Demonstration-Based Training of Non-Player Character Tactical Behaviors

John Drake
University of Pennsylvania
drake@seas.upenn.edu

Alla Safonova
University of Pennsylvania
alla@seas.upenn.edu

Maxim Likhachev
Carnegie Mellon University
maxim@cs.cmu.edu

Abstract

State of the art methods for generating non-player character (NPC) tactical behaviors typically depend on hard-coding actions or minimizing a given objective function. In many games however, it is hard to foresee how the NPC should behave to appear intelligent or to accommodate human player preferences for NPC tactics. In this paper we consider an alternative approach, by training NPC tactical behavior via demonstrations. We propose a heuristic search-based planning method based on previously-developed Experience Graphs, which facilitates the use of behavior demonstration data to plan goal-oriented NPC behavior. Our method provides a principled solution to the problem which tolerates some amount of differences in between the training demonstration and the actual problem and yet still grants guarantees on the quality of the solution output.

1 Introduction

Planning NPC tactical behavior is a complex problem, with current solutions typically depending on hard-coded components or minimization of some given objective function (Millington and Funge 2009). Moreover, there do not exist ways to train NPC tactical behavior by demonstration. Heuristic graph search techniques such as A* search (Hart, Nilsson, and Raphael 1968) can be used for planning NPC tactics, however they can be computationally expensive and do not by default incorporate player preference on how NPCs should behave. The previously-developed Experience Graph (E-graph) (Phillips et al. 2012) method allows the use of experience or demonstration data in a graph search. However, it is formulated in a way that requires demonstrations to be part of the graph used by the search. As we explain in the paper (See section 4 on Technique), this breaks down in the context of games.

In this paper we propose an alternative Training Graph (T-graph) heuristic formulation. Our method can utilize demonstration data that does not lie directly on the search graph and is not accessible from the search graph. It tolerates changes to the environment and initial conditions of the problem. Our method also permits the usage of demonstration data with low sample density or downsampled demonstration data. Our method encourages more complete use of demonstration data than the E-graph method does. Finally, our heuristic can be combined with other deterministic AI behavior control methods. These features together enable demonstration-based behavior planning to be used under various scenarios and configurations. We demonstrate our method on several test scenarios, including one in the video game *The Elder Scrolls V: Skyrim* (Skyrim) .

2 Previous Work

NPC behavior planning is typically accomplished via hard-coded methods (such as behavior trees (Isla 2005) or finite state machines (Coman and Muñoz-Avila 2013)) or by minimizing some given objective function, as in planning approaches (such as (Macindoe, Kaelbling, and Lozano-Perez 2012)).

Some methods have incorporated demonstration data into planning. In robotics, inverse optimal control (Finn, Levine, and Abbeel 2016) has been used to learn a cost function from demonstration for a planning problem. Our method uses a known cost function and instead uses demonstrations to bias the search. In other words, we use training data directly to guide the search rather than learning a cost function from it. A 2009 research survey (Argall et al. 2009) collects some other related methods for robot control.

Our technique is based on another method from robotics, Experience Graphs (E-graphs) (Phillips et al. 2013), which in turn is an extension of A* heuristic graph search (Hart, Nilsson, and Raphael 1968). A* uses a heuristic estimate of the transition cost between nodes to constrain the amount of the search graph which needs to be examined before finding a solution to the problem. The E-graph algorithm extends A* search to be able to use the results of previous searches and/or demonstration data. It does it in a way that provides provable bounds on the sub-optimality of the solution. The standard formulation of E-graph search, however, assumes that the E-graph is reachable from the search graph and traversable along its length. Our method uses demonstration data just like the E-graph method, but does not require that the demonstration graph is reachable from the search graph. Our method also encourages more complete usage of demonstration data than the E-graph method does, which is important for tactical behaviors.

