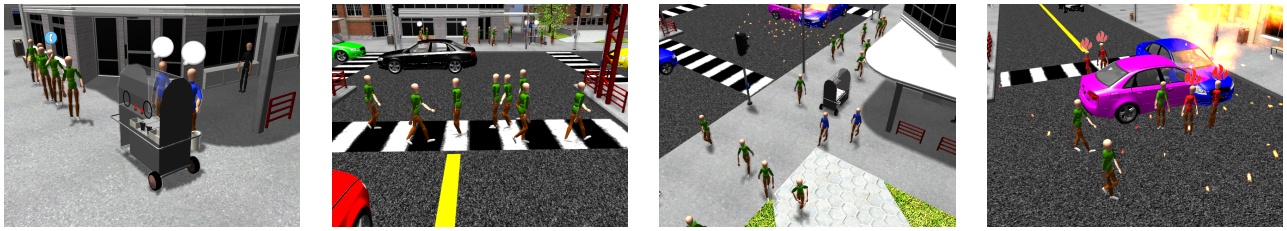
A Behavior Authoring Framework for Multi-Actor Simulations

Mubbasir Kapadia[1,2*] Shawn Singh[1,3†] Glenn Reinman[1‡] Petros Faloutsos[1§]
[1]University of California Los Angeles
[2]University of Pennsylvania
[3]Google Inc.



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

Figure 1: *Snapshots of a city simulation authored using our framework: (a) Actors queue up at a hot dog stand while the vendors talk to one another. In the meantime, a thief lies in the shadows waiting for an opportunity to steal the money from the stand. (b) Cars giving right of way to pedestrians. (c) Cautious actors run to a place of safety in the event of an accident. (d) Fire-fighters extinguish the fire while daring actors look on.*

Abstract

There has been growing academic and industry interest in the behavioral animation of autonomous actors in virtual worlds. However, it remains a considerable challenge to author complicated interactions between multiple actors in a way that balances automation and control flexibility.

In this paper, we propose a behavior authoring framework which provides the user with complete control over the domain of the system: the state space, action space and cost of executing actions. Actors are specialized using *effect* and *cost* modifiers, which modify existing action definitions, and *constraints*, which prune action choices in a state-dependent manner. *Behaviors* are used to define goals and objective functions for an actor. Actors having common or conflicting goals are grouped together to form a *composite domain*, and a multi-agent planner is used to generate complicated interactions between multiple actors. We demonstrate the effectiveness of our framework by authoring and generating a city simulation involving multiple pedestrians and vehicles that interact with one another to produce complex multi-actor behaviors.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Intelligent Agents

Keywords: Crowds, high-level behaviors, coordination, authoring

1 Introduction

Multi-actor simulation is a critical component of cinematic content creation, disaster and security simulation, and interactive entertainment. One key challenge is providing an appropriate interface to allow the user to *author* the behavior of autonomous actors that populate the simulated environment. For example, a user may want to author massive armies in movies, autonomous actors interacting in games, a panicked crowd in urban simulations, etc. Authoring is often a bottleneck in a production process, requiring the author to either manually script every detail in an inflexible way or to provide a higher level description that lacks appropriate control to ensure correct or interesting behavior. The challenge is to provide a method of authoring that is intuitive, simple, automatic, yet has enough "expressive power" to control details at the appropriate level of abstraction.

There are two components to authoring behaviors: (1) behavior specification, and (2) behavior generation. A behavior is specified as a scripted sequence of actions, desired goals state and constraints, finite state machines, or complex cognitive models. Then, a behavior generation module computes an action trajectory for all actors corresponding to the desired behavior(s). There exists a trade-off between specification and generation of behaviors. Detailed specification of behaviors (*e.g.*, scripted sequences of actions) require a simple generation module, while abstract specifications (*e.g.*, high-level motivations for actors) require more complexity and automation in behavior generation. In general, these methods suffer from the following disadvantages:

- **Flexibility**. Scripted behaviors are dependent on the current configuration of the actors and the environment and do not

*email: mubbasir@cs.ucla.edu
†email: shawnsin@cs.ucla.edu
‡email: reinman@cs.ucla.edu
§email: pfal@cs.ucla.edu

generalize easily to different scenarios.

- **Complexity**. Authoring complicated interactions between multiple actors becomes intractable in current approaches. For example, describing the collaboration of two actors to pick-pocket a victim could vary drastically based on the properties of the environment, the victim, or presence of other actors such as a police officer.

- **Effort**. There is no clear way of directing the trajectory of the story without defining behaviors for every participating actor. For example, a user may wish to specify that two vehicles meet with an accident without having to script the series of events that precede and follow the accident.

1.1 Related Work

Generating such behaviors in crowds has been studied extensively from many different perspectives [Badler 2008]. These methods represent different tradeoffs between the ease of user specification and the autonomy of behavior generation.

Scripted approaches [Loyall 1997; Mateas 2002] describe behaviors as pre-defined sequences of actions where small changes often require far-reaching modifications of monolithic scripts. Crowd approaches [Braun et al. 2003; Durupinar et al. 2008] provide interfaces to specify goals for *smart* avatars or map parameters to personality traits and examine the emergent behaviors in crowds. Smart Events [Stocker et al. 2010] is a method of externalizing behavior logic to authored events that occur in the environment.

