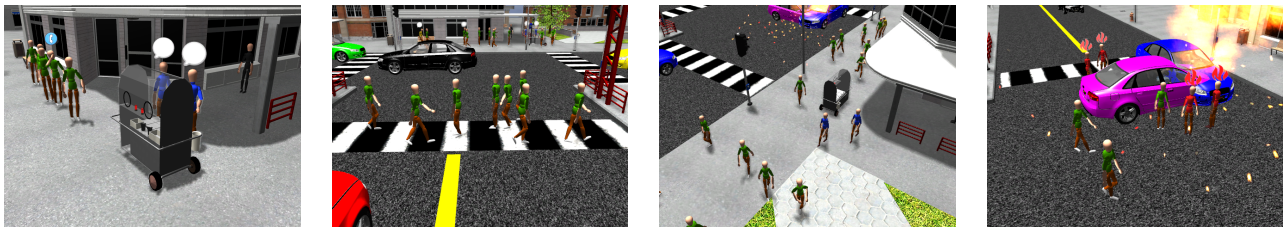
Technical Report: 100041
Multi-Agent Behavior Specification and Generation



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

Figure 1: *Snapshots of the 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, the 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. (e) Firefighters 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 automatically generate complicated interactions between multiple actors in a customizable way with minimal user specification.

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 heuristic search technique is used to generate complicated multi-actor behaviors. Using our method, users can work at any level of abstraction – from specifying scripted sequences of actions, goals, constraints on trajectories of one or more agents, to specifying high-level motivations for an entire scene. 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 goal state, finite state machines or using 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 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.

To our knowledge, no prior work provides a flexible means of spec-

ification with little effort, while generating complex interactions between multiple actors. The far reaching goal that still remains a considerable challenge is this: 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.

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 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 space), 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 space. 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 preceeding and succeeding events. This allows users to make incremental changes in the specification in an isolated manner.

2 Related Work

Behavioral animation in crowds has been studied extensively from many different perspectives. A comprehensive survey of the most representative work can be found in [Badler 2008]. There are two components to authoring behaviors: (1) behavior specification, and (2) behavior generation. Using a specification interface (*e.g.*, CML [Funge et al. 1999], BML [Vilhjálmsson et al. 2007], STRIPS [Fikes and Nilsson 1971], TAEMS [Horling et al. 1999]), an author defines the state and action spaces of the actors in a scenario. Behavior specification can be classified as follows:

Scripted action sequences. This method provides complete control to the animators in defining behaviors as pre-defined sequences of actions. HAP [Loyall 1997] provides a language for expressing behaviors as sequences of primitive actions. [Loyall et al. 2004] builds on this work and proposes a motion synthesis system that combines motion data with authored procedures. [Mateas 2002] focuses on the integration of behavioral agents in a global story, in an effort to build interactive, dramatic worlds. The work in [Vilhjálmsson et al. 2007] provides a framework for generating behaviors in conversational agents. These approaches require the user to hand-script each behavior and small changes often require far-reaching modifications of monolithic scripts.

Personalities and goals. Most crowd approaches (e.g. [Sung et al. 2004; Braun et al. 2003]) use goals and parameters to add heterogeneity to their simulations. The work in [Durupinar et al. 2008] maps parameters to personality traits and examine the emergent behaviors in crowds. The work in [Bindiganavale et al. 2000] uses natural language instructions to define goals for *smart* avatars.

Behavior State machines. Behaviors are defined as rules which govern how actors act based on certain conditions. Improv [Perlin and Goldberg 1996] and LIVE [Menou 2001] use nested scripting to control the motion of a character. SmartBody [Thiebaux et al. 2008] employs a hierarchical animation controller for behavioral animation in conversational agents. Massive [Massive Software Inc. 2010] is a commercially available system which allows users to create probabilistic state machines for defining agent behaviors. These sytems are *reactive* in nature: i.e. they produce predefined behaviors corresponding to the current situation and are not equipped to generate complicated agent interactions that pan out over the course of an entire simulation.

Cognitive Models. The work in [Funge et al. 1999] proposes a cognitive model to govern what a character knows, how that knowledge is acquired and how it can be used to plan actions. Artifical life approaches [Shao and Terzopoulos 2005] model motor, perceptual, behavioral, and cognitive components for an autonomous pedestrian. Decision networks are used in [Yu and Terzopoulos 2007; Yu 2007] for complex cognitive modeling subject to uncertainty. [Blumberg 1997] propose an ethologically inspired model of action selection and learning for behavioral animation. However, these models cannot be easily designed and modified by non-experts.

