Proceedings, The Thirteenth AAAI Conference on
Artificial Intelligence and Interactive Digital Entertainment (AIIDE-17)

Towards Adaptability of Demonstration-Based Training of NPC Behavior

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

Using demonstration to guide behavior generation for non-player characters (NPCs) is a challenging problem. Particularly, as new situations are encountered, demonstration records often do not closely correspond with the task at hand. Open-world games such as The Elder Scrolls V: Skyrim or Borderlands often reuse locations within the game world for multiple quests. In each new quest at each location, the particular configuration of game elements such as health packs, weapons, and enemies changes. In this paper, we present an approach that utilizes user demonstrations for generating NPC behaviors while accommodating such variations in the game configuration across quests.

Introduction

Training NPC behavior allows video game players to interact with friendly NPCs in tactics of their own design. Game designers can also use demonstration to author specific NPC behavior not easily captured by other approaches. A particular challenge in the context of NPC behavior training is that of adapting behavior as demonstrated to new environments and new problems. Open-world games often present the player with quests or missions that give the player familiar tasks in new locations and novel tasks in old locations. For example, the player may be asked to clear a dungeon of enemies in one quest, and then to go back to the same location in another quest to retrieve a special item. If the NPC is trained how to behave in the first quest, it is unclear how to apply that experience to the new scenario.

NPC behavior generation can be posed as a planning problem and accomplished by graph search. Graph search algorithms like A* will find an optimal solution, if a solution exists (Hart, Nilsson, and Raphael 1968). Other algorithms, like Weighted A* (Hart, Nilsson, and Raphael 1968) can find a solution in less time or with fewer computational resources, but at the cost of guaranteed solution optimality (Likhachev, Gordon, and Thrun 2003).

As discussed in (Drake, Safonova, and Likhachev 2016), the Experience Graph method can be used to train NPC behavior. The E-Graph method (Phillips et al. 2012) produces solutions which prefer to reuse segments of experience or demonstration paths. It does so by storing demonstrations as paths and then computing a heuristic function that biases the search towards the reuse of these paths within provided bounds on sub-optimality.

When the NPC behavior needed to accomplish a quest goal differs from behavior which was demonstrated in training, the use of the E-Graph heuristic can be *more* computationally costly than planning a solution with the plain A* algorithm. It can also produce strongly sub-optimal solution behaviors, since the search prefers to use the training data, which leads it astray. We propose a method of incorporating Multi-Heuristic A* Search (Aine et al. 2015) to adapt demonstration from one quest configuration to another.

Previous Work

Generating behavior for video game NPCs is typically accomplished by hand-crafted control schemes such as behavior trees (Isla 2005) or finite state machines (Coman and Muñoz-Avila 2013). Alternatively, a given objective function can be used to guide behavior generation, as in planning (Macindoe, Kaelbling, and Lozano-Perez 2012) approaches.

Graph search-based planning is commonly used in video game AI. The A* graph search algorithm has been used in myriad applications, including for spatial trajectory planning (e.g. navigation meshes (Snook 2000)) and AI decision-making. A* is a general-purpose graph-search technique which finds the optimal path between two nodes in a graph and does so by exploring the fewest number of nodes necessary to guarantee that the optimal solution has been found (Hart, Nilsson, and Raphael 1968). Variations on A* exist, such as Weighted A*, which relax the guarantee on solution quality (Likhachev, Gordon, and Thrun 2003). These variations tend to produce feasible solutions in less time than the standard A* algorithm can produce the optimal solution.

Demonstration can be used to guide NPC behavior generation. This can permit NPC behaviors to change (as new training demonstrations are provided) to suit new situations not anticipated at development time. In the field of robotics, inverse optimal control (Finn, Levine, and Abbeel 2016), (Ratliff, Silver, and Bagnell 2009) is used to learn a cost function from demonstration to bias the search. The E-Graph method instead recomputes the heuristic function to bias the search, leaving the cost function intact.