Approaches such as Improv [Perlin and Goldberg 1996], LIVE [Menou 2001], and commercial systems like "Massive" describe behaviors as rules which govern how actors act based on certain conditions. These systems are *reactive* in nature, and typically produce pre-defined behaviors corresponding to the current situation. They are not designed to generate complicated agent interactions over the entire course of a lengthy simulation. Cognitive approaches [Yu and Terzopoulos 2007] use complex models such as decision and neural networks to model knowledge and action selection in virtual agents. They are not easy to edit or author, and they are often a result of a learning process.

The use of domain-independent planners [Fikes and Nilsson 1971] is a promising direction for automated behavior generation. Planning approaches provide automation at the expense of computation. Also, collaboration among agents requires the overhead of a centralized planner or the modeling of agent communication. Hence, current systems that use planners for behavior generation, *e.g.*, [Funge et al. 1999], are restricted to simple problem domains (small state and action spaces) with a small number of agents exhibiting limited interactions.

To our knowledge, no prior work provides a flexible means of specification with little effort, while generating complex interactions between multiple actors. The far reaching goal that still remains a considerable challenge is to provide an animator with the ability to easily orchestrate complicated "stories" between multiple interacting actors that can be easily customized and is portable across scenarios, with minimal user specification.

1.2 Our Approach

In this paper, we provide the user with complete control over the domain of the system: the state space, action space, and costs of executing an action. The environment and actors are described with *state metrics* that are affected by actions. The costs of actions are characterized by general *cost metrics*. Metrics and actions are extensible: users can create additional metrics that better interpret the simulation by applying operators on existing metrics, and user can create additional actions as metric modifiers. Existing actor definitions can be specialized using modifiers that modify the *effects* and *costs* of actions based on the current state. *Constraints* are used to enforce requirements on actors or on the story (*e.g.*, two cars must collide during the simulation). *Behaviors* are specified as a desired goal state and an objective function that an actor or group of actors must optimize. Actors having common or conflicting goals, are grouped together to form a *composite domain* and a heuristic search technique is used to plan in this domain to generate complicated multi-actor behaviors.

The main contribution of this paper is to combine the expressive nature of actions, action specializations, constraints, and behaviors (specification atoms) along with the automation of a heuristic search planner that works in the composite space of interacting actors. The intended audience for this framework is two-fold: domain specialists can define metrics and actions for a given scenario (state and action spaces), while end-users can specialize existing action definitions to add variation and purpose to their own simulation. The planner allows complex behaviors for multiple interacting actors to be generated with minimal user specification. Our method has the following benefits:

- **Modular and Natural Specification:** Domain specialists define the state and action space for different scenarios while end-users can specialize and constrain existing definitions to add variation and purpose to their simulation. Specializations and constraints can focus on different levels of abstraction and can be as general or specific as necessary. Behaviors are specified as goals and objectives for actors that are triggered based on their current state.
- **Cooperative and Competitive Planning:** Complicated interactions between multiple actors can be authored by simply specifying common or contradicting goals for actors in the scenario. Our method automatically clusters actors that have cooperative or conflicting goals to define a *composite* state and action spaces. This avoids the complexity of modeling communication between actors or the need for explicit scripting of cooperation schemes in agents. Collaborative behaviors arise as a solution found by the planner which minimizes the combined cost of actions of all agents in the composite space.
- **High-Level Story Specification:** Constraints can be used to enforce requirements at various points in the simulation without explicitly scripting preceding and succeeding events. This allows users to make incremental changes in the specification in an isolated manner.

1.3 Comparison to Previous Work

Our method strikes a happy medium between flexibility of specification and automation of behavior generation by allowing the users to work at different levels of abstraction. Users can control fine details in their simulation by designing the state and action space of actors or simply direct the high-level details by specializing existing actor definitions. Our search method generates complicated multi-actor interactions without the need of an expensive global optimization for all actors in the scenario.

2 Behavior Specification

Figure 2 presents an overview of our framework. A domain expert first defines the problem domain (state and action spaces) of the actors in the scene. Next, a director specializes the actors using

