Dynamic DDN Construction For Lightweight Planning Architectures

William H. Turkett, Jr.
Department of Computer Science
Wake Forest University
Winston-Salem, North Carolina 27109

Abstract

POMDPs are a popular framework for representing decision making problems that contain uncertainty. The high computational complexity of finding exact solutions to POMDPs has spawned a number of research projects which are investigating means of quickly finding high quality approximate solutions. This work uses information gained at runtime to reduce the cost of reasoning in POMDP type domains. Run-time dynamic decision network construction is introduced and the role it plays in an agent architecture for reasoning in POMDP domains targeted to run locally on personal digital assistants (PDAs) is evaluated. The ability to suport reactive, high quality approximate planners on lightweight devices should help to open the door for localizable and personalizable POMDP applications.

Introduction

Recent market research (Gartner 2006) indicates continued growth in the PDA and Smartphone market. As these devices become more popular, more and more people will have at hand lightweight portable computing devices. The question of how to exploit these devices beyond the standard cellular, scheduling, and Internet access capabilities is an interesting and important question.

Recent work suggests that there are several mobile device application domains where incorporating AI, and in particular decision theoretic, algorithms could be of practical use. Tambe's *Electronic Elves* (Scerri, Pynadath, & Tambe 2001) agents incorporate Markov Decision Processes (MDPs) as their reasoning mechanism and use these to control autonomous decision making, including re-scheduling meetings and ordering the PDA owner's lunch. Bohnenberger (Bohnenberger *et al.* 2002) has used MDPs to support PDA-based location aware shopping assistance. POMDP applications applicable to PDAs have been proposed in the recent past, including WiFi-based localization and navigation (Ocana *et al.* 2005), ad-hoc network packet routing (Chang, Ho, & Kaelbling 2004), and intelligent wireless spectrum usage (Zhao & Swami To appear). A potential and very exciting example of a POMDP application that is amenable to PDA usage is intelligent tutoring. POMDP-based tutoring systems (Murray, VanLehn, & Mostow 2004) available on PDAs could allow for novel learning mechanisms.

This paper describes an architecure that has been developed for reasoning in POMDPs on lightweight hardware. Due to the difficulty of finding exact solutions to problems modeled as POMDPs, a significant portion of POMDP research in the past couple of years has been on efficiently finding high-quality approximate solutions. The core of the developed architure outlined in this paper involves the use of dynamic decision networks (DDNs) to perform run-time decision making. The intent of a move away from pre-execution policy optimization to run-time decision making is to exploit any information gained at runtime to simplify the decision making problem. The developed approach is approximate: a limit on the number of timepoints represented in the DDN must exist to ensure computational feasability, and that limited lookahead requires a heuristic estimate of the future value of states. The key point of this paper, however, is to describe the process of dynamic DDN construction and state how it can be used to reduce the computational requirements of decision-network based action selection. Dynamic DDN construction as used in this work is quality-preserving, as it allows the same solutions to be found that would have been found if dynamic DDN construction was not used.

The developed planning architecture has been designed with lightweight and lowcost hardware and software in mind. Development over the last year has been on the HP Ipaq 4150 PDA and has used freely available software tools.

Background

Partially Observable Markov Decision Processes (Cassandra 1999) can be used to represent scenarios in which there exists uncertainty in both the outcomes of actions an agent may perform and in the observations an agent may receive from the environment. An agent in a POMDP environment maintains a probability distribution, called a belief state, over the possible states of the world. Given a belief state, an agent selects an action to perform through a reasoning mechanism, such as policy lookup or DDN lookahead, and then updates its belief state given the observation received as feedback. Implemented reasoning mechanisms return the maximum utility action for the current state of the world (this maximum utility action may not be the true optimal action, how-

ever, if approximate reasoning algorithms are used, but it should be considered as the best choice given the implemented reasoning mechanism). For a deeper introduction to POMDPs and exact approaches to solving POMDPs, see (Cassandra 1999).

Dynamic decision networks (DDNs) are an extension of Bayesian networks (Jensen 2001) which can be used to implement utility based planning. Dynamic decision networks extend Bayesian networks over a finite planning horizon and can represent both partially observable states and probabilistic action outcomes (Russell & Norvig 2002). A decision node exists at each time slice to represent the action to be performed in that slice, and the effects of a combination of the agent's current state and potential actions to be taken can be propagated through the decision network. Utilities can be represented in each time slice if the reward is time-separable or at the end of the sequence of actions (Russell & Norvig 2002; Forbes *et al.* 1995).