Behavior generation depends on the method of user specification. These methods of authoring represent different tradeoffs between the ease of user specification and the autonomy of behavior generation. Manual approaches (e.g. [Perlin and Goldberg 1996; Menou 2001]) provide fine-grained control over agent behaviors at the expense of generalization and automation. Automated approaches (e.g. [Yu and Terzopoulos 2007; Lau and Kuffner 2005; Fikes and Nilsson 1971; Erol et al. 1994]) generate state-dependent action sequences that lead autonomous actors towards their user-defined goals. In these approaches, the behavior generation task is more complicated, and details may not be easy to control.

The use of domain-independent planners [Fikes and Nilsson 1971; Blum and Furst 1995; Erol et al. 1994] is a promising direction for automated behavior generation. The automatic derivation of heuristic estimators [Bonet and Geffner 2001] to guide the search is a popular technique in domain-independent planning, and is the method of choice in this paper. 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 [Funge et al. 1999; Decugis and Ferber 1998] using planners for behavior generation are restricted to simple problem domains (small state and action space) with a small number of agents exhibiting limited interaction.

Our method. 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.

The rest of this document is organized as follows: Section 3 presents the constructs for domain specification and specialization. Section 4 describes the underlying framework for behavior generation. In Section 5, we demonstrate the efficacy of our technique by authoring a complicated city simulation involving pedestrians and vehicles.

3 Behavior Specification

Our framework allows the author to define the state space, action space and action costs for actors in the scene. Actors are specialized using *effect and cost modifiers* – which modify existing action definitions, and *constraints* which prune action choices depending on the state of the actor. *Behaviors* are used to define the goals and objective functions for an actor. Section 3.1 defines the method of domain specification. Specializing actors is described in Section 3.2, while Section 3.3 outlines the method of specifying behaviors.

3.1 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. For example, `distance_to_target` is derived from the positions of the actor and a target actor which is used for specifying behaviors such as `follow` or `seek`. 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) preconditions 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* $\{c_i\}$
}

3.2 Specialization

In our method, we focus on the 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.

Effect Modifiers: Users can specialize the effect of an action depending on the state of the actor and the environment. For example, an effect modifier can be placed on elderly actors to reduce their normal speed of movement. Similarly, a reckless vehicle can be authored by reducing its collision radius, not follow signals and move at greater speeds.

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

Cost Modifiers: Just as effect modifiers specialize the effect of an action, cost modifiers specialize the cost of an action depending on the state of the actor and the environment. They 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. running a yellow light) or keep an actor in danger (e.g. remaining in a burning building). A daring actor on the other hand, could be authored by lowering the cost of actions that may place the actor in danger. In both of these examples, the notion of *danger* would be a user specified metric in the state space of these actors.

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

Constraints: Constraints are a powerful and general method for enforcing strict requirements on actors in a scenario. Constraints on a single actor typically restrict 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 on multiple actors can be used to author specific events (e.g. two cars must collide), generate complex interactions between actors and, direct the high-level story of the simulation.

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

3.3 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.

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

4 Behavior Generation

The previous section describes the specification of the problem domain (state space, action space, action-costs and, modifiers) and problem definition (start state and behaviors) for all actors. The entire problem domain is divided into composite domains of actors having common or contradicting goals. A heuristic search technique is used to independently optimize the objectives of all actors in each composite domain.

4.1 Behavior Generation Algorithm

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 heuristic search technique plans a trajectory of actions for all actors in that domain that satisfies the composite goal while optimizing the objective of each actor. The result of each search is combined into a global plan which is executed to generate the resulting simulation. An algorithm outlining our framework is described below:

1. Define Actors, $\mathsf{Ac}_i = \langle S_i, A_i, C_i, B_i \rangle$, 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, $\mathsf{CD}_j = \langle S_j^c, A_j^c, C_j^c, B_j^c \rangle$, 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 CD_j.
3. For each Composite Domain, 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, 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.

4.2 Composite Search Domain and Problem Definition

The search domain, Σ comprises the state space S, the action space A, and the set of specializations C that are defined for all the actors in the composite domain (Step 3(a)). Given a search domain Σ, a problem definition is specified as $(\Sigma, s^0, g, \{o_i\})$ where s^0 is the composite start state, g is the composite goal and $\{o_i\}$ is the set of objective functions for all actors in the composite space. The composite start state, s^0 is the cartesian product of the initial states, $\{s_i\}$ of each actor (Step 3(b)). The current behavior of all actors in the composite space is determined to obtain the goal and objective functions for each actor (Step 3(c)). The composite goal, g is the logical combination of the goals for each individual actor (Step 3(d)). Common goals are combined using an $\wedge$ logical operator and are used to generate collaborative behaviors. Contradicting goals are combined using an $\vee$ logical operator and are used to simulate competitive behaviors. 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).