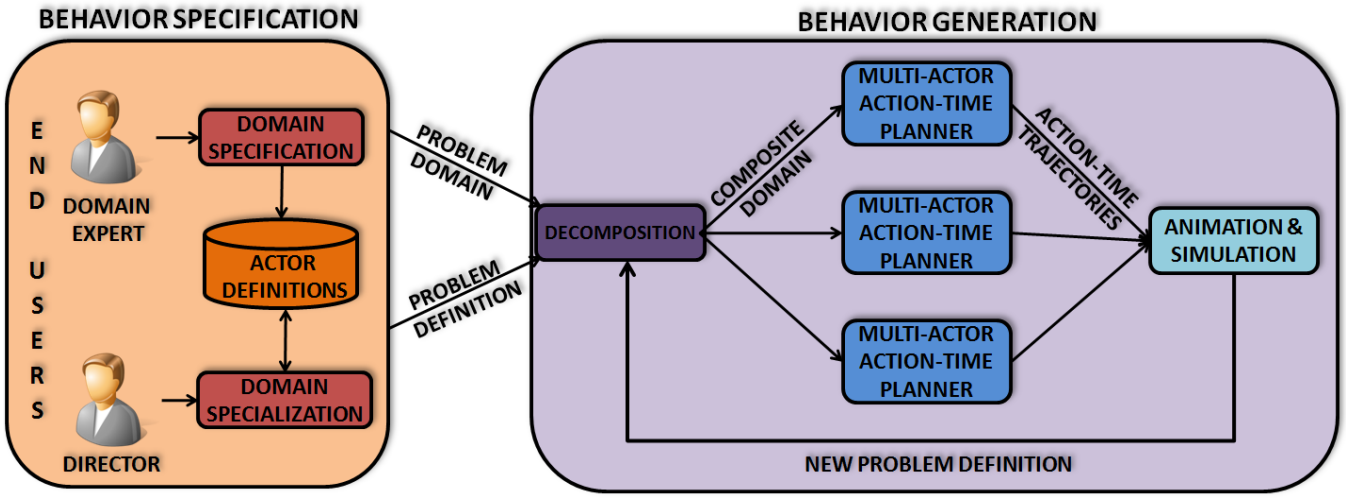


Figure 2: *An overview of the framework.*

modifiers, constraints, and behaviors. Actors with dependent goals or constraints enforcing their interaction are grouped together into a composite domain, forming a set of independent domains. For each of these domains, a multi-actor planner generates a trajectory of actions for all actors in that domain that satisfies the composite goal while optimizing the individual objective of each actor. The result of each search is combined into a global plan which is executed to generate the resulting simulation.

2.1 Domain specification

Domain specification is the lowest level of abstraction at which a user can work to author behaviors. It entails defining the state space, the action space and, costs of executing actions for actors in a scene. An actor is an entity which has a state and can affect the state of itself or other actors by executing actions. Different actors in the same scenario may have different domain specifications. For example, a traditional actor can be defined to simulate pedestrians in a virtual environment, while the environment can be defined as an actor which can be used to trigger global events such as a natural disaster that would affect the state of other actors in the scenario.

State Space. We represent the state space of an actor using *metrics* – physical or abstract properties of an actor that are affected by the execution of actions. Users can extend metrics by applying operators on existing metrics to provide an intuitive understanding of the properties of the simulation. Let $\{m_i\}$ define the space of metrics for all actors in the scenario.

Costs. Costs are a numerical measure of executing an action. Different actions can affect different cost metrics by different amounts. Examples of cost metrics include distance, energy, etc. Let $\{c_i\}$ define the space of costs.

Action Space. The action space of an actor is a set of actions which it can perform in any given state. Actions affect one or more metrics of an actor. An action has the following properties: (1) pre-conditions which determine if an action is possible in a given state, (2) the effect of the action on the state of the actor as well as target actors and, (3) the cost of executing an action.

Action *actionName* (*parameters*) {
  Precondition: *conditions on elements of* $\{m_i\}$ *or* $\{c_i\}$
  Effect: *effects on elements of* $\{m_i\}$
  Cost Effect: *effects on elements of* $\{c_i\}$
}

Modifier *modifierName* {
  Precondition: *conditions on elements of* $\{m_i\}$ *or* $\{c_i\}$
  Effect: *effects on elements of* $\{m_i\}$ *or* $\{c_i\}$
}

Constraint *modifierName* {
  Precondition: *conditions on elements of* $\{m_i\}$ *or* $\{c_i\}$
  Constraint: *conditions on elements of* $\{m_i\}$
}

Behavior *behaviorName* {
  Precondition: *conditions on elements of* $\{m_i\}$
  Goal: *conditions on elements of* $\{m_i\}$
  Objective Function: *values of* $\{w_i\}$
}

Figure 3: ***Domain Specification and Specialization.*** *Actions, modifiers, constraints and, behaviors defined using our framework.*

2.2 Domain Specialization

End users can re-use of existing actor definitions across different scenarios by specializing actors in a state-dependent manner without modifying the original definition. Our intent is that an author will spend a majority of his time at this level of abstraction where he can specify and generate vastly different, purposeful simulations in an intuitive manner with minimal specification. We provide three methods of specializing actors: (1) effect modifiers, (2) cost modifiers and, (3) constraints.

Modifiers. Users can specialize the effects and costs of executing an action in a state-dependent manner using modifiers. For example, an *effect modifier* can be placed on elderly actors to reduce their normal speed of movement. *Cost modifiers* indicate what actions are in an actor's best interest at a particular state. For example, a cautious actor can be authored by increasing the cost of actions that may place the actor in danger (e.g. enter a burning building).

Here, the notion of *danger* would be a user specified metric in the state space of these actors.

Constraints. Constraints are used to enforce strict requirements on actors in a scenario. Constraints can be used to prune the action choices of an actor in a particular state. For example, constraints can be used to prevent pedestrians from walking on the road and obey traffic signals. Constraints can also be placed on the trajectory of the simulation to author specific events (e.g. two cars must collide), generate complex interactions between actors, and direct the high-level story.

2.3 Behavior State Machine Specification

A particular behavior state defines the current goal and objective function of an actor. The goal of an actor is a desired state that the actor must reach, while the objective function is a weighted sum of costs that the actor must optimize. The objective function o of an actor is specified by setting the weights $\{w_i\}$ of the different cost metrics $\{c_i\}$, and is defined as $o = \min(\sum_i w_i \cdot c_i)$. A user can define multiple behaviors for an actor which are activated depending on the current state.

3 Behavior Generation

The entire problem domain is first decomposed into a set of independent composite domains, where actors in one composite domain have common or contradicting goals (Section 3.1). A multi-actor action-time planner generates action trajectories for each actor in the composite domain which optimizes their individual objectives and satisfies the composite goal (Section 3.2). Finally, models of virtual humans and vehicles are simulated and animated to follow these trajectories to generate the authored scenario (Section 3.3).

3.1 Domain Decomposition

The entire problem domain is decomposed into a set of independent composite domains where actors with dependent goals or constraints enforcing their interaction are part of one composite domain. The composite state space is the cartesian product of the states of each actor and the composite action space is the union of the actions of each actor in the composite domain. The composite domain is denoted by Σ.