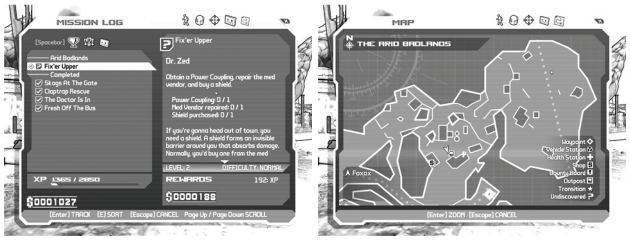
The E-Graph technique builds on A* by modifying the search heuristic. The E-Graph heuristic makes the search prefer to use paths from demonstration or previous solutions (experience) while generating a new solution path. A heuristic inflation parameter controls how much the search prefers to use experience paths, and it also affects the guarantee on solution optimality much like Weighted A*. Thus, the E-Graph method allows its user to make a tradeoff between how closely a new solution matches the experience and how far the new solution's cost might be from optimal. In cases where an experience path closely matches the optimal solution, the algorithm produces high-quality solution paths in little time.

For the purpose of NPC behavior generation from demonstration, the training-graph (T-Graph) heuristic (Drake, Safonova, and Likhachev 2016) was introduced to handle some of the particular qualities of the video game NPC behavior problem. While the E-Graph method either embeds experience within the search graph or augments the search graph with demonstration data (Phillips et al. 2013), the T-Graph heuristic permits the experience graph to lie off of the search graph so long as a heuristic function can be computed between the two. This is important because NPC behavior is often constrained (for example, the NPC may only be able to navigate on the navmesh), and the behavior of a player demonstrating a tactic is not. The T-Graph heuristic also tends to smooth out the heuristic gradient to remove local minima in the search space which slows down search progress. NPC behavior can be planned from demonstration in this way, but as with the E-Graph heuristic, when the spatial configuration of a problem is greatly different from the configuration of the demonstration record, large local minima are introduced and performance suffers. Addressing such issues is the focus of this paper.

Game Context

We consider open-world video games such as the games in the Elder Scrolls and Borderlands series. These kinds of games often send the player and supporting NPCs on quests (or "missions") through portions of the game environment we will call "dungeons" here. Each dungeon is accessible from the greater game world, but itself only represents a limited physical area. The game's storyline presents the player with quests, and so-called "side-quests" tangential to the main storyline are also available to the player. See Figure 1 and Figure 2 for examples of quest objectives and the corresponding dungeon areas from both the Borderlands and Skyrim games.

These quests often re-use particular dungeons at different points in the game. The player may be sent through a dungeon to retrieve a special item in one quest, but then be sent back to the same dungeon to activate a device in another quest. Parts of the dungeon may be exactly the same in a later quest as they were previously, but other important parts of the dungeon may have moved around. A key to open a door might be moved to a new location, there might be additional health pack pick-ups available, and enemies may have been randomly placed in new locations. See examples of environment reuse across quests in Figure 3 and Figure 4.


MISSION LOG

[Spacebar]

Arid Badlands
Fix'er Upper
Completed
☑ Skags At The Gate
☑ Claptrap Rescue
☑ The Doctor Is In
☑ Fresh Off The Bus

XP 1365 / 2850

$0001027

Fix'er Upper

Dr. Zed

Obtain a Power Coupling, repair the med vendor, and buy a shield.

Power Coupling: 0 / 1
Med Vendor repaired: 0 / 1
Shield purchased: 0 / 1

If you're gonna head out of town, you need a shield. A shield forms an invisible barrier around you that absorbs damage. Normally, you'd buy one from the med

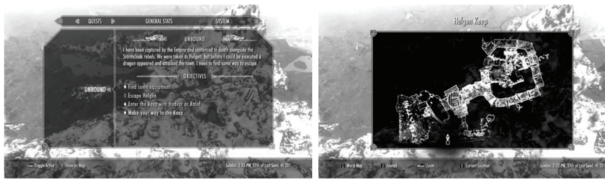
LEVEL: 2 DIFFICULTY: NORMAL

REWARDS 192: XP

$0000188

[Enter] TRACK [E] SORT [Escape] CANCEL Page Up / Page Down SCROLL

Figure 1: A Borderlands game mission with three objectives. The first objective's location (waypoint) is marked with a diamond marker at the bottom of the local map.


QUESTS GENERAL STATS SYSTEM

UNBOUND

UNBOUND

I have been captured by the Empire and sentenced to death alongside the Stormcloak rebels. We were taken to Helgen, but before I could be executed a dragon appeared and attacked the town. I need to find some way to escape.