A recent work on tactical behavior planning by N. Sturtevant (Sturtevant 2013) proposed a method which takes into

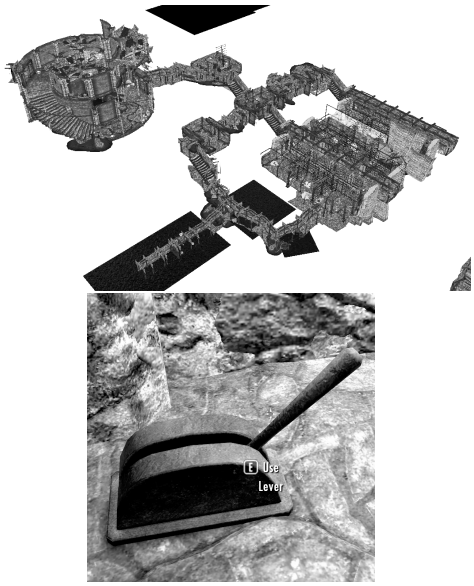

Figure 1: Example dungeon and lever as goal.

account relationships between NPCs while planning paths. The proposed method focuses on path planning alone and decomposes NPC interaction into parameters which, in sum with distance traveled, make up the cost function of the search. Our method plans for behavior actions beyond navigation and provides an avenue for player preference to affect tactical planning. Our method could be combined with cost formulations such as this.

3 Motivating Example

We consider here the following common scenario in a game. The player encounters a friendly NPC who winds up joining the player on a mission. The mission is short and has a clearly specified goal, e.g. to pull a lever at the end of a small dungeon. The player and his companion begin at the door to the dungeon. There may be forks in the passageways between the door and the lever, creating different route options. There may be different actions to take before reaching the lever, for example to kill particular enemies along the way or to leave them alone. The player and NPC can also choose to move stealthily or run through quickly.

The player begins to navigate the dungeon to get to the lever at the end (Figure 1). His companion NPC acts according to its programming. It often happens that this is insufficient. For example, the NPC may only know to charge ahead despite the fact that the level demands a stealthy approach. There may be a very strong enemy NPC in one part of the level which could be bypassed via another route. There may be a navigational trap, such as a spear pit. It is not possible to foresee every such situation while designing the AI code for the NPC. The player may wish to instruct the NPC how to behave, and indeed many games incorporate commands which can be given to friendly NPCs, but it is often not possible to issue compound commands for complex tactics and not possible to foresee command schemes to handle every scenario. The game developer might also wish to author NPC behavior by demonstration, perhaps to augment other behavior control code for complex scenarios that leave the NPC incapable of solving them.

We have used Skyrim as a testing ground for our work. Since we do not have access to full models of how the game's existing NPCs behave, we have simplified the game's wolf and bear AI models so that they can be better modeled in our system. We depopulated game areas and repopulated them with our custom wolf and bear agents for testing. We extracted game navmesh data, including Skyrim's unidirectional *drop down* edges, from these game areas for use in our system's navigation routines. We have used the game's *Creation Kit* editing software and modified the *Skyrim Script Extender* (SKSE, http://skse.silverlock.org/) for integration with our behavior planner and demonstration recorder. Demonstration data is recorded by sampling all relevant state information of the game world (health values, agent positions, etc.) at a regular interval and saving this information to a log file. The player presses a key (handled with SKSE and *Papyrus* scripting) to start and stop each log file. These logs are then used by our algorithm as demonstration data.

4 Technique

Graph Search

NPC tactic planning can be accomplished as a graph search problem. Our search graph is composed of nodes representing search states and edges representing possible transitions between search states. Graph search states are composed of world state information and the state of the NPC being planned for. World state information is composed of the state of every NPC (including enemies), and other miscellaneous world parameters. NPC state information encodes values for position, health, stamina, navigation information, gait (sneaking, walking, running), etc. for the NPC. Graph edges have an associated cost defined as the amount of time it takes to complete that state transition. A transition that results in companion NPC death has infinite cost.