A particular problem instance $\mathsf{P} = (\Sigma, s_0, g, \{o_i\})$ is defined by determining the initial state s_0, composite goal g , and objectives $\{o_i\}$ of each actor in the composite domain. The composite goal, g, is the logical combination of the goals for all actors in the composite domain. Common goals are combined using an $\wedge$ operator, indicating that all actors must satisfy their goal. Contradicting goals are combined using an $\vee$ operator, indicating that any one of the actors must satisfy their goal. A composite goal can thus be one of the following:

- A single objective for a single actor (e.g. get a hot dog).
- Multiple objectives for a single actor (e.g get a hot dog and meet a friend at the park).
- Common objectives for a group of actors (e.g two actors collaborating to lift a heavy load).
- Conflicting objectives between actors (e.g. the objective of the thief is to steal from the victim while the objective of the victim is to protect his money).
- Combination of common and conflicting objectives (e.g. two actors collaborating to corner a third actor).
- One or more desired events during the course of the behavior (e.g. a thief must be caught).

Procedure **Search**$(\Sigma, s^0, g, \{o_i\})$
Input: Σ = The composite domain
Input: s^0 = The composite start state
Input: g = The composite goal objective
Input: $\{o_i\}$ = The cost objectives for each actor in the composite space
Output: π = Action trajectories for all actors that meets desired behavior
$\mathtt{OPEN} = \{s^0\}$
$\mathtt{CLOSED} = \{\phi\}$
while *OPEN* $\neq \{\phi\} \vee N < N_{max}$ **do**
  $N = N + 1$
  $s = \arg_s \min(f(s))$ where $s \in \mathtt{OPEN}$
  goalReached = **isGoalConditionSatisfied**(g, s)
  if *goalReached* **then**
    π = **generatePath**(g,CLOSED)
    return π
  end
  $\mathtt{OPEN} = \mathtt{OPEN} - \{s\}$
  $\mathtt{CLOSED} = \mathtt{CLOSED} \cup \{s\}$
  $\mathtt{A}$ = **generateCompositeActions**(s)
  foreach *a in A* **do**
    t = **getTime**(s)
    s' = **simulate**$(s,$a,t,t+1$)$
    if $g(s') > g(s)+$***costFunction***$(s,$a,t,t+1$)$ **then**
      $g(s')$ = $g(s)+$**costFunction**$(s,$a,t,t+1$)$
    end
    $f(s') = g(s')+$ **heuristicFunction**(s')
    $\mathtt{OPEN} = \mathtt{OPEN} \cup \{s'\}$
  end
end
if $N == N_{max}$ **then**
  $s = \arg_s \min(f(s))$ where $s \in \mathtt{OPEN}$
  π = **generatePath**(s,CLOSED)
  return π
end
else
  return NULL
end

Algorithm 1: Heuristic Search Algorithm

Since the planner works in the composite space of multiple actors, complicated interactions between actors that may be collaborating or competing with one another can be generated, without the need of global centralized planning across all actors in the scenario. Even though the actions of an actor only affect the state space of the composite domain it belongs to, the possibility of an action is determined by considering the global state space of all actors in the scenario. This is done to ensure collision-free trajectories between two independent plans. As a result, the action trajectories generated for actors in different groups can be overlayed to generate a complete simulation.

3.2 Multi-Actor Action-Time Planner

The input to the planner is the problem definition $\mathsf{P} = (\Sigma, s_0, g, \{o_i\})$, described above. The heuristic search process generates a trajectory of actions for all actors in the composite space which meets the composite goal, g while optimizing the individual objectives, $\{o_i\}$ of all actors in the group. Our planner extends traditional planning approaches [Fikes and Nilsson 1971] as follows: (1) it works in the composite space of multiple actors with competitive or collaborative goals, (2) it explicitly takes time into account with different actions taking variable amounts of time and actions of different actors overlapping and, (3) it uses an automatically derived heuristic estimate to speed up the search process.

Overview. Using a heuristic planner, the search tree expands as follows: For the current state, a set of possible *transitions* is first

generated. Each transition represents the forward simulation of the actions for all actors in the composite space by one time step, where actors are simultaneously executing actions. The planner chooses a transition by minimizing the sum of the *total cost* of the transition and the *heuristic estimate* to reaching the composite goal. The cost of a transition is computed such that the action chosen by an actor optimizes its own objective function. When the planner reaches a state which satisfies the composite goal g, it returns the generated plan.

Transitions. A transition represents the simultaneous execution of actions chosen by all actors in the composite domain by one time step. A transition is said to be *valid* and ready to simulate if all the actors have a valid action that they are executing or ready to execute. An action for a particular actor possible if all the following conditions are met: (1) the actor is currently not executing an action, (2) the preconditions of the action are satisfied and, (3) no constraints prohibit the action. For a valid transition, the actions of all actors are simulated for one time step in a random order. The explicit modeling of time in the action definition results in overlapping actions, actions being partially executed (action failure) and actors choosing to perform new actions while other actors are still performing their current action.

After the simulation of a transition, an actor may find itself in one of the following states: (1) *Success.* The action is successfully completed and the actor must chose a new action in the next time step. (2) *Executing.* The action is partially executed at the end of the simulation routine for that time step. (3) *Failure.* The preconditions of the action are negated as a result of the execution of actions of other actors. The action is said to have failed and the actor must choose a new action.