The use of dynamic construction of Bayesian networks to allow for effective representations of problems has been studied in the past (Provan 1993; Nicholson & Russell 1993; Nicholson & Brady 1994; Ngo *et al.* 1997). This work has primarily been directed towards dynamic construction from a knowledge base in response to a query presented by the user. Provan (Provan 1993) specifically states that, due to the complexity of inference in temporal networks, dynamic construction is key to being able to perform inference under reasonable time constraints. Provan's work is directed towards construction of temporal influence diagrams for use in medical applications. He describes heuristic techniques such as temporal modeling of just a subset of the variables (such as those driving the network or just the observation variables) and metrics for evaluating the effects of the loss of information from such pruning. Nicholson and Brady (Nicholson & Brady 1994) describe an exact technique for removing unneeded states from a network. Removing states is possible when the probability of being in a given state is set to zero. This state no longer needs to be modeled and can be removed from a given variable. When such a state is removed, states in the successor variables can also be removed if they are no longer possible given the states with probability greater than zero. A combination of these ideas is used in this work as part of a general DDN architecture for reasoning in POMDP-type domains.

In a set of informal notes (Cassandra 2003), Cassandra describes a subclass of problems that can be modeled as POMDPs that can be optimally solved in polynomial instead of exponential time. The problems discussed by Cassandra have the following properties: a uniform initial distribution over states, observational uncertainty that only stems from states that look similar (leading to a set of states that go to zero probability after each action is taken and observation is received), and, after each iteration, the set of states with non-zero probability shrinks.

It is exactly this type of information gathering that a dynamic DDN construction algorithm is able to exploit. In these types of domains, the agent is taking significant advantage of its information-gathering actions in reducing uncertainty, and its reasoning mechanisms should also be able to exploit such a reduction in uncertainty. While for the exact set of problems described by Cassandra, an exact approach can (and should) be used, a dynamic DDN construction algorithm should be effective in reducing planning costs for other problems that don't fit the Cassandra definition but where a reduction in uncertainty occurs. The critical point is that the cost of building a reduced network and then planning with that network has to be less than planning with the original full state space network. For domains with noisy observations and significant action uncertainty, the construction process may often result in a network being built much like an original network. The overhead of doing construction will then outweigh the usefulness of the dynamic construction approach. The algorithms developed in this work perform a significant amount of pre-construction activity, trading space (storage of reachability information) for time, as a means of reducing the actual construction time required.

Dynamic DDN Construction

An uninformed implementation of DDNs for planning contains factored representations of the entire state space of a POMDP domain. All possible states for each variable are represented in the respective variable nodes. This approach allows one DDN to be designed by an agent designer which can be used at anytime by the agent. If an agent is planning at runtime, however, it has specific beliefs about the current state of the world. After performing several actions, an agent's beliefs may start to center around only a small number of states, with the likelihood of the other states becoming zero. A related structural feature that can be exploited in many domains is fully observable variables in the state space. In these instances, the beliefs states over such a variable are always collapsed to a single belief in one state.

Given a description of the variables needed in a DDN, a description of the states for each variable, and the transition and observation probabilities, it is possible to specify, for any current state of the world, all future states of the world that can be reached and the corresponding observations that could be seen, given a finite number of time-points are being modeled.

The process for finding such states is implemented through dynamic programming, starting with the assumption that one action is taken. Given a current state of the world, a set of next possible states is defined for each action in the transition table. The set of all possible next states for the current decision is the union of the outcome states that are entered from all possible actions. The corresponding observations can be found at the same time, as they are tied to (previous-state, action) pairs or outcome states. For each timestep after the first, the set of reachable states is the union of states reachable from the set of reachable states found for the previous timestep. A pseudo-code description of this algorithm can be found as Algorithm 1. This algorithm descends from the knowledge base construction algorithms used by Nicholson and Brady (Nicholson & Brady 1994) to perform data-driven construction of dynamic Bayesian networks. The work presented here is different than that of Nicholson and Brady in several ways. Their work primarily considered techniques for expanding a monitoring DBN at

each new timeslice and determining what could be pruned or removed. In the current work, the application domain deals with DDNs (involving action and utility nodes), a significant amount of work (reachability analysis) is done before hand, and the complete network is reconstructed every time instead of a new timeslice being added and the network updated.