We use heuristic graph search to find a feasible path from the start state to the goal. We tested our planner on different partially-specified goal states, including one that is satisfied when the NPC is within two meters of a destination location and one that is satisfied when a target enemy NPC is dead. Heuristic graph search algorithms use a *heuristic* which focuses search efforts to dramatically speed up the search process. A heuristic estimates the distance between two states. A graph search heuristic can estimate the distance between a state and the goal, even if the goal is partially-specified.

A graph heuristic used throughout this paper, $h^S(a, b)$ between search states a and b (visualized in a test environment in Figure 2), is defined as Euclidean distance between the planned NPC positions in a and b divided by the maximum possible travel speed for the NPC. It therefore estimates the time-to-goal, which makes it consistent with the cost function.

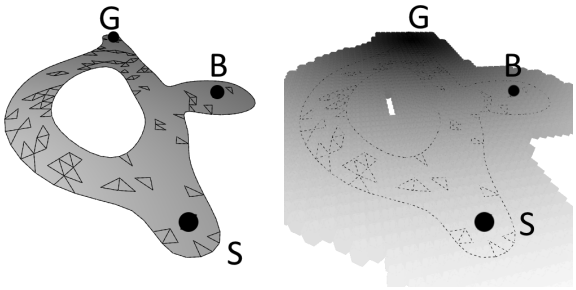

Figure 2: Left: Test environment with forked navmesh, start position S, goal G, and killer bear B hiding in den.
Right: Standard graph heuristic h^S, based on Euclidean distance. The values decrease (darken) near the goal in a smooth manner.

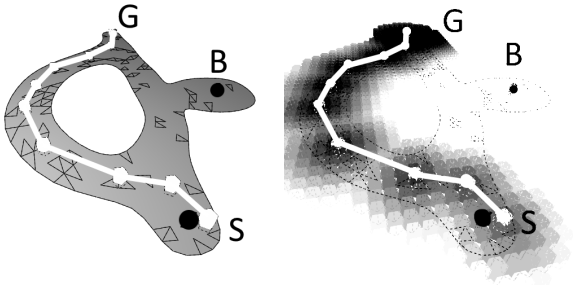

Figure 3: Left: Test environment with training path (white lines) visualized.
Right: E-graph heuristic used without E-graph being connected to the search graph. The white lines represent the E-graph. Heuristic values decrease both toward the E-graph nodes and overall toward the goal, but each node forms a local minima, which significantly delays search progress.

E-graph Heuristic

First we explain Experience Graphs, which our work builds on. The E-graph algorithm introduces a method to guide a graph search to use trajectory data encoded in an E-graph to help the search avoid local minima. The E-graph can be created from the solutions of prior planning problems in the same domain, or it can be created by demonstration (Phillips et al. 2013). If necessary, new edges are added to the search graph to connect it to the E-graph. The E-graph heuristic h^E is used to bias the search toward reuse of the E-graph edges while searching for a solution.

E-graph E is a directed graph with nodes E^N encoding search state information. Successor function $succ(a) = b$ records valid transitions between states a and b as observed in the experience with an associated cost function $c^E(a, b)$ which records the observed costs of these transitions. The standard experience graph heuristic $h^E(a, b)$ between nodes a, b on the search graph can be computed in a general way as follows. Note that node b is typically the goal state in a search problem, and then you can understand $h^E(a)$ as shorthand for $h^E(a, goal)$. Let h^G be some heuristic on the search graph, such as h^S described above.

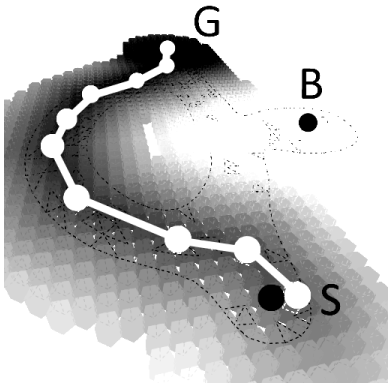

Figure 4: T-graph heuristic values decrease both toward the T-graph and along it toward the goal in a smooth manner. White lines represent the T-graph.