Cost Function. The cost of simulating a transition $\{a_i\}$, at state s, in the time interval $(t, t+1)$ where a_i is the action chosen by actor i in the composite domain is given by: $\sum_i o_i(\{c_j\})$, where $\{c_j\}$ are the values of the cost metrics for simulating action, a_i for actor i at state, s, from time, t to $t+1$, and o_i is the objective function of actor i.

Heuristic Function. The heuristic function is used to provide a cost estimate from the current state to the goal state. Since our system is domain independent (a user may specify any state space and action space), manually defining heuristics for such a domain becomes cumbersome. Even worse, the goal specification is not a single state or set of states but a condition that must be satisfied. The automatic derivation of heuristics [Bonet and Geffner 2001] has been extensively studied in task planning literature and is shown to scale well for large problem domains. Our design of a heuristic function is fairly straightforward and efficient. We first relax the preconditions on the actions (all actions are deemed possible at any given instant of time) and do a fast greedy search for a trajectory of actions that takes the planner from the current state to the goal. The sum of the cost of all actions is the heuristic, h for that particular state, s.

3.3 Animation and Simulation Engine

The output of the planner is a set of action-time trajectories for each actor in the simulation. These trajectories provide information regarding the spatial position of each actor at a given point of time as well as the current action of the actor. Since the planner takes time into account and always considers the global state of all actors (not just the actors in the composite domain) while generating a valid plan, the resulting solution is guaranteed to be collision-free. Hence, a simple steering algorithm is first used to simulate coin shaped agents to accurately follow the paths. Next, an animation system is used to animate models of virtual humans and vehicles along the simulated paths. Characters are animated by transitioning between walk, run and stop animations based on the speed of movement. Also, animations are used to visualize the current action being performed by an actor (e.g., a thief stealing from money from the hot-dog stand).

3.4 Behavior Generation Algorithm

The algorithm used to generate multi-actor behaviors is described below:

1. Define Actors, $\mathtt{Ac}_i = \langle S_i, A_i, C_i, B_i \rangle$, where S_i is the state space, A_i is the action space, C_i is the set of constraints and modifiers, and B_i is the set of behaviors defined for actor i.
2. Determine Composite Domains, $\mathtt{CD}_j = \langle S_j^c, A_j^c, C_j^c, B_j^c \rangle$, where $S_j^c = \{S_1 \times S_2 \times ... \, S_n\}$ is the composite state space, $A_j^c = \bigcup_{i=1}^{n} \{A_i\}$ is the composite action space, $C_i^c = \{C_i\}$ is the set of specializations, and $B_j^c = \{B_i\}$ is the set of behaviors defined for all actors $i = 1$ to n in the composite domain $\mathtt{CD}_j$.
3. For each Composite Domain, $\mathtt{CD}_j$
   (a) Define Search Domain, $\Sigma = (S^c, A^c, C^c)$.
   (b) Determine initial state in the composite space of all agents, $s^0 = \bigcup_{i=1}^{n} s_i^0$
   (c) Determine active behaviors, b_i for each actor, i in composite domain, $\mathtt{CD}_j$. The active behavior for each actor determines the goal, g_i and the objective function o_i.
   (d) The composite goal, g is the logical combination of the goals, $\{g_i\}$ for all actors in the composite domain. Common goals are combined using an $\wedge$ operator, indicating that all actors must satisfy their goal. Contradicting goals are combined using an $\vee$ operator, indicating that any one of the actors must satisfy their goal.
   (e) If no behavior is active for actors, **Return**.
   (f) Solve for sequence of actions π by performing a search, $\pi =$ **Search**$(\Sigma, s^0, g, \{o_i\})$, where Σ is the search domain, s^0 is the composite start state, g is the composite goal, and $\{o_i\}$ are the objective functions for each actor.
4. Combine plans for all domains, $\Pi = \pi_1 \cup \pi_2 \cup ... \pi_n$.
5. Execute Global Plan, Π.
6. Determine new states of all actors.
7. Repeat Steps 2-6.

4 City Simulation

We demonstrate the effectiveness of our framework by authoring a car accident in a busy city street and observing the repercussions of the event on other actors that are part of the simulation, such as the old man and his son, whose behaviors are automatically generated using our framework.

4.1 Actor Specification

We first define the state space and action space of three actors in the scenario: (1) a generic pedestrian, (2) a vehicle and, (3) a traffic signal.

Pedestrian: The state of a pedestrian comprises its position, orientation, speed of movement, mass, and, a collision radius. In addition, pedestrians have the following abstract metrics: hunger, safety, amount of money. These metrics are variables whose values are modified by actions. The *Move* action (Figure 5(a)) is defined

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

Figure 4: *Interaction between thief and the vendors: (a) The thief steals money from the hot dog stand when the vendors walk away (because of the accident). (b)-(d) The vendors collaboratively work together to surround the thief in the alley and manage to catch him.*

to kinematically translate an actor and has an associated distance and energy cost. The routine `CheckCollisions(..)` returns false if the *Move* action causes the pedestrian to enter a state of collision. Additional actions (e.g. `Eat`) can be associated with different metrics (e.g. hunger). The pedestrian is given a simple behavior to move towards a specified goal position ((Figure 5(b)) while minimizing distance and energy cost. The goal positions are randomly generated to produce a realistic city simulation with wandering pedestrians. Additionally, the pedestrians monitor the state of a traffic signal which coordinates the movement of pedestrians and vehicles at an intersection ((Figure 5(c)).