Algorithm 1 Computing Needed (Reachable) States and Observations

```
for all s ∈ States do
  for all a ∈ Actions do
    for all sNew ∈ destinationStates(s,a) do
      s.nextStates[0] ← s.nextStates[0]+ sNew
    end for
    for all o ∈ O(s,a) do
      s.nextObservations[0] ← s.nextObservations[0]
      + o
    end for
  end for
end for
for all s ∈ States do
  for i ← 1,2,... n-1 do {n is the number of decisions}
    for all sLast ∈ s.nextStates[i-1] do
      for all sNew ∈ sLast.nextStates[0] do
        s.nextStates[i] ← s.nextStates[i] + sNew
      end for
      for all o ∈ sLast.nextObservations[0] do
        s.nextObservations[i] ← s.nextObservations[i]
        + o
      end for
    end for
  end for
end for
```

This algorithm is performed before execution so that an agent at runtime only needs to consider the set of reachable states and does not need to compute such states. The structure of the DDN (which variables and edges are present) is also known in advance and the structural representation does not change as the agent is executing. At runtime, the agent only needs to set the possible states in each variable and fill in the conditional and prior probability tables. Given a description for each timeslice of the states necessary from a given start state, an agent can dynamically build a DDN representing the needed state space using Algorithm 2.

Given a current belief state, the agent has beliefs that it currently is in some subset of the possible states of the world. This subset could range from the entire set of states (if it has no knowledge about its state in the world) to a single state (if it knows exactly the state it is in). The DDN that the agent is constructing needs to represent all states the agent could possibly be in now and that it could be in after a finite number of future timesteps. Thus, the actual set of states and observations represented in each timeslice of the DDN is the union of the states and observations that can be reached in that timeslice from any of the states which the agent currently believes it could be in (those states for which the agent has a belief greater than 0). The reachable states from a single current state have already been defined from the previous reachability analysis process shown in Algorithm 1, so the agent solely needs to determine their union. The agent then needs to compute all possible sets of table indices (valid parent state to child state relationships) that are needed for each timeslice and add the appropriate entries to the appropriate conditional probability tables. Probabilities for transitions and observations for each of these states can be found in the transition and probability tables provided by the designer for the domain.

Algorithm 2 Runtime Construction of a Dynamic Decision Network

```
for all n ∈ decisionNetwork.nodes do
  if n.type == Chance then
    possibleConfigurations ← {allCombinations(p, s) |
    p ∈ n.parents ∧ p.type ≠ Action ∧ s ∈ p.states ∧
    P(s) > 0 } {allCombinations(p,s) generates the un-
    factored combination of states from a set of factore d
    variables; in this case, only those combinations with
    probability > 0}
    for all pc ∈ possibleConfigurations do
      n.neededStates ← n.neededStates + precomputed-
      Needed(n, pc)
    end for
  end if
  if n.type == Action then
    for all a ∈ Actions do
      n.neededStates ← n.neededStates + a
    end for
  end if
end for
for all n ∈ decisionNetwork.nodes do
  if n.type == Chance then
    possibleConfigurations ← { allCombinations(p, s) |
    p ∈ n.parents ∧ s ∈ p.states ∧ P(s) > 0 }
    for all pc ∈ possibleConfigurations do
      for all s ∈ n.neededStates do
        n.probabilityTable.add(precomputed(P(pc,s)))
      end for
    end for
  end if
  if n.type == Utility then
    possibleConfigurations ← { allCombinations(p, s) |
    p ∈ n.parents ∧ s ∈ p.states ∧ P(s) > 0 }
    for all pc ∈ possibleConfigurations do
      n.utilityTable.add(precomputed(U(pc)) *
      n.discount)
    end for
  end if
end for
```

General Architecture

Agents developed with the implemented agent architecture act under a traditional sense-plan-act cycle control mechanism. In this control mechanism, any observation an agent makes is used to update the agents belief state. After updat-

ing its beliefs, the agent reasons about the appropriate action to take given it's modified view of the world. The chosen action is then executed and effects of the action in the domain, as well as natural changes occuring in the domain itself, are sensed in the next sensory cycle. Figure 1 (on the next page) is a graphical representation of the local agent components and their interaction during a control cycle.