OBJECTIVES

- Find some equipment
- Escape Helgen
- Enter the Keep with Hadvar or Ralof
- Make your way to the Keep

Toggle Active Show on Map

Sundas, 2:55 PM, 17th of Last Seed, 4E 201

Figure 2: A Skyrim game quest with four objectives. The local map of the associated game area shows a quest objective marked with a **V** marker at the bottom of the view.

NPC Behavior Training

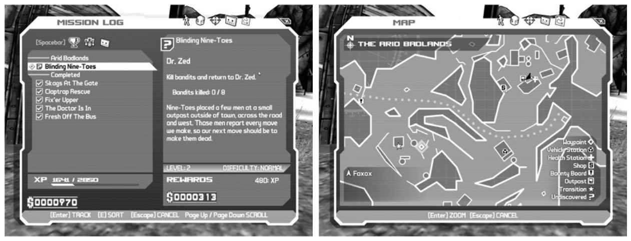
As discussed in (Drake, Safonova, and Likhachev 2016), it can be useful to train NPC behavior in these kinds of games, so that a companion NPC can more effectively assist the player, or so that the game developer can augment the NPC's ordinary AI capabilities with trained behavior for special circumstances. Training can be done by allowing the player (or developer) to record a trace of gameplay in a dungeon environment and then feeding this trace into a demonstration-based planning method like the E-Graph planner.

Since these kinds of heuristic graph search algorithms are *complete*, they will find a solution if any solution exists, so eventually they will find a behavior path for the NPC. Unfortunately, when the player and NPC encounter a new quest for a dungeon, the dungeon configuration may change so much that the training trace misleads the search into large local minima, causing a large computational delay before a solution path is found.

Demonstration-Based Training Via Graph Search

Graph Search

To use graph search for NPC behavior generation, the game's configuration space is discretized as a graph. Nodes on the graph describe the game state at a moment in time, including all positions of enemies and items, NPC behavior state, etc. Edges between nodes represent possible transitions from one game state a to another game state b and each edge from a to b has an associated cost $cost(a, b)$. For navigation planning, the cost may be in units of distance (the distance travelled on an edge), but in our case we use units of time, which captures a notion of the expense of both navigation actions and other kinds of actions such as attacks.


MISSION LOG

[Spacebar]

Arid Badlands
Blinding Nine-Toes
Completed
☑ Skags At The Gate
☑ Claptrap Rescue
☑ Fix'er Upper
☑ The Doctor Is In
☑ Fresh Off The Bus

XP 1641 / 2850
$0000970

Blinding Nine-Toes

Dr. Zed

Kill bandits and return to Dr. Zed.

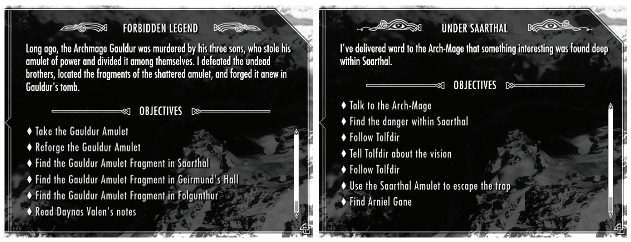
Bandits killed: 0 / 8

Nine-Toes placed a few men at a small outpost outside of town, across the road and west. Those men report every move we make, so our next move should be to make them dead.

LEVEL: 2 DIFFICULTY: NORMAL
REWARDS 480: XP
$0000313

[Enter] TRACK [E] SORT [Escape] CANCEL Page Up / Page Down SCROLL

Figure 3: A Borderlands mission sharing the same environment with Figure 1. Note that the objective has moved.


FORBIDDEN LEGEND

Long ago, the Archmage Gauldur was murdered by his three sons, who stole his amulet of power and divided it among themselves. I defeated the undead brothers, located the fragments of the shattered amulet, and forged it anew in Gauldur's tomb.

