. If the number of optimization steps has not yet reached a maximum threshold, the algorithm loops back to step 2;

5. *Append*: If no modifications to the original schedules have been accepted, a random activity is appended at the end of scheduled queue at a random future point in time.

The optimization algorithm is an algorithm which takes as input the scheduled *activies* inside the sliding window, an utility function, a priority function, a location selection function and moreover can consider the needs and personality types. Such algorithm can use any optimization scheme (neural networks, genetic algorithm, tabu search, simulated annealing, etc.) to come up with a new candidate schedule. For example, our current local optimization uses priority function to order unscheduled activities while the utility function is used to evaluate the duration; needs and personality type influence the sorting of unscheduled activities. The output of local optimization (a new schedule) is evaluated by the objective function. The idea behind the objective function is that we should penalize schedules that do not consider the value of utility, priority, and/or location. The modules that impact the objective function evaluation are described in more detail in the following subsections.

3.1 Utility Functions

The utility that an agent gains from performing an activity can ideally be explained in monetary terms. In our case the level of satisfaction cannot be mapped to a unit, but what matters are relative values of satisfaction. Every activity type is associated with a utility function. The utility of performing an activity is typically a function of the duration for which the activity is performed. There usually are lower and upper bounds for the utility that an agent can gain from performing an activity: consider sleep as an example, where even 5 minutes of sleep can have high utility and maximum utility is reached after about 8 hours for most people; sleeping 15 hours usually comes with less utility than sleeping only 8 hours. Utility functions have the following characteristics:

i) $U'(t) \geq 0$

ii) $U : \mathcal{R} \rightarrow [U_{min}, U_{max}]$ with U_{max} maximum marginal utility and with U_{min} minimum marginal utility;

iii) Let $y \in \mathcal{R} : U''(y) = 0 \Rightarrow$ for every $x \in \mathcal{R} : x \geq y$ $U''(x) \leq 0$

Figure 4 shows the utility functions as a function of duration that we have chosen for some of our test runs. Our utility functions have relatively steep transitions from a low level to a maximum level of utility. We have consulted with social scientists and economists to obtain these utility functions, but they should be considered a preliminary set. The functional form of the utility (adopted from [7]) is defined as follows for an activity type a:

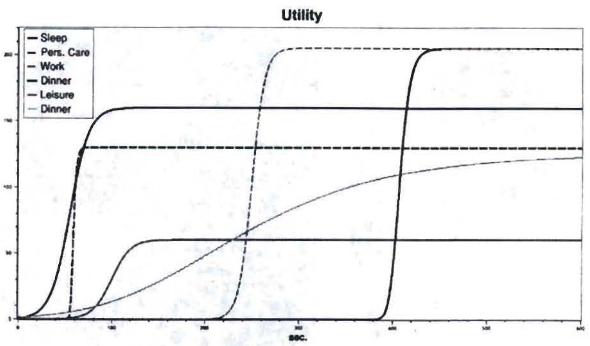


Figure 4: Example Utility Functions

$$U_a = U_a^{min} + \frac{U_a^{max} - U_a^{min}}{(1 + \exp[-\beta\,(\nu_a - \alpha_a)])^{\gamma_a}}$$

where

ν is the duration ($\nu \geq 0$)

U^{max} is the upper asymptote of the curve ($U^{max} > 0$)

U^{min} is the lower asymptote of the curve ($U^{min} \leq 0$)

α is the parameter of x-translation

β is the parameter of the slope

γ is the parameter of the inflection point.

3.2 Constraints

Constraints are non-negotiable conditions that a schedule must satisfy in order to be considered valid. Activities can be different from person to person and can be biased by age, personalities, marital status, family degree, employed status and so on. Thus, every agent chooses its activities from a set of activity types that is specifically tuned to the individual agent. ActivitySim imposes constraints on every activity type that must be obeyed during the activity selection. These constrains are minimum duration, maximum duration, earliest start time, latest start time, earliest end time and latest end time. Every location has a set of activity types that can be performed at that location (such as work, if it is a business location; or shopping, if it is a retail location). Thus similar constraints are imposed on a per-location-per-activity type basis regarding possible start and end times of activities. These constraints are akin to opening hours.