Vehicles: The state and action space of vehicles is defined similarly to simulate their movement. In addition, they have a metric `damage` which increases if a vehicle collides with another vehicle. Vehicles are constrained to stay on the roads, give right of way to pedestrians, and obey the traffic lights.

Traffic Signals: A traffic signal represents an environment actor that models the simulation of the traffic signals at the intersection. It has a single metric `signal state` which is the current state that the traffic signals at the intersection are in. An action, `ChangeTrafficSignal` (Table 1(a)) determines the state of the traffic signal based upon the current simulation time. The pedestrian and the vehicles query the signal state in order to follow the traffic signals.

4.2 Actor Specialization

These generic actors are specialized as follows:

- **Fire-fighters:** Fire-fighter actors are specialized pedestrians whose common goal is to extinguish any fires that may be present in the city block. These actors have proportionally lower weights for the safety metric – this implies that they can move closer to a dangerous situation in performing their jobs. Table 1 (g) outlines the behavior description for fire-fighters.
- **Elderly:** An elderly person is specialized by creating an effect modifier which reduces their walking speed. In addition, the elderly actor has an effect modifier which makes him follow his grandson (Table 1(e)).
- **Grandson:** The objective of the grandson is to escort his grandfather at all times and to keep him away from danger (e.g. car accidents, oncoming traffic and other pedestrians). We achieve this by introducing a simple script, `ProtectGrandfatherB` which changes the objective function of the grandson to include the safety cost metric of the grandfather as well.
- **Cautious Actor:** A cautious actor is authored by increasing the cost of actions that may place him in danger (Table 1(b)).
- **Daring Actor:** A daring actor is authored by lowering the cost of actions that may place him in danger (Table 1(c)). Here, danger is a user-defined metric that is associated with each actor in the scenario.

```
Action Move(Velocity : v, TStep: dt) {
  Precondition:
   CheckCollisions(self.position + vdt) == false;
  Effect:
   self.position = self.position + vdt;
  Cost Effect:
   self.energyCost = ½(self.mass)|v|²;
   self.distanceCost = |v|dt;
}
```

(a)

```
Behavior GoalBehavior {
  Precondition: self.goalPosition ≠ 0;
  Goal: self.goalPosition;
  Objective Function:
   min(self.distanceCost + self.energyCost);
}
```

(b)

```
Constraint PedSignalConstraint {
  Precondition: true;
  Constraint:
   if ((signal.signalState == 0
   ∧CrossingRoad(self.position,C))
   ∨(signal.signalState == 1
   ∧CrossingRoad(self.position,A))
   ∨(trafficSignal.signalState == 2
   ∧CrossingRoad(self.position,B)))
    true;
   else false;
}
```

(c)

Figure 5: *A generic pedestrian with a simple Move action (a), a behavior to go to a specified goal position (b), and a constraint to follow the traffic signals (c).*

- **Street Vendor:** A street vendor is given the behavior of manning his hot dog stand and ensuring that his money is not stolen.
- **Thief:** The goal of the thief actor is to make money while minimizing his risk of getting caught (Table 1(j)). He has an action *Steal* (Table 1(k)) which allows him to steal money from the hot dog stand. In addition, a cost modifier *RiskCostModifier* (Table 1 (l)) assigns a high cost to stealing in the presence of other actors. He is therefore looking for an opportunity to steal when the vendor is distracted.
- **Reckless Vehicle:** A reckless vehicle is modeled by introducing a high cost to moving at slower speeds and relaxing the constraints of obeying traffic signals and collisions with other vehicles. Table 1 (n) and (o) define the effect and cost modifiers for a reckless vehicle.

4.3 Results

We populate a city block with pedestrians and vehicles using our framework. Furthermore, actor specializations provide an easy and intuitive way to add variety and purpose to the virtual world. We observe pedestrians walking along the sidewalks in the city in a goal-oriented manner (satisfying hunger by getting a hot dog, going to the park to meet a friend, stopping to take a look at objects of interest) while obeying constraints and modifications (obey traffic lights, avoid collisions, stay off the streets etc).

In order to add drama to the simulation, we introduce constraints on the trajectory of the entire simulation. First, we introduce a constraint, *AccidentC* (Table 1(m)), that an accident must happen (i.e. two vehicles must collide). A simulation is generated where two reckless vehicles collide with one another, resulting in a fire that stops the traffic at the intersection (Figure 1(d)). Cautious pedestrians who are near the accident run away to a safe distance in panic or walk away calmly (depending on their specialization) while daring actors approach the scene of the accident.

The car accident triggers the activation of the behaviors in the fire-fighters who run to the location of the fires. They work together collaboratively to extinguish both fires (a result of the planner working in the composite domain). Upon noticing the accident, the vendor runs to a place of safety (high cost modifier on safety). As soon as the thief notices that the vendor has left his stand, he slowly approaches the stand, steals the money and runs to a place of safety.

We vary the simulation result by introducing other specializations or modifying existing ones. In a first take, we define the objectives of the two vendors to minimize safety cost as well as the cost of being robbed as individuals (Table 1(h)). When the accident happens, they run to a place of safety while keeping the stand in eyesight. As soon as they see the thief stealing the money, they both chase after him. However, the thief has a head-start and runs away. This is because the planner generates solutions that tries to achieve the objective of each vendor independently. Hence, we observe that in the composite domain of the thief and two vendors, the thief succeeds. In a second take, we modify the objectives of the vendors to minimize the cost of both being robbed (Table 1(i)). The common goal of the vendors implies that the planner searches for a solution that optimizes their combined objectives. As a result, the two vendors cooperate to corner the thief in an alley (Figures 4(a)-(d)).