OBJECTIVES

- Take the Gauldur Amulet
- Reforge the Gauldur Amulet
- Find the Gauldur Amulet Fragment in Saarthal
- Find the Gauldur Amulet Fragment in Geirmund's Hall
- Find the Gauldur Amulet Fragment in Folgunthur
- Read Daynas Valen's notes

UNDER SAARTHAL

I've delivered word to the Arch-Mage that something interesting was found deep within Saarthal.

OBJECTIVES

- Talk to the Arch-Mage
- Find the danger within Saarthal
- Follow Tolfdir
- Tell Tolfdir about the vision
- Follow Tolfdir
- Use the Saarthal Amulet to escape the trap
- Find Arniel Gane

Figure 4: Two Skyrim quests sharing objectives in the same dungeon, Saarthal.

States in the search space are generated on the fly. To do this, a successor generation function $succ(a)$ is defined to generate the set of all neighboring nodes reachable from input state a. This function generates successors according to all of the action choices available to the NPC, the effects those actions have on the world (e.g. attacking: if the NPC initiates an attack, this can harm enemies), and effects the world has on the NPC (e.g. physics: if from state a the NPC chooses to jump off a ledge, a result encoded in the corresponding successor b is that the NPC's position falls with gravity by an amount appropriate for the time cost $cost(a, b)$ of that edge.

The initial state of the game world is encoded as S^{start}, one of the states in the graph. A desired behavior outcome (e.g. a task from a game quest) defines the goal of the search problem. Since the goal may only be partially-specified, a function $SatisfiesGoal(a)$ is defined to compute whether or not state a satisfies the conditions of the behavior goal, rather than having a single goal state in the graph.

A graph search algorithm is used to find a path through the state space from the start state to any state satisfying the goal conditions. The path then encodes a sequence of actions which brings the NPC from S^{start} through the dungeon to the quest task goal defined by $SatisfiesGoal$.

Heuristic graph search algorithms like A* use a *heuristic* function $h(a, b)$ to focus the graph search process. A heuristic for A* search should be *admissible* and *consistent*. An admissible heuristic never overestimates the cost between two nodes. A consistent heuristic satisfies this triangle inequality: $h(a, c) \leq c(a, b) + h(b, c)$ (note that node b is a successor of a). For example, a simple admissible and consistent heuristic commonly used in, e.g., navigation planning is defined by the Euclidean distance between the NPC positions in states a and b. When the $cost$ unit is time instead of distance, Euclidean distance can be divided by the maximum possible travel speed to produce a result in units of time.

Weighted A* search is much like A*, but it inflates an admissible and consistent heuristic by a factor of w. This allows the search to find solutions which are guaranteed to be no worse in cost than w times the optimal solution cost. This tends to make searches find a path to the goal in substantially less time than plain A* can.

E-Graph Heuristic

The Experience Graph (E-Graph) method introduces a way to guide a graph search to utilize recorded experience paths. These E-Graph paths are recorded from prior graph search solutions, or they can come from demonstrations. The E-Graph algorithm is implemented by substituting the special E-Graph heuristic h^E in place of the original graph heuristic $h^G(a, b)$. For example, for navigation planning, the E-Graph heuristic may replace the Euclidean distance heuristic. The E-Graph method uses a new parameter $1 \leq \epsilon^E$ which controls how much the search prefers to use the experience paths. In extension from Weighted A*, when $\epsilon^E > 1$, the solution quality is bounded between the cost of the optimal solution and $w\epsilon^E$ times the cost of the optimal solution.

The E-Graph heuristic $h^E(a, b)$ can be defined as follows:

$$\min_{\pi} \sum_{i=0}^{N-1} min \left\{ \epsilon^E h^G(s_i, s_{i+1}), c^E(s_i, s_{i+1}) \right\} \qquad (1)$$

where π is a path $\langle s_0 ... s_{N-1} \rangle$, s_0 is a and s_{N-1} is b. $c^E(a, b)$ returns the ordinary cost $cost(a, b)$ if its inputs a and b are both on the E-Graph, otherwise returns an infinite value. This heuristic returns the minimal path cost from a to b where the path π is composed of an arbitrary number of two types of segments. One type of segment is a jump between s_i and s_{i+1} at a cost equal to the original graph heuristic inflated by ϵ^E. The other type of segment is an edge on the E-Graph, and its cost in π is its actual cost. In this way, the larger ϵ^E gets, the more the search prefers to utilize path segments on the E-Graph, since searching off of the E-Graph becomes costly.

