Automated Generation of Diverse NPC-Controlling FSMs Using Nondeterministic Planning Techniques

Alexandra Coman
Ohio Northern University
a-coman@onu.edu

Héctor Muñoz-Avila
Lehigh University
hem4@lehigh.edu

Abstract

We study the problem of generating a set of Finite State Machines (FSMs) modeling the behavior of multiple, distinct NPCs. We observe that nondeterministic planning techniques can be used to generate FSMs by following conventions typically used when manually creating FSMs modeling NPC behavior. We implement our ideas in DivNDP, the first algorithm for automated diverse FSM generation.

Introduction

Finite State Machines (FSMs) are among the most frequently-used mechanisms for representing NPC behavior. An FSM consists of a collection of states and a collection of transitions, where each transition connects two states. When used in the context of modeling NPC behavior, the states represent the actions that the NPC can perform, while the transitions represent observations of events that force the NPC to change the action it is currently executing (Fu and Houlette, 2003). Figure 1 shows an example of an FSM controlling an NPC (a variant of an FSM by Fu and Houlette, (2003)).

FSMs are intuitively understandable: non-programmers and programmers alike can conveniently describe and modify NPC behavior by editing FSMs instead of writing programming code. FSMs are also easily translatable into code. Tools like Bioware's Aurora engine automatically generate code from user-created FSMs and had been used to create sophisticated scripting capabilities (McNaughton *et al*., 2004).

Still, manually creating FSMs can be a cumbersome process, especially when the number of NPC actions and events is large. This problem is exacerbated in modern games, in which players have come to expect interacting with dozens to hundreds of characters. If all characters exhibit the same behavior, the game can fail to engage players. On the other hand, creating a set of diverse FSMs modeling the varied behavior of multiple NPCs places an additional burden on the game developers.

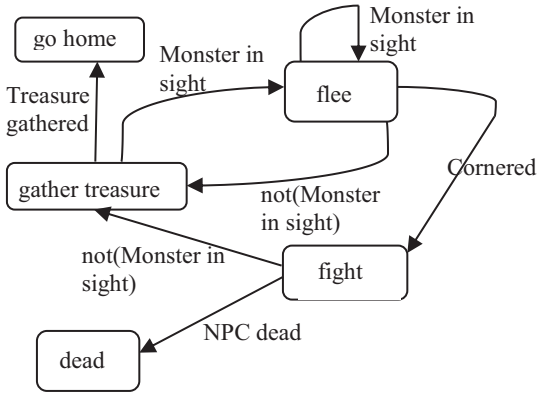

Figure 1. An example of an FSM.

We introduce DivNDP, an algorithm for the automated generation of diverse FSMs. It generates not only one FSM controlling an NPCs' behavior, but a set of FSMs modeling the behavior of multiple distinct NPCs. DivNDP uses nondeterministic (ND) planning techniques. As an application, we envision automatically generating a collection of FSMs which could then be tweaked by game developers.

Related Work

Our work is motivated by GOAP, i.e., goal-oriented action planning (Orkin, 2003). In GOAP, the game AI developer defines the actions that an NPC can perform. Using deterministic planning techniques, plans controlling NPCs are generated using these actions. The main differences between our approach and GOAP are that (1) we use ND planning techniques, (2) we generate FSMs instead of plans, and (3) we are focusing on solution diversity. We believe that ND planning can closely model the

nondeterministic nature of many games. For example, in the FSM in Figure 1, when fighting, the NPC might kill the monster opponent or die. Modeling nondeterministic outcomes occurs naturally in user-crafted FSMs. In the next section, we will discuss the shortcomings of using plans to model character behavior and argue that FSMs are a more natural choice.

Plan diversity has gained attention over the years and has been explored, among others, by Myers and Lee (1999) in HTN planning; and Srivastava *et al*. (2007) and Coman and Muñoz-Avila (2011) in deterministic heuristic-search planning. Bryce, Cushing, and Kambhampati (2007) explore diversity in probabilistic planning. We address diversity in non-probabilistic nondeterministic planning, which does not assume any knowledge about the probability distribution of the action outcomes, hence more accurately reflecting the characteristics of many domains in which it is unrealistic or impractical to assign probabilities to the outcomes of the actions, as pointed out by Ghallab, Nau, and Traverso (2004, page 432).

Outside planning, solution diversity has been explored extensively in case-based reasoning (Smyth and McClave, 2001; McSherry, 2002; McGinty and Smyth, 2003, and many others), predominantly for the purpose of matching diverse user preferences and needs in e-commerce.