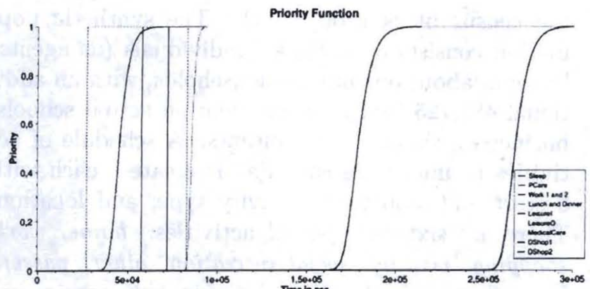


Figure 5: Example Priority Functions

3.3 Priority functions

A priority function is a function of time and represents the priority of an activity in a particular instant in time. This concept is particularly important during an emergency scenario where even though, for example, a person needs to eat, all evacuation activities must precede the activity eating. The priority function characteristics can be summarized as follows:

i) is function of time;

ii) is monotonically increasing;

iii) $\lim_{t\to\infty} P(t) = 1$;

iv) $\lim_{t\to 0} P(t) = 0$.

The priority function is set to zero as soon as an activity is performed. If an activity is selected always at the same time, the priority function is also a periodic function. Performing an activity type usually becomes more urgent with the time that has passed since the last execution of the activity. Thus, the time axis defined in these function represents the time since last execution of the activity type.

Figure 5 illustrates the sigmoid priority functions we have used in some our test runs. More formally, the priority function is similar to the previous utility function. If an *activies* has been already scheduled then the priority value is equal to zero. Let a be an unscheduled activity we have:

$$P_a = \frac{1}{(1+\exp[-\beta\ ((t_a - T_a) - \alpha_a)])^{\gamma_a}}$$

where:

$T_a = startTime_a + duration_a$ is the last time that activity a was performed

t_a is the current time

α, β, γ have the same meaning presented in the utility function.

3.4 Needs function

An individual is driven by his needs. A need is a dynamic characteristic of a person that decrease in the space of one or more days and it is directly influenced by some activities. So a person performs some activities to satisfy his needs. At the same time an activities can also influence negatively other needs. For example, the activity "work" influences negatively needs as "energy". A need is described by the inverse of the presented priority function. Performing specific activities will bring the need value to his optimum while others will decrease it to 0.

3.5 Location selection function

The location function represents the attractiveness of a zone. The function decrease with distance, time spent to reach it and/or other cost parameters (money, energy, ecc..) and increase with the number of activities that is possible to perform in such location as well as with personal preferences.

In our test runs the location function is simply a function of distance and decrease with it. Let a be the selected activity and i, j the current and next location respectively, we have:

$$L^a_{i,j} = \frac{1}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}}$$

3.6 Objective function

The objective function is a linear function that evaluates the new schedule. Such function takes as input all scheduled activities, the utility, priority, location function used also in the local optimization. Defining S as the set of scheduled activities

$$S = \{s_0, .., s_N\} \text{ with } N > 0$$

we can define the objective function as follows:

$$Obj(S) = \frac{\sum_{k=0}^{N} U(s_i) - \sum_{k=0}^{N} P(s_i) + \sum_{k=0}^{N} L(s_i)}{N}$$

with $U(s_i)$, $P(s_i)$ and $L(s_i)$ the utility, priority and location function value of scheduled activity s_i

The objective function evaluates a proposed schedule at the beginning of each new activity by giving positive points for achieved utility, negative points for incurred non-zero priority values at the beginning of each scheduled activity for every activity type, and by penalizing long travel times to new locations. Finding good schedules is obviously quite a challenge that calls for smart optimization schemes.

3.7 Optimization Algorithm

The optimization loop can either try to evaluate a large number of new schedules thru many iterations or it can attempt to invest cycles in finding a relatively small number of good new schedules by taking into account the utility, priority, and constraint modules. It is open which strategy is better and we are currently experimenting with light-weight simulated annealing approaches with very cheap neighbor functions. However, we have used the local optimization algorithm described below for our test runs, which invests many cycles into finding a few good new schedules.

The currently implemented algorithm uses set operators to modify the schedule. These operators are: *insert*, *substitution* and *adjustment*. The insert operator puts an unscheduled activities, respecting its constraints, between two already scheduled activities or between the last activities and the end of sliding window. The substitution tries to substitute a random scheduled activity with an unscheduled one. In such case is used the same start time and duration of substituted activity. The adjustment sets the duration close to its optimum value that is:

$$\nu = \alpha - \frac{1}{\beta}\lg(\frac{1}{\sqrt[\gamma]{1 - \frac{\varepsilon}{U_{max} - U_{min}}}} - 1) \text{ with } \varepsilon > 0$$

In our experiments we have used $\varepsilon = 10\%$ of U_{max}. The adjustement is always first applied to the activity that has the highest utility value and then to the other one. The goal of the algorithm is to fill the sliding window with the highest priority activities. Once the activity type has been selected, the location with highest location function value will be selected. The algorithm tries to fill the sliding window using insert and substitution operators. For every operator, the total utility function is computed as:
let $A = a_1, ..a_N$ scheduled activities inside the sliding window then:

$$U_{tot} = \sum_{i=1}^{N} U_i$$

We select the operator with the highest total utility. Since the insert and substitution operators do not consider optimum value for duration of an activity, the adjustment operator is applied to all scheduled activities.