Adaptability Across Quests

When the E-Graph heuristic (or T-Graph heuristic from (Drake, Safonova, and Likhachev 2016)) is used to guide an NPC behavior planner search to reuse training paths and the current quest configuration of a dungeon differs from its configuration in the training quest, search progress can be delayed by large local minima due to these differences. The contribution of this paper is in the use of Multi-Heuristic A* graph search to alleviate such issues.

Multi-Heuristic A* (MHA*) (Aine et al. 2015) performs a graph search while exploiting the guidance of multiple different heuristics. We will focus on the MHA* implementation called Shared Multi-Heuristic A*. MHA* essentially conducts multiple separate graph searches, each using a different heuristic. One search, called the anchor search, uses an admissible heuristic, while the other searches do not need

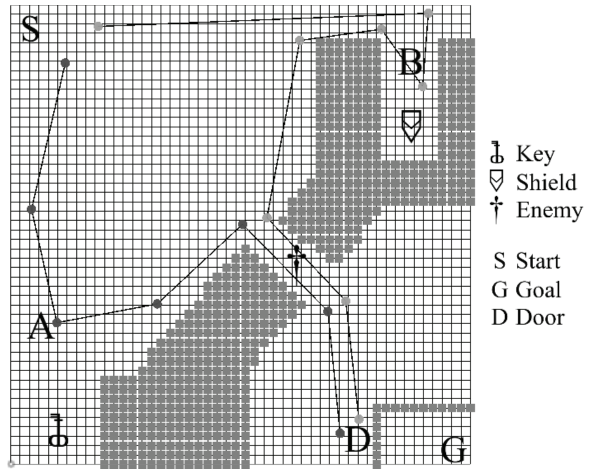


Figure 5: Test environment used in this paper to illustrate the advantage of using MHA*. The two lines with dots at each vertex represent E-Graph paths where the key and shield were instead located at points A and B, respectively.

to be admissible. Search g-values are shared across searches, which allows the algorithm to have parts of the solution path be found by different searches.

Consider the following example of a simplified game with an NPC and a dungeon environment. The NPC can navigate the environment by walking or sneaking, but there are obstacles in the environment blocking some paths. In addition, the NPC can complete some spatial events in the environment, such as picking up an item. A key and a shield are available in the environment. The NPC starts without holding the key nor the shield and must walk over these items to pick them up. There may be an enemy in the environment which attacks when the NPC gets near it. The enemy's attacks hurt less if the NPC is carrying a shield. There may be a door in the environment, which will not open unless the NPC possesses the key.

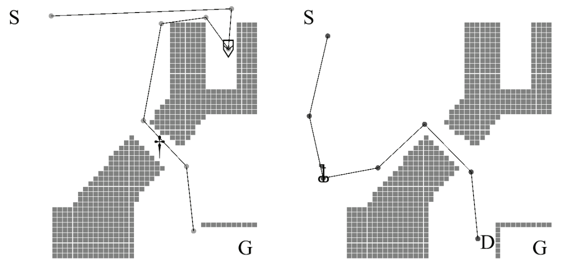


Figure 6: The configurations of the dungeon when the two demonstration paths were each recorded. On the left is a path demonstrating a behavior to pick up a shield before facing an enemy at the choke-point. The right side path demonstrates to pick up a key before reaching the locked door near the goal.

See Figure 6 and Figure 5 to see how a dungeon configuration can change between quests. In this example there was one dungeon iteration with a shield and an enemy, another dungeon iteration with a key and a door, and finally a third iteration with a shield in a new position, an enemy, a key in a new position, and a door.

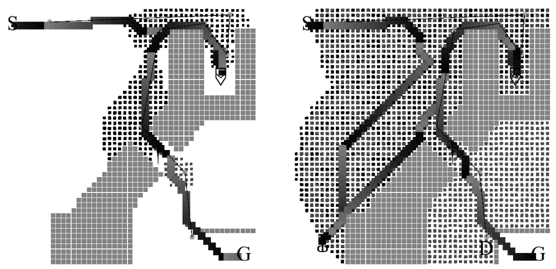


Figure 7: Left: The T-Graph planner is used to reach the goal quickly when given a demonstration similar to the solution. Right: The T-Graph planner encounters huge local minima when the door and key are introduced. Many states need to be expanded before the goal can be found.