Generating FSMs using Nondeterministic Planning Techniques

For automatically generating FSMs, we propose using non-deterministic (*ND*) planning techniques (Cimatti *et al*., 2003). In this form of planning, actions may have more than one possible outcome.

We believe that ND planning is a natural fit for modeling the dynamics of many games. In our example, the outcome of the *flee* action that an NPC might execute when it sees a monster could be that the NPC gets cornered or that it gets away from the monster. Here are informal definitions of this action and the *gather treasure* action:[1]

ACTION flee **Preconditions**: (Monster in sight) **Effects**: not (Monster in sight) OR (Cornered and Monster in sight) OR (Monster in sight)
ACTION gather treasure **Preconditions**: not(Monster in sight) **Effects**: (Treasure gathered) OR (Monster in sight)

Executing a plan, i.e., a sequence of actions $(a_1 \ldots a_n)$, as done in GOAP, may not guarantee that the goal will be reached. The reason for this is that the actions have ND effects: action a_{k+1} might assume a specific outcome of action a_k, but, during execution, action a_k might result in a different outcome, thereby making action a_{k+1} inapplicable. For example, *flee* might be followed by *gather treasure* in the plan but, while executing the plan, the NPC might end up cornered, with the monster still in sight, which renders the *gather treasure* action not executable.

Nondeterministic planning addresses this issue by generating ***policies*** (sets of state-action pairs) instead of plans. A policy accounts for all possible outcomes of the actions. Formally, a policy π is a mapping:

$$\pi: S \rightarrow A$$

This mapping indicates, for every state $s \in S$, the action $a \in A$ that must be executed if the NPC reaches state s. Intuitively, the policy indicates a "way out" of the state. The crucial point is the following observation about the ***convention*** for manually constructing FSMs (e.g. Fu and Houlette, 2003): in an FSM, states represent the action currently being executed by NPCs (e.g., *gather treasure* in Figure 1) and state transitions represent states of the game world observed by the NPC (e.g., *Monster in sight*). When following this convention, policies are FSMs.

As in GOAP, we must define the actions that an NPC can perform but, unlike in GOAP, these actions are nondeterministic, so they may have multiple possible outcomes.

The FSM shown in Figure 1 can be generated using ND planning techniques. Assume we specify the goal *Treasure gathered* and, as initial state, *no(Monster in sight)*. When an ND planner selects the action *gather treasure*, it will notice that a possible outcome of this action is the desired goal, but that another possible outcome is *Monster in sight*. Hence, the planning process will have to continue in order to account for this second possibility. In the next section, we describe a procedure for automatically generating multiple diverse FSMs.

There is a question of when the ND planner has generated a solution policy. There are multiple definitions of what it means for a policy to solve a problem. These definitions are based on the notion of a ***policy execution path***, which is the sequence of actions executed in a particular run, while following the policy starting from the initial state. Subsequent runs of the policy might result in different policy execution paths because of the ND outcomes. Ideally, any policy execution path would yield the goal, but this might not be possible; in Figure 1, after the action *fight* has been executed, the NPC might be dead, making the *Treasure gathered* goal unreachable. The FSM in Figure 1, viewed as a policy meant to reach this goal, is considered a ***weak solution*** because only some of the policy execution paths will reach the goal. The algorithm we present is able to generate both weak and strong cyclic solutions. ***Strong cyclic solutions*** guarantee that any policy

[1] The formal definitions would use first-order logic notation, e.g. *flee(n,m)*, where *n* is the NPC and *m* is an enemy monster.

execution path will reach the goal in a finite number of steps, provided it exits any loop that it enters. If a strong cyclic solution to the problem does not exist, the algorithm could generate a weak solution. For more details on these different types of solutions, see Cimatti *et al*. (2003).

FSM Diversity

We aim to generate **sets of diverse policies representing FSMs**. The degree of difference between two policies can be evaluated based on criteria we call *policy distance metrics*. An ND planning problem is defined as a *(S_0, G, Dom)* triple, where S_0 is the set of initial states, *G* is the set of goal states, and *Dom* is the domain description (i.e., a collection of ND action descriptions).