The heuristic search planner solves for a sequence of actions for all actors in the composite domain which optimizes the *individual* objective functions of each actor while satisfying the *composite* goal. Since our technique works in the composite space of multiple actors, complicated interactions between multiple actors that may be collaborating to satisfy a common goal or competing with one another can be generated. However, actions for an actor are chosen to optimize its individual objective function. This ensures that actors always do what is in their best interest in "trying" to achieve their goals even if the resulting simulation does not meet their individual goal.

4.3 Heuristic Search Algorithm

Algorithm 1 illustrates the working of the search process. The input is the problem definition, $\mathsf{P} = (\Sigma, s^0, g, \{o_i\})$ and the output is a sequence of actions for all actors in the composite domain that meets the composite goal, g while trying to optimize the individual objective functions, $\{o_i\}$ of all actor in the group. An actor always decides an action to perform that meets its individual objectives. However, the search process works in the composite space of multiple actors and chooses a trajectory of actions for all actors which meets the composite goal. The output of the search π, is the sequence of actions which meets the desired behavior for all actors in the composite space.

Even though actors that belong to one group do not simultaneously plan with other actor groups, the search always considers the global state space for all actors in the scenario while generating a valid action trajectory. Hence, the action trajectories generated for actors in one group can be overlayed on top of the trajectories of other actors whose goals and objectives are independent of their actions to generate a complete simulation.

Composite Actions. A composite action is a set of overlapping actions, $\{a_i\}$ chosen by *all* actors in the composite domain at a par-

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
OPEN = $\{s^0\}$
CLOSED = $\{\phi\}$
while *OPEN* $\neq \{\phi\}$ **do**
  $s = \arg_s \min(f(s))$ where $s \in$ OPEN
  goalReached = **isGoalConditionSatisfied**(g, s)
  if *goalReached* **then**
    π = **generatePath**(g,CLOSED)
    return π
  end
  OPEN = OPEN $- \{s\}$
  CLOSED = CLOSED $\cup \{s\}$
  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')
    OPEN = OPEN $\cup \{s'\}$
  end
end

Algorithm 1: Heuristic Search Algorithm

ticular point of time. A composite action is said to be *invalid* if there exists an actor who is yet to choose an action. The **generateCompositeActions**(s) function generates the set of all possible composite actions at the current state s, which is the cross product of all actions possible at s, for each actor in the composite domain. An action for a particular actor is said to be possible, if 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. Once all actors have chosen an action that optimizes their individual objective functions, the **simulate** routine simulates the overlapping actions of all actors in the group to progress by one time step.

Simulation of Composite Actions. A composite action 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. Actions for an actor are modeled as functions of time and take a finite amount of time to execute. Different actions can thus take differing amounts of time. 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. For a valid transition, the actions of all actors are simulated for one time step in a random order. After the simulation of a composite action, an actor may find itself in one of the following states:

- **Success.** The action is successfully completed and the actor must chose a new action in the next time step.
- **Executing.** The action is partially executed at the end of the simulation routine for that time step.
- **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. Given a composite action, $\{a_i\}$ where a_i is the action chosen by actor i in the composite domain, its cost is calculated as follows:

$$\textbf{costFunction}(s, A, t, t+1) = \sum_i o_i(\{c_j\}) \quad (1)$$

where $\{c_j\}$ are the values of the cost metrics for simulating action, a_i for agent i at state, s, from time, t to $t+1$, and o_i is the objective function of agent 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 heurisitics [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.