See Figure 7 for a visualization of how the T-Graph method can perform well when the demonstration path closely matches the current quest configuration, but performance suffers severely when the quest changes.

We use h^S (Euclidean Distance heuristic) as the anchor heuristic in our MHA*-based NPC behavior planner. We use several inadmissible heuristics as follows. Let each event instance i be defined as a partially-specified world state where the event occurs. A property $T(i, s)$ is true whenever the effect of i being *T*riggered is detectable on state s (e.g. after picking up a shield, the shield is present in NPC inventory). For each event instance i, we include an additional heuristic $h^{Q_i}(a, b)$. This heuristic guides the search first toward the event location and then from the event location to b according to the E-Graph heuristic. Specifically, the heuristic is computed conditionally as defined in Equation 2:

$$h^{Q_i}(a,b) = \begin{cases} h^S(a,i) + h^E(i,b) & : \neg T(i,a) \\ h^E(a,b) & : \quad T(i,a) \end{cases} \tag{2}$$

If the event instance i has not been triggered (e.g. for the shield pick-up event, if the shield is not yet in inventory) at state a, then $h^{Q_i}(a, b)$ is the distance from a to i plus the E-Graph heuristic from i to b; otherwise if the event instance i has already been triggered (e.g. if the shield is already in inventory) at a, then $h^{Q_i}(a, b)$ is simply the E-Graph heuristic from a to b. A variation on our method would use a different heuristic (perhaps the E-Graph heuristic) for the distance between a and i in the case $\neg T(i, a)$, the case where event i has not yet been triggered at a.

Theoretical Properties

Our technique inherits theoretical properties from MHA*. Though we have focused on simply generating *feasible* solutions in less time, MHA* provides a bound on the cost of the solution path. Two parameters used within the MHA* algorithm are w_1 and w_2. The w_1 parameter inflates the heuristics used within the MHA* searches, and the w_2 parameter

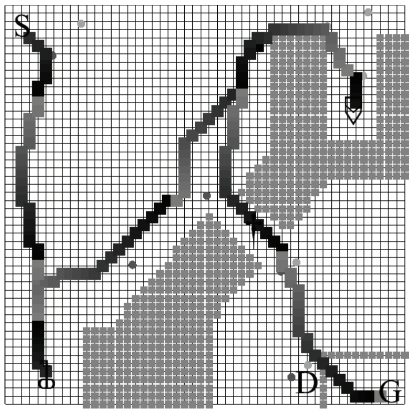


Figure 8: An example of a typical path planned to solve the quest task. The NPC starts at S, the start position, moves south to pick up the key, moves northeast and around a wall to get the shield, then moves to the center of the map to pass through the choke point with the enemy (which can only be survived with the shield), then finally moves southeast through the door (D), to the goal (G).

is a factor to prioritize the inadmissible searches over the admissible anchor search. The cost of the solution path found by MHA* is guaranteed not to be more than $w_1 * w_2$ times the cost of the optimal solution.

Though the E-Graph heuristic on its own introduces the possibility of sub-optimality in solution paths (in proportion to the size of the ϵ^E parameter), since we only use the E-Graph heuristic within the inadmissible searches of MHA*, ϵ^E has no effect on the sub-optimality guarantee of the MHA* search. MHA* provides its bound on sub-optimality independently of how inadmissible the inadmissible heuristics are.

Results

All of our results were generated on a Windows 7 machine with a 2.8GHz (up to 3.46 GHz) Intel i7-860 CPU and 12GB of dual-channel PC3-10600 RAM. Our code was written in C# for the Unity3d Game Engine.

To illustrate the benefit of using MHA*, we implemented a simplified NPC behavior planning problem. In the example test game (Figure 5) is an NPC whose state includes 2D (x & y) position (each discretized into fifty possible values), a health value from 0 to 100 (discretized in multiples of twenty, so there are six possible values), stealth mode (sneaking or not sneaking), and an inventory which can hold (or not) a key and hold (or not) a shield.