1. The E-graph is augmented with virtual edges from every E-graph node s to every other E-graph node s' at their known E-graph cost $c^E(s, s')$ if the transition already existed in the E-graph (if $s' \in succ(s)$) or at cost $\epsilon^E h^G(s, s')$ otherwise.
2. The E-graph is further augmented with virtual edges from every E-graph node s to b at cost $\epsilon^E h^G(s, b)$.
3. Dijkstra's algorithm is run on the augmented graph from b as the source. The Dijkstra output distances are used as $h^E(s, b)$ for all nodes $s \in E^N$.
4. The smallest of the direct path $\epsilon^E h^G(a, b)$ and every sum $\epsilon^E h^G(a, s) + h^E(s, b)$ among all $s \in E^N$ is selected as $h^E(a, b)$.

This standard implementation of the E-graph heuristic guides the graph search in a way that assumes the E-graph is traversable and directly reachable from the search graph. The search is guided toward E-graph nodes, so if the E-graph density is sparse compared to the search graph's discretization, the search may be guided backwards (away from the goal and backward along the E-graph) to reach an E-graph node before making progress toward the goal (these are the local minima visible in Figure 3). Also, due to the use of Dijkstra's algorithm in the existing E-graph heuristic formulation, the search may leave the E-graph long before reaching its end. This can prevent the E-graph demonstration data from leading the solution path first toward the goal and then away from it, though that kind of behavior is important to some tactical maneuvers.

T-graph Heuristic

We modify the computation of the heuristic so that it encourages the search to expand states that *follow* the demonstration without requiring the state to be *exactly* on the demonstration. In other words, we need to make the heuristic decrease between the start and the demonstration graph, then decrease along the length of the demonstration graph and, finally also decrease between the end of the demonstration graph and the goal. Training graph (T-graph) T is represented the same way as an E-graph. Let T^N indicate the

set of nodes making up T. Let $succ(s)$ return the set of successor neighbors of T-graph node s. Let $pred(s)$ return the set of T-graph nodes s' such that $succ(s') = s$. For all $s \in T^N$ and $s' \in succ(s)$, let $c^T(s, s')$ describe the cost of transitioning from s to s'. Let $T^N_{term} \subset T^N$ represent the set of terminal nodes s in T^N such that $succ(s) = \emptyset$. Let $T^N_{init} \subset T^N$ represent the set of initial nodes s in T^N such that $pred(s) = \emptyset$. A *shortest path* graph heuristic $h^P(a, b)$ is defined as the shortest path length from a to b on the search graph divided by the maximum possible travel speed.

We compute our training heuristic $h^T(a, b)$ (visualized for a test environment in Figure 4) as follows:

1. Let ϵ^T be a heuristic inflation factor like ϵ^E in the E-graph method.
2. For each terminal node $s \in T^N_{term}$, assign $h^T(s, b) = h^P(s, b)$.
3. Working backwards from each terminal node s, where $s' \in pred(s)$, assign $h^T(s', b) = h^T(s, b) + c^T(s', s)$. Repeat this until every T-graph node s' is assigned a $h^T(s', b)$ value. This step replaces the Dijkstra calculation in the E-graph method.
4. An estimation $h^T_{est}(a, b, s, s')$ of the heuristic between a and b is computed for every pair of T-graph nodes $s \in T^N$ and $s' \in succ(s)$ by observing that $h^S(s, s')$, $h^S(s, a)$, and $h^S(a, s')$ are available to compute a notion of how far a is located between s and s'. This is used to choose a value for h^T_{est} between $h^T(s, b)$ and $h^T(s', b)$ (or larger). In our method, we decided to compute this as a projection π and rejection ρ from an imaginary line between s and s' as follows:
   (a) Let $\pi = \frac{h^S(a,s')^2 - h^S(s,a)^2 + h^S(s,s')^2}{2h^S(s,s')^2}$
   (b) Let $\alpha = \pi / h^S(s, s')$
   (c) Let $\rho = \sqrt{h^S(s, a)^2 - (h^S(s, s') - \pi)^2}$
   (d) If $\alpha < 0$, let $h^T_{est}(a, b, s, s') = \epsilon^T h^S(a, s') + h^T(s', b)$
   (e) If $\alpha > 1$, let $h^T_{est}(a, b, s, s') = \epsilon^T h^S(s, a) + h^T(s, b)$
   (f) If $0 \leq \alpha \leq 1$, let $h^T_{est}(a, b, s, s') = \epsilon^T \rho + \alpha c^T(s, s') + h^T(s', b)$
5. An estimation $h^T_{est}(a, b, s, b)$ of the heuristic between a and b is computed for every pair of terminal node $s \in T^N_{term}$ and node b in the same way as step 4. This allows the search to be drawn in a focused way from the training data toward the goal. In our work, we use h^P here instead of h^S as it tended to make this last section of the heuristic computation conform to the navmesh better and therefore produced better results.
6. An estimation $h^T_{est}(a, b, a, s)$ of the heuristic between a and b is computed for every pair of node a and initial node $s \in T^N_{init}$ in the same way as step 5. This allows the search to be drawn in a focused way from the start node toward the training data.
7. The smallest of the direct path $\epsilon^T h^S(a, b)$ and every $h^T_{est}(a, b, s, s')$ estimate among all $s \in T^N \cup \{a\}$, $s' \in T^N \cup \{b\}$ is selected as $h^T(a, b)$.

On state expansions, for every state s and successor state s' (with transition cost $t = c^T(s, s')$), a deterministic world simulation function $w' = sim(s_w, t)$ is used to forward-simulate world state s_w over a duration of t, outputting updated world state w'. Successor state s' is then assigned w' as its world state information. The simulation function evaluates models of behavior for all dynamic components of the world. In this function, behavior models for all NPCs are evaluated. The NPC being planned may be partially modeled by sim, for example if despite the planner, parts of this NPC's behavior are to be controlled by another technique. In our implementation, planned parameters include desired location (as a position) and willingness to fight (as a Boolean value) and then deterministic scripted behavior handles the details of navigation and combat in sim. As long as accurate NPC behavior models are available to sim, sim accurately predicts how all NPCs behave during the graph search. Other dynamic components of the world relevant to the planning problem can be modeled here, such as doors which automatically open or close, physics on moving objects, or consideration for how some kinds of attack damage (e.g. *splash damage* which could affect multiple targets) should be resolved.

Theoretical Properties

Some general properties of ϵ-admissible heuristic graph search are preserved in our method. The search is *complete*; if a solution is possible on the search graph, the search will return a solution. We do not modify the search graph in any way and we use Weighted A* graph search, which is a complete planner, so our search is also complete.

Our heuristic is ϵ^T*-admissible*. An admissible heuristic between nodes a and b never overestimates the actual transition cost between a and b. An ϵ-admissible heuristic never overestimates the transition cost by more than a factor of ϵ. Our heuristic $h^T(a, b)$ is computed as the minimum of several options, one of which is the direct connection (step 7 above) between nodes a and b, computed as $\epsilon^T h^S(a, b)$, so $h^t(a, b)$ must always be less than or equal to $\epsilon^T h^S(a, b)$. $h^S(a, b)$ is an admissible heuristic, so $\epsilon^T h^S(a, b)$ is an ϵ^T-admissible heuristic, and since $h^T(a, b) \leq \epsilon^T h^S(a, b)$, it must also be an ϵ^T-admissible heuristic.