The diversity of a set of policies is the average pair-wise distance between policies in the set (Myers and Lee, 1999, in deterministic planning). In Equation 1, Π is a non-empty set of policies, π and π' are policies, and *D* is a *policy distance metric* (a measure of the dissimilarity between two policies).

$$Div(\Pi) = \sum_{\pi,\pi' \in \Pi} \frac{D(\pi,\pi')}{\frac{|\Pi| \times (|\Pi|-1)|}{2}} \quad (1)$$

The distance metric *D* can, as in the case of deterministic planning, be either quantitative (Srivastava *et al*., 2007) or qualitative (Coman and Muñoz-Avila, 2011).

Quantitative diversity is domain-independent and, while requiring less knowledge engineering, may not reflect meaningful differences between solutions to planning problems. For example, in the FSM in Figure 1, the action *flee* would be considered, by a quantitative distance metric, equally distant from the actions *go home* and *fight*, although the first two actions indicate conflict avoidance, while the third one indicates the intention to engage in battle.

As an example of a quantitative policy distance metric, we define D_{PS} ("pair set" distance; Equation 2), derived from a quantitative plan distance metric commonly used in deterministic planning (Srivastava *et al*., 2007; Coman and Muñoz-Avila, 2011).

$$D_{PS}(\pi, \pi') = \frac{|\pi \setminus \pi'| + |\pi' \setminus \pi|}{|\pi| + |\pi'|} \quad (2)$$

In Equation 2, π and π' are policies, |π| is the number of state-action pairs in policy π, and π \ π' is the set of state-action pairs which are in π, but not in π'.

Qualitative diversity is domain-dependent and reflects meaningful differences between solutions, but requires domain-specific information (degrees of distance between actions or state-action pairs, e.g *fleeing* is more similar to *going home* than it is to *fighting*) to be provided.

We define two subtypes of qualitative distance metrics based on whether they reflect different ways of achieving the goals: goal-achievement distance and inflation-based distance.

Goal-achievement distance reflects differences between policies in terms of the way in which they achieve the goals, with different policies embodying varied approaches to achieving a given goal or set of goals.

Inflation-based distance reflects policy differences that are not related to the way in which goals are achieved. *Inflation*, i.e. the addition, for the sake of increasing policy-set diversity, of state-action pairs which do not specifically contribute to achieving the goals, was previously considered inherently bad, being described as unnecessarily increasing the size of solutions (Coman and Muñoz-Avila, 2011). While this may certainly be the case with meaningless inflation (e.g. inflation obtained using quantitative distance), in planning for games and interactive storytelling, larger solution policies may include side-stories revealing characters' personalities and motivations, while a policy execution path which is the shortest path from the initial state to a goal state may prove comparatively dull.

We exemplify all these types of policy distance in a real-time strategy game environment. The character units are of various types (e.g. *peasants*, *soldiers*, *archers*, *mages*). They can fight enemies, harvest resources, build villages, etc. Assume that the goal is for the enemy base to be destroyed.

In such an environment, the following is a set of policies which are diverse in terms of goal-achievement. Assume a state s_1 in which our units are located near the enemy base, and three policies π_1, π_2 and π_3 such that $\pi_1(s_1)$ is action *soldier attack*, $\pi_2(s_1)$ is action *mage attack*, and $\pi_3(s_1)$ is action *archer attack*. These three policies differ solely in the way in which they achieve the goal; all three attacks are legitimate ways of attempting to reach the goal, but they are significantly different, because different unit types have different strengths and weaknesses. Inflation (the addition of state-action pairs which are not essential for reaching the goal) is not needed to create diversity in this example.

Inflation-based policy diversity allows us to simulate character variation in terms of personality traits. The following policies exemplify this: π_4 includes a detour to a village, where the units attack non-hostile peasants (such behavior reflects ruthless personalities), π_5 includes building an extension to a village (indicative of kindness), and π_6 includes collecting a treasure (indicative of greed).

Generating Diverse FSMs

We introduce DivNDP, a diverse planner which incorporates NDP (Kuter *et al*., 2008), an algorithm for solving nondeterministic planning problems by converting nondeterministic planning domains into deterministic ones, then repeatedly calling a planner *PL* (e.g. FF, by Hoffman and Nebel, 2001) for deterministic planning domains, and putting together a policy from the plans *PL* generates.

Assume that *P* is a deterministic planning problem with the goal state *g* and the initial state s_0. *PL* builds a solution plan to *P* in the following way: for the current state *s* (initially, $s = s_0$), it looks for actions that are applicable in *s* (actions the preconditions of which hold in *s*). It chooses *a*, one of the applicable actions, using a heuristic evaluation function (herein referred to as h_{PL}). Assume that *s*' is the state that is obtained by executing *a* in *s*. The heuristic evaluation of *a*, using function h_{PL}, is the estimated length of the plan from *s*' to the goal state *g*.