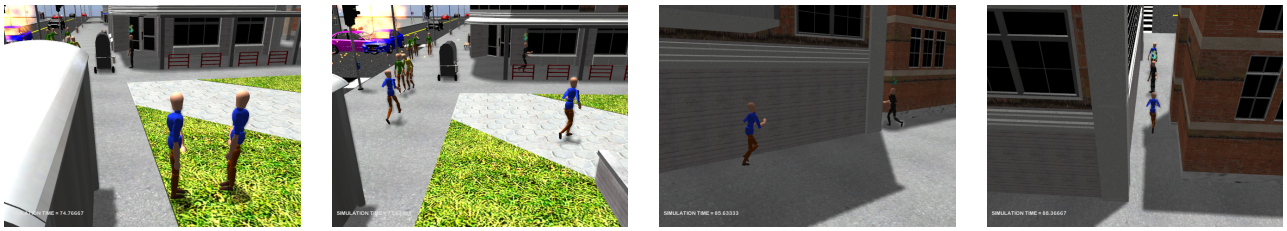
5 City Simulation

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 a thief and a hot dog salesman, whose behaviors are automatically generated using our framework. Section 5.1 details the task of specifying the state and action space of three generic actors (a pedestrian, a vehicle and a traffic signal) and illustrate the ease with which actors can be specialized to add variety and purpose to the simulation. Section 5.2 discusses a variety of results that can be achieved by changing the actor specializations or introducing constraints on the story line. The scripts used for defining actors, specializations and behaviors are outlined in Table 1.

5.1 Actor Specification and Specialization

Three generic actors, each having their own state and action space are defined as follows:

- **Pedestrians:** Pedestrians have a position, orientation, speed of movement and actor radius to model their movement in the environment. The action, `Move` (Script 1(a)) is defined to kinematically translate the actor and has an associated distance and energy cost. In addition, pedestrians have the following abstract metrics: hunger, safety, and amount of money. Actions such as `Eat` are modeled to affect particular abstract metrics and have an associated cost metric. Pedestrians are constrained to move on the sidewalks, use the crossings when the signal turns green (Script 1(c)) and are given high-level behaviors (e.g. satisfy their hunger by getting a hot dog, meet a friend at the park).
- **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` (Script 1(d)) determines the state of the traffic signal based upon the current simulation time. The pedestrian and the vehicles query the signal



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

Figure 2: *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 (Script 1(l)) to surround the thief in the alley and manage to catch him.*

state in order to follow the traffic signals.

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 perfoming their jobs. Script 1 (j) 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 (Script 1(h)).
- **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 (Script 1(e)).
- **Daring Actor:** A daring actor is authored by lowering the cost of actions that may place him in danger (Script 1(f)). Here, danger is a user-defined metric that is associated with each actor in the scenario.
- **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 (Script 1(n)). He has an action *Steal* (Script 1(n)) which allows him to steal money from the hot dog stand. In addition, a cost modifier *RiskCostModifier* (Script 1 (o)) 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. Scripts 1 (q) and (r) define the effect and cost modifiers for a reckless vehicle.

5.2 Results

Populating the city block with pedestrians and vehicles injects life into the simulation. 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* (Script 1(p)), 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. First, we introduce a partner to the vendor, and balance the objective of the vendors to minimize safety cost as well as the cost of being robbed as individuals. 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. Next, we change the objective of the vendor to minimize the cost of both being robbed (common objective) and place a constraint that the thief must be caught. As a result, the two vendors cooperate to corner the thief in an alley (Figures 2(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 by the author: (1) 15 cars and 4 fire-fighters, (2) two vendors and one thief, (3) old man and son and, (4) 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 searches which can be achieved at interactive frame rates. Note that the choice of composite domains is arbitrary and in the hands of the author. For example, authoring interactions between the old man and firemen would necessitate the old man and firemen belonging to the same planning domain. The plans for each of these domains is then overlayed to form the complete solution. The performance results are provided in Figure 3. 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.

Characters are animated by transitioning between walk, run and stop animations based on the speed of movement. This results in artifacts with abrupt transitions from one animation state to another. Note that the animation of the virtual humans is tangential to this research which focuses on behavior authoring.

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

Figure 3: *Performance Results.*

6 Discussion

In this paper, we present a framework for generating complicated behaviors between multiple interacting actors in a scenario. Our specification interface allows authors to work at any level of abstraction: Domain specialists define the state and action space of actors while end-users can re-use existing libraries of actor definitions and specialize them for the purpose of their own simulation. Actors with common or contradicting goals are grouped together into a set of composite domains where the behavior of an actor in one domain is independent of all actors in other domains. For each of these domains, a heuristic search technique plans a trajectory of actions for all actors to meet the desired behavior. This facilitates the generation of complex multi-actor interactions without the need of a centralized planner or the explicit modeling of communication between actors. 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 author your own simulations.

Limitations. Our framework currently does not work in real-time, making it unsuitable for applications such as games. Possible extensions for the purpose of real-time behavior generation include restricting the horizon of the search and using an anytime planner which returns a valid solution at any point of time and incrementally improves the solution with each iteration. Our heuristic search method is currently serialized and not optimized for parallelization. Also, the different plans for each composite domain need to be serialized since choosing an action for each actor queries the global state space of all actors in the scenario. We are currently investigating parallel search algorithms [Grama and Kumar 1993] in an effort to achieve real-time performance.