A consistent heuristic obeys the triangle inequality as specified here: $h(a, c) \leq c(a, b) + h(b, c)$. If you compute the heuristic between nodes a and c, this value is less than or equal to the cost to transition from a and to successor node $b \in succ(a)$ plus the heuristic between b and c. An ϵ-consistent heuristic obeys this inequality: $h(a, c) \leq \epsilon c(a, b) + h(b, c)$. Consistency (or ϵ-consistency) of our h^T heuristic depends on the formation of h^T_{est} in steps 4 and 5 above and this remains to be proven for our implementation. However, we found that it functioned well and path costs were always well within sub-optimality bound $\epsilon^T \epsilon$ times the cost of the optimal path.

Analysis

Our h^T heuristic calculation guides the graph search along training demonstration data paths, even where they might

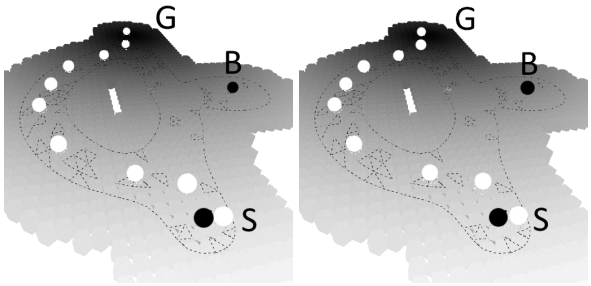

Figure 5: Left: E-graph heuristic with $\epsilon^E = 1$.
Right: T-graph heuristic with $\epsilon^T = 1$. Note that these both degenerate to the simple graph heuristic h^s seen in Figure 2.

move away from the goal. Like with the E-graph heuristic computation, if $\epsilon^T = 1$, the heuristic degenerates to the graph heuristic h^S (see Figure 5). Higher values of ϵ^T encourage the search to use the demonstration data more closely. Our formulation tolerates situations where the demonstration data does not lie directly on the search graph.

Since our method tolerates sparse demonstration data samples, it permits demonstration data to be sampled below its full resolution for performance improvements at the expense of training precision, if desired.

If the sim function has no effect on the heuristic functions, then many parts of the h^T computation can be precomputed and accessed from memory at runtime. For example, if the goal of the problem is to kill an enemy NPC and the sim function updates that NPC's position over time, that can alter the heuristic during the search and so parts of h^T cannot be precomputed.

5 Results

All of our results were generated on a machine with a 2.8GHz (up to 3.46 GHz) Intel i7-860 CPU, 12GB of dual-channel PC3-10600 RAM, and Windows 7 64-bit OS. Our code was compiled by MS Visual Studio 2013 and all runs on a single thread.

For most experiments, we compared our method to standard A* search in a testbed environment separate from our target game Skyrim.

We show planner execution time performance results in Figure 6 for combined (ϵ and ϵ^T) epsilon values of 1, 10, and 100 for the *Bear Bypass* scenario shown in Figures 2 and 3. In this scenario, the start and goal locations are at opposite ends of a straight passageway. Midway along the passage is a bear den with a very dangerous killer bear inside. An alternative route bypasses the bear den. A* search with a Euclidean heuristic expands directly toward the bear den first, encountering a large local minima. T-graph search expands around the alternative route as demonstrated, bypassing the bear den and local minima entirely. We are primarily concerned with examining T-graph output path shape so these performance results are included only to illustrate that computation times for the T-graph plans tended to be similar to those for A*.

We also tested our method with a scenario from Skyrim.

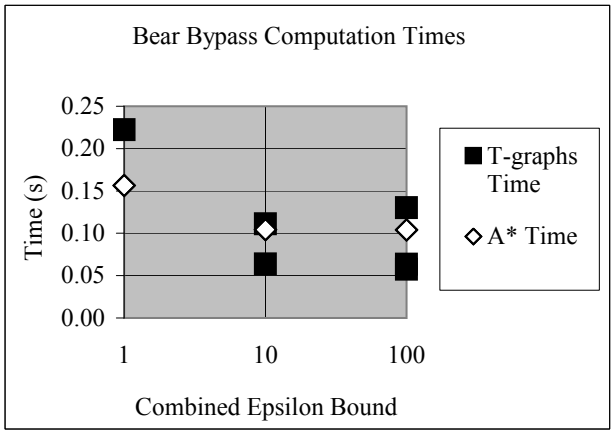

Figure 6: *Bear Bypass* planning time results for several combined epsilon values.