DivNDP generates a set of *k* diverse policies as follows (also see the pseudocode below).

First, the regular version of NDP is used to convert the nondeterministic planning domain into a deterministic one and to generate the first policy.Nondeterministic planning domains are converted into deterministic ones by replacing each nondeterministic action *a*, where *a* has *n* possible outcomes, with *n* deterministic actions $a_1,..,a_n$, where each a_i corresponds to one possible outcome of the original action *a*. For example, the ND action *flee* (which has 3 possible outcomes) would be replaced with 3 deterministic actions: *flee1*, the outcome of which is *not*(*Monster in sight*); *flee2*, with the outcome (*Cornered and Monster in sight*); and *flee3*, with the outcome (*Monster in sight*). A set of open nodes (*ON*) is maintained: the open nodes represent possible states of the world that the policy should include. In the beginning, only the initial state of the problem is included in *ON*. *PL* is run on the planning problem using this deterministic domain description. On the first run, if successful, *PL* produces a complete plan p_1 from an initial state s_0 to a goal state *g*. Each time a deterministic action a_i originating from a nondeterministic action *a* is added to the partial plan constructed by *PL*, all states corresponding to outcomes of *a* other than the intended one, called *failed effects* (Fu *et al*., 2011), are added to *ON*. In our example, if the action *flee1* is selected by *PL*, the states (*Cornered and Monster in sight*) and (*Monster in sight*) are the failed outcomes that are added to *ON* (because they are the effects of *flee* which are not also effects of *flee1*). At each step, until no open nodes remain, an open node *o* from *ON* is chosen arbitrarily, and *PL* is used to attempt to generate a plan p_k from *o* to a goal state, following the same procedure as above.

These plans generated with *PL* are used to put together a policy π. After generating each plan p_k, the state-action pairs (*s*, *a*), where *a* is the nondeterministic action corresponding to deterministic action a_i in p_k, and *s* is a state in which a_i is applied in p_k, are added to the partial policy π (unless π already contains state *s* and an associated action). For example, if p_k contains the action *flee1*, then the state-action pair (*Monster in sight*, *flee*) will be added to π.

The policy π is complete when either (a) no open nodes remain, i.e. *ON* is empty; or (b) *PL* has attempted (and, possibly, failed) to generate a plan for each open node. In case (a), the generated policy is a strong cyclic one. In case (b), the policy is a weak one[2].

Assume that we have generated one policy modeling NPC behavior. To generate (*k*-1) additional diverse policies, we conduct the policy-generating process described above *k*-1 more times, but using $PL^{Div}(\Pi)$, a modified version of *PL*. $PL^{Div}(\Pi)$ uses a modified heuristic function h_{Div} (Equation 3), instead of h_{PL}, to assess candidate actions.

The h_{Div} formula is made up of two components: h_{PL} and *RelDiv* (an estimate of how dissimilar a policy extracted from the partial plan p_p would be from the set of previously-generated policies Π). The parameter α enables the adjustment of the weights assigned to policy diversity and goal distance. The regular *PL* heuristic h_{PL} must be minimized, while the diversity *RelDiv* must be maximized, hence the corresponding signs (assuming *PL* seeks to minimize heuristic values).

$$h_{Div} = -\alpha RelDiv(p_p,\Pi) + (1-\alpha)h_{PL} \quad (3)$$

The *RelDiv* formula is shown in Equation 4: the parameters are p_p (a partial plan) and Π (a set of policies). |Π| is the number of policies in Π, π is one such policy, and *RelD* is a distance metric indicating the degree of difference between a plan p_p and a policy π. $RelD_{PS}$ (Equation 5) is an example of such a distance metric. In Equation 5, *length(p)* is the number of actions in *p* and *P(p)* is the set of state-action pairs (*s*, *a*), where *a* is the ND action corresponding to deterministic action a_i in *p* and *s* is a state in which a_i is applied in *p*. P(p)\π is the set of state-action pairs which are in P(p), but not in π.

$$RelDiv(p_p,\Pi) = \sum_{\pi\in\Pi} \frac{RelD(p_p,\pi)}{|\Pi|} \quad (4)$$

$$RelD_{PS}(\mathrm{p},\pi) = \frac{|\mathrm{P(p)}\backslash\pi|}{length(p)} \quad (5)$$

[2] It is assumed that the planner *PL* is complete, so that if it does not find a plan from an open node to a goal, no such plan exists.