This test quest requires the NPC to acquire both the shield and the key to reach the goal. The quest task is to traverse the dungeon space to reach the goal position G. The goal is within a room blocked by a locked door, and the vicinity of the room can only be reached by making it past an enemy at a choke point. The enemy can only be passed alive while holding the shield. See Figure 8 to see an example of the kind of NPC behavior required to complete the quest task.

Two demonstrations, shown in Figure 5, each from a different quest in the same environment, are provided to the NPC. In one, there is no enemy at the choke point, and the key is in a slightly different location. In the other, there is no door, and the shield is in a slightly different location.

We compiled results for Weighted A* (including $w = 1$, which is standard A*), the T-Graph algorithm (ϵ^T is its inflation factor, like E-Graph's ϵ^E), and MHA* as described in the previous section. Various configurations of the w, ϵ^T, w_1, and w_2 parameters were tested and from these we chose the best for each algorithm. Listed in Figure 9 are the configurations we used. Because of the poor performance of the T-Graph method at adapting across quests, the optimum values for ϵ^T were actually all found to be 1, degenerating it to A* search, so we picked the value 2 instead to illustrate the problem. We generated MHA* results for $\epsilon^T = 2$ to compare, and also for $\epsilon^T = 100$ to show its capacity for improved performance for the same bounds.

As seen in Figure 10 the overhead of MHA* can cause it to run slower than the other methods in some cases when the bound factor is low, but as the bound factor increases, MHA* manages to outperform the other methods.

As seen in Figure 11, our MHA* approach outperforms the other methods in terms of expansion counts. Our MHA*

Label	w	ϵ^T	w_1	w_2	Bound Factor
					w
A	1				1
B	10				10
C	100				100
D	1000				1000
E	10000				10000
F	100000				100000
					$w * \epsilon^T$
G†	1	1			1
H†	5	2			10
I†	50	2			100
J†	500	2			1000
K†	5000	2			10000
L†	50000	2			100000
					$w_1 * w_2$
M*		2	1	1	1
N*		2	10	1	10
O*		2	20	5	100
P*		2	200	5	1000
Q*		2	2000	5	10000
R*		2	20000	5	100000
S*		100	1	1	1
T*		100	1	10	10
U*		100	1	100	100
V*		100	20	50	1000
W*		100	200	50	10000
X*		100	2000	50	100000

Figure 9: The parameter configurations used to generate performance results in Figure 11 and Figure 10. † indicates a T-Graph search, * indicates an MHA* search using our heuristics, all others are Weighted A*. The final column shows the sub-optimality bound for each configuration, used as a common reference to compare results between algorithms.

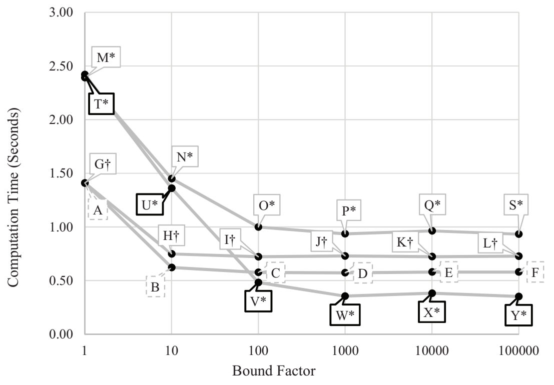


Figure 10: Computation time results. Reference Figure 9 for the configuration associated with each label.

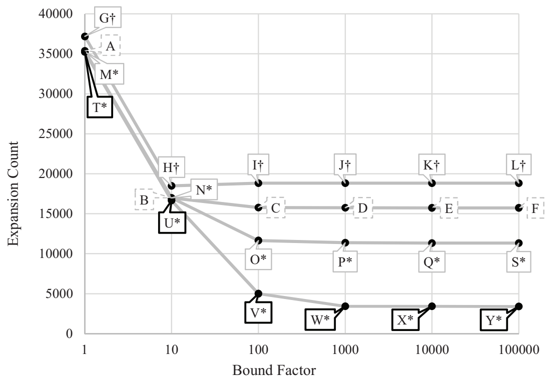


Figure 11: Expansion count results. Reference Figure 9 for the configuration associated with each label.

method performs the best because it adapts the demonstrations to the new event locations and utilizes information from both experience paths together while searching for the goal. Note that the MHA* samples for $\epsilon^T = 2$, N*, O*, P*, Q*, and S* outperform the corresponding T-Graph samples H†, I†, J†, K†, and L†, even though ϵ^T is the same and they have the same sub-optimality bound.