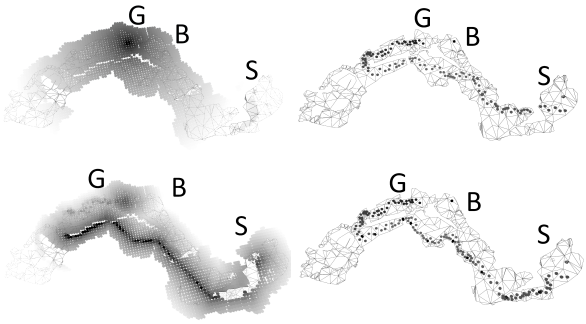

Figure 7: Tests on navmesh taken from the *Greywater Grotto* cave level in Skyrim.
Top: A* heuristic and path plan ($\epsilon = 200$) to go from the start position at the right to the goal at the top, behind the bear, for a sneak attack.
Bottom: T-graph heuristic and path plan for the same scenario with the same combined $\epsilon^T \epsilon = 200$ ($\epsilon = 2, \epsilon^T = 100$). Path follows a training demonstration and was planned in 18% fewer expansions than A*.

See Figure 7 for a visualization of the graph heuristics and planned paths from this test. To record the demonstration data for this example, we run Skyrim through SKSE with our mod file enabled and went to the *Greywater Grotto* location. A keyboard button begins demonstration recording mode and then the player walks through the level demonstrating how he wants the NPC to behave, including choosing when to engage in combat and when to move stealthily. Another keyboard button resets the level and allows the NPC to begin behaving according to the planned behavior. This time, the player can go through the level with the NPC, knowing it will behave according to the demonstration. The planner could be called again at intervals to alter the plan based on changing circumstances in the game.

The T-graph can be used to guide the search in arbitrary ways according to player (or developer) preference. Figure

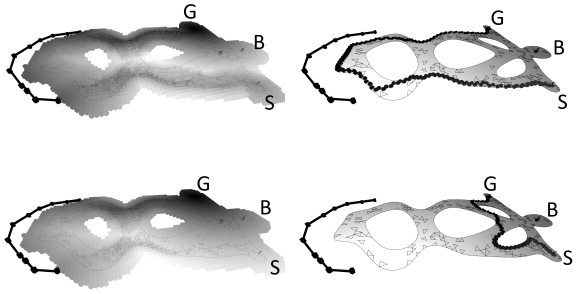

Figure 8: Top Left: T-graph heuristic for $\epsilon^T = 10$. The black lines mark the T-graph nodes.
Top Right: T-graph path for $\epsilon^T = 10$ follows the demonstration even though it is far away.
Bottom Left: T-graph heuristic for $\epsilon^T = 5$.
Bottom Right: T-graph path for $\epsilon^T = 5$ no longer follows the demonstration because at this ϵ^T value, the direct path heuristic dominates in the region of the search graph near the start and goal.

8 shows a T-graph that both strays from the search graph and begins and ends far from the start and goal of the search. The demonstration path successfully biases a search at $\epsilon^T = 10$, but then fails to bias a search at $\epsilon^T = 5$ because of its distance from the start and goal. ϵ^T controls how much influence training data has on the search and so irrelevant demonstration can be filtered out by lowering ϵ^T. If influence is low enough, the graph heuristic h^S dominates and search progresses as in plain A*.