In computing $RelD_{PS}$, we only take into consideration state-action pairs (s, a) in $P(p)$ but not in π (and not the other way around). We do this to avoid shorter partial plans being evaluated as more dissimilar from π than longer ones. Furthermore, states which do not appear in $P(p)$ but appear in π are less relevant for the distance computation, as state-action pairs may be added to $P(p)$ later on in the planning process, while the policies in Π are already complete.

The state-action pairs in $P(p)$ contain the original nondeterministic actions, not their deterministic versions in p; any two deterministic actions, a_i and a_k, originating from the same nondeterministic action a are considered identical for the purposes of this comparison.

The pseudocode of DivNDP is shown below (simplified for brevity and readability). *Dom* is a ND planning domain, S_0 is the set of initial states of the ND planning problem, G is its set of goal states, *PL* is the planner for deterministic domains, $PL^{\text{Div}}(\Pi)$ is a version of *PL* which uses the heuristic function h_{Div} (Equation 3), and k is the number of diverse policies to be generated.

DivNDP(*Dom*, S_0, G, *PL*, k)
1. *Dom'* ← a deterministic relaxation of *Dom*
2. $\pi \leftarrow \varnothing$
3. $ON \leftarrow S_0 \setminus G$
4. While $ON \neq \varnothing$
5. $s \leftarrow$ randomly selected state in ON
6. $ON \leftarrow ON \setminus \{s\}$
7. $p \leftarrow$ solution plan produced by *PL* to problem (s, G[3], *Dom'*)
8. If $p \neq \varnothing$ //planning attempt has not failed
9. $\pi \leftarrow \pi \cup \{(s, a) \mid a_i$ is an action in p, a is the nondeterministic action corresponding to a_i, and s is a state in which a_i is applied in $p\}$
10. Add *failed effects* to ON
11. Add π to Π
12. For i = 1 to k-1
13. $\pi \leftarrow \varnothing$
14. $ON \leftarrow S_0 \setminus G$
15. While $ON \neq \varnothing$
16. $s \leftarrow$ randomly selected state in ON
17. $ON \leftarrow ON \setminus \{s\}$
18. $p \leftarrow$ solution plan produced by $PL^{\text{Div}}(\Pi)$ to problem (s, G, *Dom'*)
19. If $p \neq \varnothing$ //planning attempt has not failed
20. $\pi \leftarrow \pi \cup \{(s, a) \mid a_i$ is an action in p, a is the nondeterministic action corresponding to a_i, and s is a state in which a_i is applied in $p\}$
21. Add *failed effects* to ON
22. Add π to Π
23. Return Π

[3] The problem is solved when any goal state g in G has been reached.

Experimental Evaluation

We implement DivNDP using JavaFF (Coles *et al*., 2008), a Java implementation of the FF planner (Hoffmann and Nebel, 2001), as the deterministic planner *PL*. Our implementation of NDP is similar to the one described by Fu *et al.* (2011), including their two extensions, state reuse and goal alternative, which are shown to contribute to outperforming other strong-cyclic planners.

We generate sets of k=4 diverse policies.

We conduct the experimental evaluation on the real-time strategy game Wargus. Attack actions in Wargus are nondeterministic (e.g. the success of an *attack* action is not guaranteed). The use of nondeterministic planning makes it possible for the generated solution to succeed in the game even in the case of unexpected losses. For example, a policy might specify that a *soldier* NPC should attack if such an NPC is available, and that an *archer* NPC should attack otherwise. Alternatively, the policy might specify that a new *soldier* NPC should be created if no *soldiers* are available.

The game scenarios vary in terms of the elements on the map (e.g. villages, treasures) and their locations. Our distance metrics are independent of the map (i.e. they do not specify any particular map coordinates). Units can move to various map locations, build villages, collect available treasure, and attack hostile and neutral locations. Nondeterminism manifests itself in battle: the result of an attack action can be either success (the unit has killed the enemy without being killed itself) or failure (the unit was killed). In the initial state, units of all types are available. The goal is to destroy all enemies on the map.