For a more complex problem, where each state expansion is more expensive (e.g. more expensive successor generation due to collision checking, world modeling, etc.), MHA*'s advantage in expansion counts outweighs the overhead of using MHA*. We tested this by adding several parameters to the NPC state, increasing the number of successors generated for a state from 16 to 256. Results are in Figure 12.

We also tested randomizing the key and shield locations to see how our MHA* method performs over a range of initial conditions. The key and shield were randomly placed in feasible locations (so they could be acquired before they are needed to solve the problem). Our results can be seen in Figure 13. MHA* greatly outperformed Weighted A* and T-Graph in these trials.

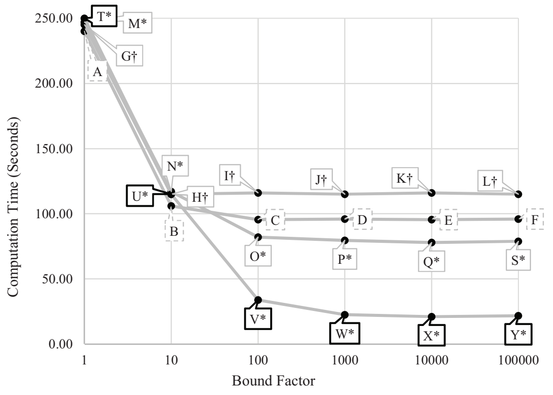


Figure 12: Computation time results for an expanded state space. Reference Figure 9 for the configuration associated with each label. Note that the computation times for MHA* have lowered significantly relative to the other search times, as compared to the standard state space results in Figure 10.

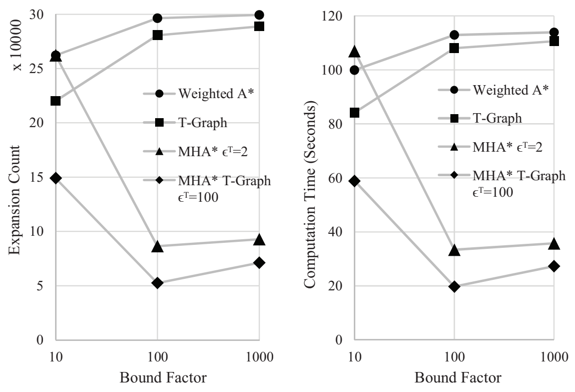


Figure 13: Randomized initial condition results. Like in the other trials, favorable parameters were selected to give each algorithm its best chance.

Conclusions & Future Work

In this paper, we introduce multiple heuristics based on the combination of pre-specified events and demonstrations. These heuristics combined with Multi-Heuristic A* yield significant performance improvements for the task of generating demonstration-based NPC behavior plans across quests in games. Our heuristic formulation guides a graph search to use experience, including partial experience, while adapting to the configuration of the current problem.

Future work includes further adaptation and generalization of experience and training as used for NPC behavior generation. Particularly, future work may explore the generalization of the spatial component of demonstrations so that those demonstrations can be used in environments entirely different in spatial configuration.

Acknowledgments

This work was supported by NSF Grant IIS-1409549

References

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and Likhachev, M. 2015. Multi-heuristic A*. *International Journal of Robotics Research (IJRR)*.

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*.

Drake, J.; Safonova, A.; and Likhachev, M. 2016. Demonstration-based training of non-player character tactical behaviors. In *Proceedings of the Twelfth Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

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

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* search with provable bounds on sub-optimality. In Thrun, S.; Saul, L.; and Schölkopf, B., eds., *Proceedings of Conference on Neural Information Processing Systems (NIPS)*. MIT Press.

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*.

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*.

Ratliff, N. D.; Silver, D.; and Bagnell, J. A. 2009. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots* 25–53.

Snook, G. 2000. Simplified 3d movement and pathfinding using navigation meshes. In DeLoura, M., ed., *Game Programming Gems*. Charles River Media. 288–304.