Bayesian networks are used in the developed architecture for both belief management and planning. Update networks allow the agent to update its beliefs given its previous beliefs and the observation it has just received. This type of network represents state variables for two timesteps, an observation variable attached to the second timeslice, and an action variable. The agents current priors, recently taken action, and recently received observation are mapped onto the appropriate variables and this evidence is propagated through the network to obtain updated beliefs. These networks are not currently dynamically constructed, but they easily could be.

Dynamic decision networks are used to determine the next best action to perform. A dynamic decision network for planning in the developed framework consists of 1) variables representing the state of the world and observations across multiple timesteps, 2) action nodes between each timestep representing the available actions at each timestep, and 3) utility nodes for representing the cost of performing actions and the utility of being in specific states. The agents current beliefs about the true state of the world are dynamically set in the variables in the first timestep of the dynamic decision network. A single maximum utility (under the contraints of the network used) action is then obtained from the DDN and executed. The current implementation only allows for simulated performance of actions by an executive mechanism, but could easily be expanded to perfom actual actions.

Lightweight Implementation Details

The decision networks and algorithms used in planning are developed on top of the Smile API (Druzdzel 1999). This API provides the basic decision-network implementations and algorithms. Microsoft eMbedded Visual C++ was used as the language to implement all higher level components of the agent architecture, as this language was necessary for implementation on the Pocket PC architecture. A domain simulator, also written in Microsoft eMbedded Visual C++, also runs on the device. Its role is to maintain the state of the domain the agent is in, simulate the effect of the agents actions on the domain, and provide observations as feedback when appropriate.

The current PDA implementation consists of a 705 kilobyte executable which can be loaded and executed from an SD memory card. For an example domain of 300 states (the LargeCheeseTaxi domain (Turkett 2004)), auxiliary external files, such as configuration files and files that describe future states reachable from a given state, require approximately 5 megabytes of SD memory card space. Execution of planning runs from start to goal state in the LargeCheeseTaxi domain requires approximately 7 megabytes of memory.

Results

Previous work (Turkett 2004) has demonstrated the effectiveness of dynamic DDN construction in several domains. The results reported in that work and summarized here are from execution an Sun Solaris workstation, using an implementation of the architecture developed in Java and using the Hugin Bayesian Network API. For tests in each domain of interest, the architecture was executed with and without support for dynamic DDN construction. Results were averaged over multiple runs, with each run consisting of the agent executing from a no-knowledge initial state and continuing until a goal state had been reached. Dynamic of the domain were simulated as per the domain model. To provide a summary of the previous work (full execution details can be found in (Turkett 2004)), average per action planning time (a measure of responsiveness) dropped from 4671 milleseconds without dynamic DDN construction to 135 milleseconds with dynamic DDN construction in the 300 state LargeCheeseTaxi domain. The 135ms used for planning with dynamic construction consisted of an average of 56ms for planning and 79ms for DDN construction. Results from other domains include a drop from 249.2 milliseconds without dynamic construction to 174.8 milliseconds with construction (150ms for planning, 25ms for DDN construction) for the 60 state Hallway (Littman, Cassandra, & Kaelbling 1995) domain and from 108.05 seconds to 2.52 seconds (0.6s for planning, 1.9s for DDN construction) in the 870 state RobotTag (Pineau, Gordon, & Thrun 2003) domain.

These three domains have characterists which directly affect the applicability of the dynamic DDN construction approach. The LargeCheeseTaxi domain has informative observations and accurate navigation movements, so an agent acting in this type of domain can quickly localize its position and reduce uncertainty. Accordingly, a planning approach that exploits exact state knowledge, such as the dynamic construction approach shown here, will be able to improve perfomance in such a domain. The Robot Planning domain is similar, as the agent can exploit knowledge of its own location and the ability to gain some insight into the location of its opponent (the domain model indicates the the opponent always moves away from the agent). Finally, the Hallway domain has both noisy observations and significant action uncertainty. Accordingly, the agent has trouble with localization and dynamic construction does not provide significant speedups.