The T-graph method also operates gracefully when multiple demonstrations are provided for the search, even if they overlap or if irrelevant demonstration data is included. See Figure 9. The output path includes planned actions that describe more than the shape of the navigation path. The arrows point to places where the output path includes changes to the NPC's gait state (not just position). The demonstration path curves toward the bear in this area, so the algorithm tested paths which went through this area and found that they all ended with the bear killing the NPC. Thus, the planner determined that in order to safely navigate this section, the quieter *sneaking* gait would need to be used to elude the bear.

6 Conclusions & Future Work

Our T-graph algorithm provides a new way to train AI tactical behavior via demonstration. We use a heuristic graph search framework and our special T-graph heuristic h^T to compute AI behavior solutions which follow demonstration data and also seek to achieve the specified tactical goal. Our system tolerates changes to the starting configuration, environment, and goal specification. It also permits the use of sparse demonstration data which does not lie directly on the search graph and permits demonstration data to be downsampled. Through the sim function, other deterministic AI behavior controls can be coupled with our system.

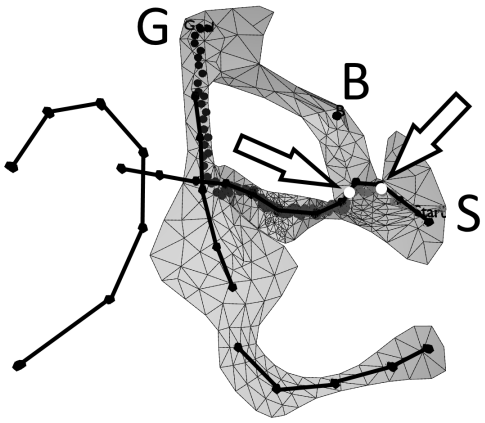

Figure 9: A scenario with a killer bear (B) to avoid and four demonstration traces (black lines) as input. A T-graph solution path starts at S, goes left to the where two T-graph traces intersect, then moves up toward the goal at G. The white dots marked by arrows indicate places where the planned path changes the NPC's gait in and out of a slower *sneaking* mode to avoid being detected by the bear while still following the demonstration data closely, demonstrating that the planner does more than simple path-planning.

Future work includes extending the system beyond tactical tasks into the realm of long-term strategy. Another important task for future work is further generalization of how experience data is treated so that it can be applicable in broader scenarios. One aspect of this is determining a graceful method for dealing with discrete state parameters; it is easy to compute a smooth heuristic for things like positions and distances, but not as easy for discrete parameters like door states or line-of-sight information.

7 Acknowledgments

This work was supported by NSF Grant IIS-1409549.

The authors would like to thank Pasan Pow Julsaksrisakul and Rikky Roy Koganti for their work developing the enemy NPC models and other miscellaneous components used in this project. The authors also thank Maximilian Kiefer for his work crafting several scenarios used in testing and in the figures of this paper.

References

Argall, B.; Chernova, S.; Veloso , M.; and Browning , B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 67:469–483.

Coman, A., and Muñoz-Avila, H. 2013. Automated generation of diverse npc-controlling fsms using nondeterministic planning techniques. In *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Finn, C.; Levine, S.; and Abbeel, P. 2016. Guided cost learning: Deep inverse optimal control via policy optimiza-

tion. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

Isla, D. 2005. Handling complexity in the halo 2 ai. GDC.

Macindoe, O.; Kaelbling, L. P.; and Lozano-Perez, T. 2012. Pomcop: Belief space planning for sidekicks in cooperative games. In *Proceedings, The Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Millington, I., and Funge, J. D. 2009. *Artificial Intelligence for Games 2nd Edition*. Morgan Kaufmann.

Phillips, M.; Cohen, B.; Chitta, S.; and Likhachev, M. 2012. E-graphs: Bootstrapping planning with experience graphs. In *Proceedings of Robotics: Science and Systems*.

Phillips, M.; Hwang, V.; Chitta, S.; and Likhachev, M. 2013. Learning to plan for constrained manipulation from demonstrations. In *Proceedings of Robotics: Science and Systems*.

Sturtevant, N. R. 2013. Incorporating human relationships into path planning. In *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.