We exemplify both inflation-based and goal-achievement qualitative distance metrics. Goal-achievement distance is computed using Equation 6, which considers policies to be different if the types of units used to attack are different. For example, if s is a state in which all types of units are available and near the enemy camp, for two policies π and π', π(s) could be the action *soldier attack*, whereas π'(s) is the action *archer attack*. In Equation 6, π and π' are policies, *FirstAtt*(π) is the type of attacking unit (*archer*, *soldier*, *mage*, or *peasant*), and d is a domain-specific degree of distance between unit types (e.g. a *peasant* is considered more similar to a *soldier* than to a *mage* because the former two employ melee attacks, while the latter employs long-range attacks).

$$D_{GA}(\pi, \pi') = \begin{cases} 0, if\ FirstAtt(\pi) = FirstAtt(\pi') \\ d, 0 < d \leq 1, if\ FirstAtt(\pi) \neq FirstAtt(\pi') \end{cases} \quad (6)$$

Inflation-based distance is used to simulate diversity in terms of character personality traits. In Equation 7, π and π' are policies, *CharTrait*(π) is the *character trait* reflected

in policy π through small side-stories not essential to reaching the goal, while *d'* is a degree of distance between possible personality traits. The personality traits represented are: (1) *ruthless* (attacks non-hostile village), (2) *hard-working* (builds extension to village), (3) *greedy* (collects treasure), (4) *indifferent* (does none of the above, focusing solely on achieving the goal).

$$D_{Personality}(\pi, \pi') = \begin{cases} 0, if\ CharTrait(\pi) = CharTrait(\pi') \\ d', 0 < d \leq 1, if\ CharTrait(\pi) \neq CharTrait(\pi') \end{cases} \quad (7)$$

These two types of qualitative distance are compared to the D_{PS} quantitative distance metric (see Equation 2).

Evaluation Metrics

To evaluate the diversity of the sets of policies generated with DivNDP, we use three evaluation metrics. The first evaluation metric is **policy set diversity** (Equation 1).

The second evaluation metric is the **Shannon entropy** over the possible modifications to the game map (e.g. treasure picked up, villages destroyed, units lost) caused by executing each policy. The higher the entropy, the more diverse the generated policies in terms of the way in which they affect the game environment when executed. Entropy-based evaluation of the diversity of plans for NPCs was previously used by Coman and Muñoz-Avila (2012), in a deterministic planning context. For goal-achievement distance, we look at map elements in the enemy camp region of the map, while, for inflation-based distance, we consider elements in the other regions. All policies lead to the achievement of the goal. However, while the goal is always achieved in all final states, there is variation regarding the other facts in the final states (e.g. whether villages have been plundered or extended), none of which are considered more desirable than others a priori.

The third performance metric is the average **generated-policy size**. Like Fu *et al*. (2011), we measure policy size as number of state-action pairs and consider smaller policies preferable. Smaller FSMs are desirable since they are likely to be easier to understand.

Experimental Results

We now discuss the results obtained for each of the three performance metrics.

Policy-set diversity is always maximal for both the inflation-based and the goal-achievement qualitative distance metrics but not for the quantitative distance metric.

The **entropy** results are presented in Figures 2 a) for goal-achievement qualitative distance and 2 b) for inflation-based qualitative distance. Each point on each chart is the average entropy value over 25 runs of a generated policy for one game scenario. The repetition is necessary due to the nondeterminism of the game; when running the same policy on the same game map multiple times, the policy execution paths may differ. For both inflation-based and goal-achievement qualitative distance, the entropy is consistently greater for qualitative distance results than for quantitative distance results.

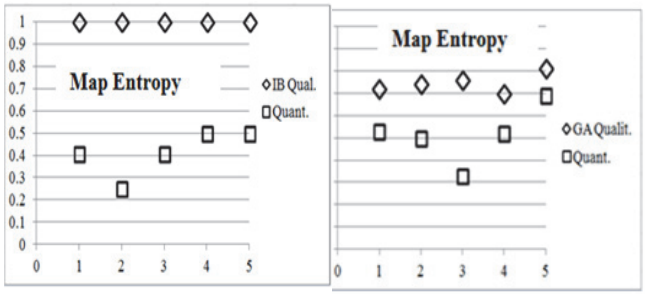

Figure 2. Map Entropy Results on the Wargus domain: Quantitative Distance vs. (a) Goal-Achievement, (b) Inflation-based Distance

The average **policy size** is slightly larger for quantitatively-diverse policies than for qualitatively-diverse policies. Quantitatively-diverse policies are often inflated in a non-meaningful way, containing state-action pairs added solely for the purpose of increasing policy-set diversity (as demonstrated by their lower entropy results).