More recent initial results provide insights into whether or not dynamic DDN construction is enabling to support execution in POMDP-type domains on a PDA architecture. The results provided for the PDA device are from execution on an HP iPAQ 4150 with 64MB of RAM and a 400MHz Intel XScale processor. The source code was compiled by eMbedded Visual C++ 4.0 in ARMV4 Release Mode with the /Oxt optimization setting. The SMILE Bayesian Network API (Druzdzel 1999) was used on the PDA device. Reported planning times were measured by adding *GetTickCount()* MS API function calls before and after the call to the plan generator. This function returns time in milliseconds.

Initial results from execution in the 300-state

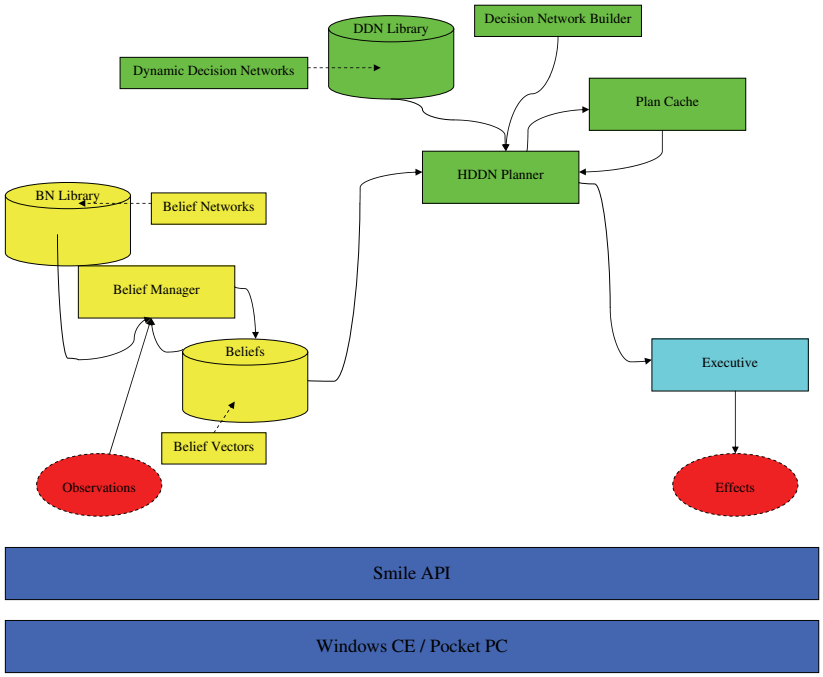

Figure 1: An overview representation of the implemented agent planning and control architecture.

LargeCheeseTaxi domain on the PDA indicate an average per-action planning time over 5 runs without dynamic DDN construction of 111.64 seconds (sd=5.13 seconds). When dynamic construction is turned on, the average per-action planning time over ten runs is 7.1 seconds (sd=0.65 seconds).

In this domain, there were four broad classes of actions that contributed differently to the total running time. The initial state, where the agent's beliefs are distributed uniformly across 30 states (unknown start location, known destination location), has a significant amount of uncertainty and is a significant contributor to total runtime. Without dynamic construction, planning for the initial state took on average 121.22 seconds (sd=0.05 seconds) while with dynamic construction, it took approximately 34.63 seconds (sd=0.06 seconds). Another expensive state was just after picking up a passenger in the domain, wherein the agent knows exactly where it is, but does not know the destination (where the passenger intends to be dropped off). Without dynamic construction, planning requires 145.77 seconds on average (sd=1.55 seconds), while with dynamic construction planning requires 28.09 seconds (sd=2.18 seconds).

The best speedups occurred during those periods where the agent had gathered useful information about it's environment and reduced it's uncertainty. On all navigation type moves, the planning time dropped from 132.03 seconds (sd=12.91 seconds) to 3.36 seconds (sd=1.19 seconds). Finally, the decision to let the passenger out required 27.08 seconds (sd=0.03 seconds) without dynamic construction and 0.6 seconds (sd = 0.09 seconds) with dynamic construction.. The decision to let the passenger into the taxi, which is essentially a fully observable state, took effectively identical times, with or without dynamic construction, of 0.5 seconds. In all cases except for the post-pickup timepoint, the construction phase was cheaper than the cost of planning. This suggests that more work for construction could be pushed into the system to further reduce planning costs. Even though this is only a small set of tests, the DDN construction algorithms do appear effective in significantly reducing the runtime required for planning episodes. Potential areas for improvement include pre-computing or caching actions for states with high uncertainty (i.e. if the agent always starts with a uniform distribution over states, that belief state could be solved once and the best action stored) and removing very low probability states instead of just zero probability states. The removal of low probability states would invalidate the quality preserving characteristics of the current algorithms.