4 Large example results

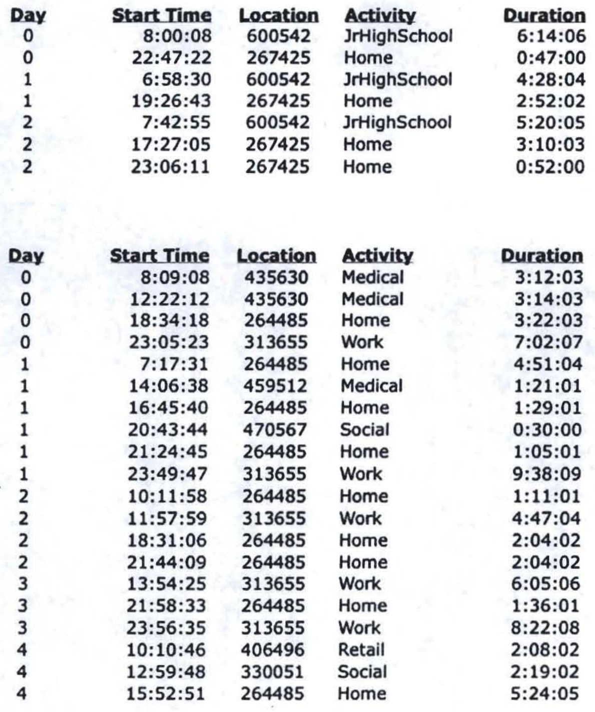
We used ActivitySim with the utility-driven scheduling to model daily activities in Twin Cities, MN. The synthetic population was constructed to statistically match the 2000 population demographics at the census block group level. The synthetic population consists of 2,592,906 individuals (as agents) living in about one million households, with an additional 487,725 locations representing actual schools, businesses, shops, or restaurants. A schedule of activities to undertake each day is created, each with a start and stop time, activity type, and location. There are sixteen types of activities: *home, work, shopping, visiting, social recreation, other, passenger server, school, college, dining out, service appointments, medical appointments, daycare, elementary school, junior high school, and high school.*

Information about the time, duration, and location of activities was obtained from the National Transportation Survey[2]. Each person agent was given an assigned set of locations based on the surveys. Locations were not provided for all activities, but only for a subset that were relevant to an agent's demographics and associated survey. Only one location was provided for home, work, passenger server, school, college, daycare, elementary school, junior high school, and high school. Four or more were provided for the remaining activity types. The *home* activity was allowed to start at any hour during the day and for any length of time. *Work* was limited to 4-10 hours at a time at any time during the day, while *junior high school* as limited to 4-6 hours at a time starting between 8 AM and 10 AM, ending between 11 AM and 4 PM. Two example schedules are shown in Figure 6. The first shows a child's schedule going to *junior high school* every morning. The second shows an adult's schedule who goes to *work*, spends time at *home*, has *medical appointments*, *goes shopping*, and participates in *social recreation*. Though multiple activities of the same type appear in sequence (ex. *home*), these initial results on a larger population are promising. More tuning of the utility and priority parameters is required to reduce the gaps.

The Twin Cities synthetic population was run on the LANL Institutional Computing parallel Coyote cluster (2,580 x AMD opteron nodes @ 2.6 GHz with 2 processors per node, Voltaire InfiniBand interconnect, 10.2 TeraBytes RAM) distributed across 8, 16, 32, 64, and 128 processors. Each was run for 10 simulated days. In all cases reading the input data took about 1.5 minutes. The average runtime per simulated day is shown in Figure 7. We see that using 32 processors is sufficient for running this problem size, with little additional gains for more.