Conclusions and Future Work

We introduced DivNDP, an algorithm which uses nondeterministic planning techniques to automatically generate diverse FSMs controlling NPC behavior. This is achieved by following typical conventions used when manually creating FSMs for modeling NPCs. DivNDP is the first algorithm for diverse non-probabilistic nondeterministic planning and can be used with both quantitative and qualitative policy distance metrics. In our experiments, the use of qualitative distance metrics resulted in more diverse FSMs than the use of quantitative distance metrics. We have also defined and demonstrated two subtypes of qualitative distance metrics: goal-achievement and inflation-based distance metrics.

The main drawback of our current approach is the need for the user to provide a collection of action definitions. Orkin (2003) argues that the benefits of automated NPC behavior generation outweigh the effort of manually defining actions. While we concur, in future work, we would like to explore the possibility of learning action descriptions and distance metrics from game traces: Yang, Wu, and Jiang (2005) have addressed learning action models outside ND planning.

Acknowledgements. This work was supported in part by NSF grant 1217888.

References

Bryce, D.; Cushing, W., and Kambhampati, S. 2007. Model-lite Planning: Diverse Multi-option plans & Dynamic Objective Functions. ICAPS 2007 Workshop on Planning and Plan Execution for Real World Systems.

Cimatti, A.; Pistore, M.; Roveri, M, and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence*, 147(1-2):35–84.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Teaching Forward-Chaining Planning with JavaFF. Colloquium on AI Education AAAI 2008.

Coman, A., and Muñoz-Avila, H. 2011. Generating Diverse Plans Using Quantitative and Qualitative Plan Distance Metrics. In Proc. of AAAI 2011, 946-951. AAAI Press.

Coman, A., and Muñoz-Avila, H. 2012. Plan-Based Character Diversity. In Proc. of AIIDE 2012, 118-123. AAAI Press.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search, *Journal of Artificial Intelligence Research*, 14:253-302.

Fu, D. and Houlette, R. 2003. The Ultimate Guide to FSMs in Games. In: Rabin, S. (Ed.) *AI Game Programming Wisdom 2*. Charles River Media.

Fu, J.; Ng, V.; Bastani, F.B., and Yen, I. 2011. Simple and Fast Strong Cyclic Planning for Fully-Observable Nondeterministic Planning Problems. In Proc. of IJCAI 2011, 1949-1954. IJCAI/AAAI Press.

Ghallab, M.; Nau, D.S., and Traverso, P. 2004. Automated Planning: Theory and Practice. Morgan Kaufmann.

Kuter, U.; Nau, D. S.; Reisner, E.; and Goldman, R. P. 2008. Using Classical Planners to Solve Nondeterministic Planning Problems. In Proc. of ICAPS 2008, 190-197. AAAI Press.

McGinty, L., and Smyth, B. 2003. On the Role of Diversity in Conversational Recommender Systems. In Proc. of ICCBR 2003. 276–290. Springer.

McNaughton, M.; Schaeffer, J.; Szafron, D.; Parker. D., and Redford, J. 2004. Code Generation for AI Scripting in Computer Role-Playing Games. In Proc of. Challenges in Game AI Workshop at AAAI 2004, 129-133.

McSherry, D. 2002. Diversity-Conscious Retrieval. In Proc. of ECCBR 2002, 219–233. Springer.

Myers, K. L., and Lee, T. J. 1999. Generating Qualitatively Different Plans through Metatheoretic Biases. In Proc. of AAAI 1999, 570–576. AAAI Press/MIT Press.

Orkin, J. 2003. Applying Goal-Oriented Action Planning to Games. In: *AI Game Programming Wisdom 2*. Charles River Media.

Smyth B., and McClave, P. 2001. Similarity vs. Diversity. In Proc. of ICCBR 2001, 347–361. Springer.

Srivastava, B.; Kambhampati, S; Nguyen, T.; Do, M.; Gerevini, A.; and Serina, I. 2007. Domain Independent Approaches for Finding Diverse Plans. In Proc. of IJCAI 2007, 2016-2022. IJCAI/AAAI Press.

Yang, Q.; Wu, K., and Jiang, Y. 2005. Learning Action Models from Plan Examples with Incomplete Knowledge. In Proc. of ICAPS 2005, 241-250. AAAI Press.