Performance and Implementation Details. We demonstrate 106 actors in the city simulation, with 15 cars and 91 pedestrians. Based on constraints, goal definitions and spatial locality, the following composite domains are defined: (1) 15 cars and 4 fire-fighters, (2) old man and son and, (3) generic pedestrians grouped together based on spatial locality. Dividing the problem domain into smaller composite domains reduces the branching factor of the search by two orders of magnitude, reducing an intractable search problem to smaller, more feasible searches. The plans for each of these domains is then overlayed to form the complete solution. The performance results are provided in Figure 6. The amortized performance of our behavior generation framework for the results shown in the video is 0.02 seconds per actor per second of simulation generated.

We use a hybrid programming model that combines scripts and C++ classes to generate the results described in the paper. For example, `Pedestrian`, `Vehicle` and, `TrafficLight` are derived implementations of an abstract `Actor` class. Actors are specialized using XML scripts which facilitate rapid prototyping and experimentation to see how varying script parameters affect the simulation. For ease of exposition, we use pseudocode to describe the scripts in the paper. From our experience with authoring the city simulation, we observe that the overhead of defining actors in C++ is mitigated as we spend a majority of our time at the scripting layer incrementally adding and modifying specializations to generate the result that we desire.

Currently, scripting is limited to the XML schema definition and requires end users to be aware of the constructs and variable names that are defined for actors. Also, there are no automatic validations to ensure that scripts do not override or conflict with each other. For future work, we will develop a graphical user interface that will expose the constructs for actor definition and specialization to end users in an easy and intuitive fashion.

Number of actors	106
Number of composite domains	12
Max # of actors in a composite domain	19
Total generation time	219 sec
Max generation time for one domain	76 sec
Min generation time for one domain	8 sec
Generation time per actor	2.06 sec
Length of output simulation	95 sec
Amortized time per actor per second	0.02 sec

Figure 6: *Performance Results.*

5 Discussion

In this paper, we present a multi-actor planning framework for generating complicated behaviors between interacting actors in a user-authored scenario. Users define the state and action space of actors and *specialize* existing actor definitions to add variety and purpose to their simulation. Actors with dependent goals are grouped together into a set of independent composite domains. For each of these domains, a multi-actor planner generates a trajectory of actions for all actors to meet the desired behavior. We author and demonstrate a simulation of more than one hundred actors (pedestrians and vehicles) in a busy city street and inject heterogeneity and drama into our simulation using specializations. With help from the community, we envision a growing open-source library of actor definitions that can be re-used to direct completely new simulations, with a few simple specializations.

Limitations. The behaviors generated by our framework are heavily dependent on the manner in which actors are grouped together and the weights of the objectives. For example, authoring interactions between the old man and firemen would necessitate the old man and firemen belonging to the same planning domain. For future work, we will adopt a dynamic clustering strategy in which actors may be re-grouped based on their current states.

One of the major design choices of this framework was the use of a multi-actor planner which prohibits its use in interactive applications such as games. The major reason for adopting a centralized approach was to facilitate the authoring of complex multi-actor interactions which would require a complex model of communication and prediction between actors in a decentralized system. We are currently investigating the use of anytime planners as well as parallel search algorithms for both centralized and agent-based planning in an effort to achieve real-time performance.

Acknowledegements

We wish to thank the anonymous reviewers for their comments. The work in this paper was partially supported by NSF grant No. CCF-0429983. We thank Intel Corp., Microsoft Corp., and AMD/ATI Corp. for their generous support through equipment and software grants.

Author Biographies

Mubbasir Kapadia Mubbasir Kapadia is the associate director and post-doctoral researcher at the SIG Center for Computer Graphics Lab at University of Pennsylvania. He received his Ph.D. and M.S. in Computer Science at the University of California, Los Angeles in 2011 with a dissertation on authoring and evaluating autonomous virtual human simulations. He received a Bachelors in Computer Engineering from D.J Sanghvi College of Engineering, Mumbai, India in 2007. His current research focuses on the application of automated planning and machine learning principles to simulate autonomous virtual humans.

Shawn Singh Shawn is a PhD Candidate in computer graphics at UCLA under Professor Petros Faloutsos and Professor Glenn Reinman. His research includes pedestrian navigation in crowds as well as architectures for real-time ray tracing and photon mapping. He received his M.S. in computer science at the University of Southern California, and his B.S. in music and computer science from the College of William and Mary.

Glenn Reinman Glenn Reinman is an associate professor in the Department of Computer Science at University of California, Los Angeles. He received his Ph.D. and M.S. in Computer Science at the University of California, San Diego in 2001 with a dissertation on fetch optimizations for aggressive out-of-order superscalar processors. He received a B.S. in Computer Science and Engineering from the Massachusetts Institute of Technology in 1996.

Petros Faloutsos Petros Faloutsos is an assistant professor at the Department of Computer Science at the University of California at Los Angeles. He received his PhD degree (2002) and his MSc degree in Computer Science from the University of Toronto, Canada and his BEng degree in Electrical Engineering from the National Technical University of Athens, Greece. Professor Faloutsos is the founder and director of the graphics lab at the Department of Computer Science at UCLA. Faloutsos' research interests include graphics, animation, virtual humans, and surgical robotics. Professor Faloutsos is also interested in computer networks and he has co-authored a highly cited paper on the topology of the Internet. Professor Faloutsos is a member of the Editorial Board of the Journal of The Visual Computer and has served as a Program Co-Chair for the ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2005. He is a member of the ACM and the Technical Chamber of Greece.