Conclusion

This work introduced and provided an initial examination of the effectiveness of dynamic DDN construction algorithms implemented as part of a lightweight planning architecture. Having already shown promise in a more computationally robust environment, the algorithms, in a small set of tests, demonstrate the ability to reduce average planning time in a PDA-type architecture. By integrating the dynamic DDN construction approach with other algorithmic improvements, such as action caching, planning times should reduce further.

As described in the introduction, several interesting applications of POMDPs to PDAs are emerging. Future research should also consider the issue of the effectiveness

of plan sharing between agents using an architecture such as the one described above. This line of research is motivated by the near-term prospects of a significant number of lightweight devices being constantly in use and within range of each other.

Acknowledgements

The author acknowledges the support of Dr. John R. Rose, Mr. Andrew V. Karode, and the Wake Forest University Science Research Fund in the development of this work.

References

Bohnenberger, T.; Jameson, A.; Kruger, A.; and Butz, A. 2002. Location-aware shopping assistance: Evaluation of a decision-theoretic approach. In *Proceedings of the Fourth International Symposium on Human Computer Interaction with Mobile Devices*.

Cassandra, T. 1999. Pomdps for dummies. Available at: http://www.cs.brown.edu/research/ai/pomdp/tutorial/. Brown University Department of Computer Science.

Cassandra, T. 2003. Pomdps: Who needs them? Available at: http://www.cassandra.org/pomdp/talks/who-needs-pomdps/index.shtml. St. Edwards University Department of Computer Science.

Chang, Y.-H.; Ho, T.; and Kaelbling, L. 2004. Mobilized ad-hoc networks: A reinforcement learning approach. In *Proceedings of 2004 International Conference on Autonomic Computing*.

Druzdzel, M. 1999. Smile: Structural modeling, inference, and learning engine and genie: A development environment for graphical decision-theoretic models (intelligent systems demonstration). In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 902–903.

Forbes, J.; Huang, T.; Kanazawa, K.; and Russell, S. 1995. The batmobile: Towards a bayesian automated taxi. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*.

Gartner. 2006. Gartner says worldwide combined pda and smartphone shipments market grew 57 percent in the first half of 2006. http://www.gartner.com/it/page.jsp?id=496997.

Jensen, F. 2001. *Bayesian Networks and Decision Graphs*. New York, NY: Springer-Verlag.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1995. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, 362–370.

Murray, C.; VanLehn, K.; and Mostow, J. 2004. Looking ahead to select tutorial actions: A decision-theoretic approach. *International Journal of Artificial Intelligence in Education* 14:235–278.

Ngo, L.; Haddaway, P.; Krieger, R.; and Helwig, J. 1997. Efficient temporal probabilistic reasoning via context-sensitive model construction. *Computers in Biology and Medicine* 27(5):453–476.

Nicholson, A., and Brady, M. 1994. Dynamic belief networks for discrete monitoring. *IEEE Transactions on Systems, Man, and Cybernetics* 24(11):1593–1610.

Nicholson, A., and Russell, S. 1993. Techniques for handling inference complexity in dynamic belief networks. Technical Report CS-93-31, Department of Computer Science, Brown University.

Ocana, M.; Bergasa, L. M.; Sotelo, M. A.; and Flores, R. 2005. Indoor robot navigation using a pompdp based on wifi and ultrasound observations. In *Proceedings of the 2005 International Conference on Intelligent Robots and Systems*.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for pomdps. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Provan, G. 1993. Tradeoffs in constructing and evaluating temporal influence diagrams. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*.

Russell, S., and Norvig, P. 2002. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, second edition.

Scerri, P.; Pynadath, D.; and Tambe, M. 2001. Adjustable autonomy in real-world multi-agent environments.

Turkett, W. 2004. Robust multiagent plan generation and execution. Doctoral Dissertation. University of South Carolina Department of Computer Science.

Zhao, Q., and Swami, A. To appear. A decision-theoretic framework for opportunistic spectrum access. *IEEE Wireless Communications Magazine: Special Issue on Cognitive Wireless Networks*.