References

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

BINDIGANAVALE, R., SCHULER, W., ALLBECK, J. M., BADLER, N. I., JOSHI, A. K., AND PALMER, M. 2000. Dynamically altering agent behaviors using natural language instructions. In *In Autonomous Agents*, ACM Press, 293–300.

BLUM, A. L., AND FURST, M. L. 1995. Fast planning through planning graph analysis. *ARTIFICIAL INTELLIGENCE 90*, 1, 1636–1642.

BLUMBERG, B. M. 1997. *Old tricks, new dogs: ethology and interactive creatures*. PhD thesis. Supervisor-Maes, Pattie.

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.

DECUGIS, V., AND FERBER, J. 1998. Action selection in an autonomous agent with a hierarchical distributed reactive planning architecture. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, ACM, New York, NY, USA, 354–361.

DURUPINAR, F., ALLBECK, J., PELECHANO, N., AND BADLER, N. 2008. Creating crowd variation with the ocean personality model. In *Proceedings of AAMAS'08*, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1217–1220.

EROL, K., HENDLER, J., AND NAU, D. S. 1994. Htn planning: Complexity and expressivity. In *Proceedings of AAAI*, AAAI Press, 1123–1128.

FIKES, R. E., AND NILSSON, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2*, 3-4, 189 – 208.

FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *Proceedings of ACM SIGGRAPH*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 29–38.

GRAMA, A. Y., AND KUMAR, V. 1993. A survey of parallel search algorithms for discrete optimization problems. *ORSA JOURNAL ON COMPUTING 7*.

HORLING, B., LESSER, V., VINCENT, R., WAGNER, T., RAJA, A., ZHANG, S., DECKER, K., AND GARVEY, A., 1999. The TAEMS White Paper.

LAU, M., AND KUFFNER, J. J. 2005. Behavior planning for character animation. In *Proceedings of the ACM SIGGRAPH/EG Symposium on Computer Animation*, ACM, New York, NY, USA, 271–280.

LOYALL, A. B., REILLY, W. S. N., BATES, J., AND WEYHRAUCH, P. 2004. System for authoring highly interactive, personality-rich interactive characters. In *Proceedings of the ACM SIGGRAPH/EG Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, 59–68.

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

MASSIVE SOFTWARE INC., 2010. Massive: Simulating life. www.massivesofware.com.

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.

SHAO, W., AND TERZOPOULOS, D. 2005. Autonomous pedestrians. In *Proceedings of the ACM SIGGRAPH/EG Symposium on Computer Animation*, ACM, New York, NY, USA, 19–28.

SUNG, M., GLEICHER, M., AND CHENNEY, S. 2004. Scalable behaviors for crowd simulation. *Computer Graphics Forum 23*, 3, 519–528.

THIEBAUX, M., MARSELLA, S., MARSHALL, A. N., AND KALLMANN, M. 2008. Smartbody: behavior realization for embodied conversational agents. In *Proceedings of AAMAS'08*, 151–158.

VILHJÁLMSSON, H., CANTELMO, N., CASSELL, J., E. CHAFAI, N., KIPP, M., KOPP, S., MANCINI, M., MARSELLA, S., MARSHALL, A. N., PELACHAUD, C., RUTTKAY, Z., THÓRISSON, K. R., WELBERGEN, H., AND WERF, R. J. 2007. The behavior markup language: Recent developments and challenges. In *Proceedings of IVA '07*, Springer-Verlag, Berlin, Heidelberg, 99–111.

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*, Eurographics Association, Aire-la-Ville, Switzerland, 119–128.

YU, Q. 2007. *A decision network framework for the behavioral animation of virtual humans*. PhD thesis, Toronto, Ont., Canada.

```
Action Move(Velocity : v, TStep: dt) {
 Precondition:
  CheckCollisions(self.position + vdt)
    == false;
 Effect:
  self.position = self.position + vdt;
 Cost Effect:
  self.energyCost = 1/2(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 PedSignalC {
 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)

```
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;
}
```

(d)

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

(e)

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

(f)

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

(g)

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

(h)

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

(i)

```
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);
}
```

(j)

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

(k)

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

(l)

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

(m)

```
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;
}
```

(n)

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

(o)

```
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;
}
```

(p)

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

(q)

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

(r)

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