References

BADLER, N. 2008. *Virtual Crowds: Methods, Simulation, and Control (Synthesis Lectures on Computer Graphics and Animation)*. Morgan and Claypool.

BONET, B., AND GEFFNER, H. 2001. Heuristic search planner 2.0. *AI Magazine 22*, 3 (Fall), 77–80.

BRAUN, A., MUSSE, S. R., DE OLIVEIRA, L. P. L., AND BODMANN, B. E. J. 2003. Modeling individual behaviors in crowd simulation. *Computer Animation and Social Agents 0*, 143.

DURUPINAR, F., ALLBECK, J., PELECHANO, N., AND BADLER, N. 2008. Creating crowd variation with the ocean personality model. In *Proceedings of AAMAS'08*, 1217–1220.

FIKES, R. E., AND NILSSON, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*.

FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *Proceedings of ACM SIGGRAPH*, 29–38.

LOYALL, A. B. 1997. *Believable agents: building interactive personalities*. PhD thesis, Pittsburgh, PA, USA.

MATEAS, M. 2002. *Interactive drama, art and artificial intelligence*. PhD thesis, Pittsburgh, PA, USA.

MENOU, E. 2001. Real-time character animation using multi-layered scripts and spacetime optimization. In *Proceedings of ICVS '01*, Springer-Verlag, London, UK, 135–144.

PERLIN, K., AND GOLDBERG, A. 1996. Improv: a system for scripting interactive actors in virtual worlds. In *Proceedings of ACM SIGGRAPH*, ACM, New York, NY, USA, 205–216.

STOCKER, C., SUN, L., HUANG, P., QIN, W., ALLBECK, J. M., AND BADLER, N. I. 2010. Smart events and primed agents. In *IVA*, 15–27.

YU, Q., AND TERZOPOULOS, D. 2007. A decision network framework for the behavioral animation of virtual humans. In *Proceedings of the ACM SIGGRAPH/EG Symposium on Computer Animation*, 119–128.

```
Action ChangeTrafficSignal {
 Precondition:
  true;
 Effect:
  timeMod = currentTime % 100;
  if (timeMode <= 35)
    self.signalState = 0;
  else if (timeMode  <= 70)
    self.signalState = 1;
  else self.signalState = 2;
}
```

(a)

```
CostModifier CautionCM {
 Precondition:
  ∃ a: a.danger > 0 ;
 Cost Effect:
  self.safetyCost = max(a.danger);
}
```

(b)

```
CostModifier DaringCM {
 Precondition:
  ∃ a: a.danger > 0 ;
 Cost Effect:
  self.safetyCost =
     MAX_COST - max(a.danger);
}
```

(c)

```
EffectModifier DaringEM {
 Precondition:
  true;
 Effect:
  a = arg max(a.danger) ;
  self.goalPosition = a.position;
}
```

(d)

```
EffectModifier ElderlyEM {
 Precondition:
  true;
 Effect:
  self.speed = min(self.speed,1.0)
}
```

(e)

```
EffectModifier FollowEM(Actor : a) {
 Precondition:
  true;
 Effect:
  self.goalPosition = a.position;
}
```

(f)

```
Behavior FireFighterB {
 Precondition:
  ∃ a ∈ Actors: a.fire > 0;
 Goal:
  ∀ a ∈ Actors a.fire = 0;
 Objective Function:
  min(0.3·self.safetyCost
      + self.distanceCost
      + self.energyCost);
}
```

(g)

```
Behavior IndividualVendorB {
 Precondition:
  true;
 Goal:
  self.money >= 100;
 Objective Function:
  min(self.stolenCost);
}
```

(h)

```
Behavior CooperativeVendorB {
 Precondition:
  true;
 Goal:
  self.money >= 100 ∧
  otherVendor.money >= 100;
 Objective Function:
  min(self.stolenCost
    + otherVendor.stolenCost);
}
```

(i)

```
Behavior ThiefB {
 Precondition:
  true;
 Goal:
  self.money >= 100
 Objective Function:
  min(self.distanceCost
     + self.energyCost);
}
```

(j)

```
Action Steal(Actor a, Amount: m){
 Precondition:
  m <= a.money ∧
  DistanceBetween(self,a) < 1.0;
 Effect:
  a.money = a.money - m
  self.money = self.money + m
Cost:
  self.stealCost = m;
  a.stolenCost = m;
}
```

(k)

```
CostModifier RiskCostModifier {
 Precondition:
  true;
 Effect:
  self.stealCost +=
  max Dist(self.position,a.position)
}
```

(l)

```
Constraint AccidentC {
 Precondition:
  true;
 Constraint:
  // Two vehicles must collide
  // at some point in time
  ∃ a1, a2 :
  IsAVehicle(a1) ∧
  IsAVehicle(a2) ∧
  Distance(a1, a2) < 5.0;
}
```

(m)

```
EffectModifier RecklessVehicleEM {
 Precondition:
  true;
 Effect:
  self.collisionRadius = MIN;
  self.followSignals = FALSE;
}
```

(n)

```
CostModifier RecklessVehicleCM {
 Precondition:
  true;
 Effect:
  // low cost for traveling
  // at MAX_SPEED
  self.speedCost
    = MAX_SPEED-self.speed;
}
```

(o)

Table 1: *Scripts used to author the city simulation.*