5 Conclusion

The work presented in this paper is a large step towards developing robust activity generation and execution for synthetic populations as part of infras-



Day	Start Time	Location	Activity	Duration
0	8:00:08	600542	JrHighSchool	6:14:06
0	22:47:22	267425	Home	0:47:00
1	6:58:30	600542	JrHighSchool	4:28:04
1	19:26:43	267425	Home	2:52:02
2	7:42:55	600542	JrHighSchool	5:20:05
2	17:27:05	267425	Home	3:10:03
2	23:06:11	267425	Home	0:52:00

Day	Start Time	Location	Activity	Duration
0	8:09:08	435630	Medical	3:12:03
0	12:22:12	435630	Medical	3:14:03
0	18:34:18	264485	Home	3:22:03
0	23:05:23	313655	Work	7:02:07
1	7:17:31	264485	Home	4:51:04
1	14:06:38	459512	Medical	1:21:01
1	16:45:40	264485	Home	1:29:01
1	20:43:44	470567	Social	0:30:00
1	21:24:45	264485	Home	1:05:01
1	23:49:47	313655	Work	9:38:09
2	10:11:58	264485	Home	1:11:01
2	11:57:59	313655	Work	4:47:04
2	18:31:06	264485	Home	2:04:02
2	21:44:09	264485	Home	2:04:02
3	13:54:25	313655	Work	6:05:06
3	21:58:33	264485	Home	1:36:01
3	23:56:35	313655	Work	8:22:08
4	10:10:46	406496	Retail	2:08:02
4	12:59:48	330051	Social	2:19:02
4	15:52:51	264485	Home	5:24:05

Figure 6: Example of produced schedule

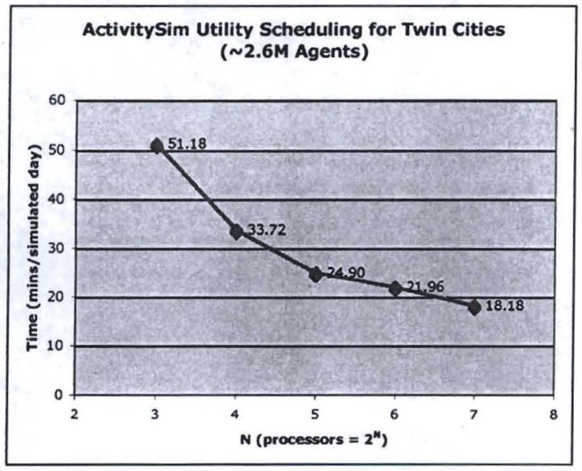


Figure 7: Running time per simulated day

tructure simulations for baseline and emergency scenarios. Tunable utility functions, priority functions, needs functions, and constraints allow for variability in each agent's daily activities adding a touch of realism. Future plans include the addition of other schedule generation and optimization strategies (such as the gradient method, local search, simulated annealing, or taboo search), scaling to larger populations (ex. southern California, the entire US), and analysis metrics for schedule evaluation.

References

[1] PRIME. Parallel Real-time Immersive Modeling Environment (PRIME). http://lynx.cis.fiu.edu:8000/twiki/bin/view/Public/PRIMEProject.

[2] US Department of Transportation (DOT), 2003. Bureau of Transportation Statistics. NHTS 2001 Highlights report BTS03-05.

[3] L. Kroc, S. Eidenbenz, and V. Ramaswamy. Sessionsim. Technical Report 07-0592, Los Alamos National Laboratory, 2007. Unclassified Report.

[4] L. Kroc, S. Eidenbenz, and V. Ramaswamy. Simcore. Technical Report 07-0590, Los Alamos National Laboratory, 2007. Unclassified Report.

[5] P. Muller. *The Design of Intelligent Agents: A Layered Approach.* Springer, 1996.

[6] V. Ramaswamy, S. Thulasidasan, P. Romero, S. Eidenbenz, and L. Cuellar. Simulating the national telephone network: A socio-technical approach to assessing infrastructure criticality. *Military Communications Conference, 2007. MILCOM 2007. IEEE*, pages 1–7, Oct. 2007.

[7] Vhang-Hyeon Joh, Theo A. Arentze and Harry J.P. Timmermans. Understanding activity scheduling and rescheduling behaviour: Theory and numerical illustration. *GeoJournal*, 53(4):359–371, Apr. 2001.

[8] R. Waupotitsch, S. Eidenbenz, P. Smith, and L. Kroc. Multi-scale integrated information an telecommunications system (miits): First results from a large-scale end-to-end network simulator. In *Proceedings of the Winter Simulation Conference*, http://portal.acm.org/citation.cfm?id=1218112.1218500, 2006.