The Computational Complexity of Boolean and Stochastic Agent Design Problems

Paul E. Dunne
Dept. of Computer Science
University of Liverpool
Liverpool L69 7ZF, UK
ped@csc.liv.ac.uk

Michael Wooldridge
Dept. of Computer Science
University of Liverpool
Liverpool L69 7ZF, UK
mjw@csc.liv.ac.uk

Michael Laurence
Dept. of Computer Science
University of Liverpool
Liverpool L69 7ZF, UK
mikel@csc.liv.ac.uk

ABSTRACT

The *Agent Design* problem involves determining whether or not it is possible to construct an agent capable of satisfying a given task specification in a particular environment. The simplest examples of such specifications are where an agent is required to bring about some goal or where an agent is required to maintain some invariant condition. Both cases can be viewed as identifying a set of states with a *single* propositional variable, x, so that the tasks are specified by formulae x (reach one of the states represented by x) or $\bar{x}$ (avoid all of the states represented by x). In previous work, the complexity of agent design problems expressed by single argument propositional formulae has been systematically investigated for a range of different types of environments. It was shown that the complexity of the problem varies from NL-complete in the best case to undecidable in the worst. In this paper, we extend these previous results by investigating the effect on problem complexity of allowing tasks for agents to be specified as arbitrary propositional formulae of n variables. We show that, in those settings where Agent Design is PSPACE or NP-complete the general problem is "no more difficult", i.e., remains in PSPACE or NP. In contrast, a number of the settings for which there were polynomial-time methods become NP-complete when considering more general task specifications. Finally, we investigate the complexity of *stochastic* agent design problems, where we require an agent that has a "better than evens" chance of success: we show that complexity results for the "guaranteed success" cases carries across to stochastic instances.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, Search; I.2.11 [**Distributed Artificial Intelligence**]: Intelligent Agents; F.1 [**Theory of Computation**]: Computation by Abstract Devices; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Theory, Algorithms

Keywords

Agent Design Problem, Task specifications, Computational Complexity

1. INTRODUCTION

In this paper, we are concerned with the computational complexity of one particular issue that arises in agent-based systems: the *agent design problem.* This problem can be informally stated as follows:

> Given (representations of) an environment and a task to be carried out, does there exist an agent that can be guaranteed to carry out the task in the environment?

Wooldridge and Dunne investigated the computational complexity of the agent design problem in a range of different settings, and for a range of different types of tasks [8, 9, 3]. The two most important types of tasks they considered were *achievement tasks* (where an agent is required to bring about some goal state), and *maintenance tasks* (where an agent is required to maintain some invariant in an environment). The complexity of these problems was shown to vary from NL-complete in the simplest case, to non-recursive in the worst case, depending upon what assumptions are made about the environment. Given "reasonable" representations, the agent design problems for both achievement and maintenance tasks were shown to be PSPACE-complete [8].

In this paper we extend the work of Wooldridge and Dunne by first considering richer task specifications. The idea is to specify tasks as *arbitrary Boolean combinations of achievement and maintenance tasks.* Thus a task specification is a formulae of propositional Boolean logic: atomic propositions denote sets of environment states, positive literals are interpreted as achievement tasks ("bring about one of these states"), and negative literals are interpreted as maintenance tasks ("avoid all these states"). Thus, for example $p \vee \bar{q}$ is a task in which an agent must either bring about one of the states denoted by p or else avoid all the states denoted by q. In addition, we consider variants of the *stochastic* agent design problem. Wooldridge and Dunne only considered agents that were either *guaranteed* to succeed with the task, or *had at least some chance* of success. In a stochastic agent design problem, we ask whether there exists an agent that has a *better than evens* chance of success.

We begin by introducing a formal model of agents, environments, and tasks. We then define Boolean task specifications and introduce several variants of the Agent Design problem for these specifications. We first summarize the results of [8, 9, 3], and then examine how the complexity of the different settings considered in this is affected when more intricate task specification structures pertain. In section 4, we turn our attention to stochastic agent design problems. Throughout the paper, we assume some familiarity with complexity theory [6].

2. AGENTS, ENVIRONMENTS, AND TASKS

In this section, we present an abstract formal model of agents and the environments they occupy; we then use this model to frame the decision problems we study. The systems of interest to us consist of an agent situated in some particular environment; the agent interacts with the environment by performing actions upon it, and the environment responds to these actions with changes in state. It is assumed that the environment may be in any of a finite set $E = \{e_0, e_1, \ldots, e_n\}$ of instantaneous states. Agents are assumed to have a repertoire of possible actions available to them, which transform the state of the environment. Let $Ac = \{\alpha_0, \alpha_1, \ldots, \alpha_k\}$ be the (finite) set of actions. The behaviour of an environment is defined by a *state transformer function*, τ. In our case τ is not simply a function from environment states and actions to sets of environment states, but from *runs* and actions to sets of environment states. This allows for the behaviour of the environment to be dependent on the *history* of the system — the previous states of the environment and previous actions of the agent can play a part in determining how the environment behaves.

DEFINITION 1. *An* environment *is a quadruple $Env = \langle E, e_0, Ac, \tau\rangle$, where E is a finite set of* environment states *with e_0 distinguished as the* initial state*; and Ac is a finite set of* available actions. *Let S_{Env} denote all sequences of the form:*

$$e_0 \cdot \alpha_0 \cdot e_1 \cdot \alpha_1 \cdot e_2 \cdots$$

where e_0 is the initial state, and for each i, $e_i \in E$, $\alpha_i \in Ac$.

Let S^E be the subset of such sequences that end with a state. The state transformer function τ *of the environment is a total mapping*

$$\tau : S^E \times Ac \to \wp(E)$$

We focus on the the set of runs *in the environment. This is the set $R_{Env} = \cup_{k=0}^{\infty} R^{(k)}$, where*

$$\begin{array}{lcl} R^{(0)} & = & \{e_0\} \quad \textit{and} \\ R^{(k+1)} & = & \bigcup_{r\in R^{(k)}} \bigcup_{\{\alpha\in Ac \mid \tau(r,\alpha)\neq\emptyset\}} \{r\cdot\alpha\cdot e \mid e\in\tau(r,\alpha)\} \end{array}$$

An environment is bounded *if $R^{(k)} = \emptyset$ for some k and* unbounded *otherwise. We denote by R^{Ac} the set $\{r \cdot \alpha \mid r \in R_{Env}\}$ so that we subsequently interpret τ as a total mapping, $\tau : R^{Ac} \to \wp(E)$, i.e., as describing the (possibly empty) set of states which may result by performing the action $\alpha \in Ac$ after a run $r \in R_{Env}$.*

A run, r, has terminated *if $\forall \alpha \in Ac$, $\tau(r \cdot \alpha) = \emptyset$. The subset of R_{Env} comprising all terminated runs is denoted T_{Env}. In bounded environments, every run is a prefix of some (set of) terminated runs.*

The length *of a run, $r \in R_{Env}$ is the total number of actions and states occurring in r and is denoted by $|\, r\,|$.*

Unless otherwise stated it is assumed that environments are bounded.

The state transformer function, τ, is encoded in an input instance by a deterministic Turing machine description T_τ with the following characteristics: the input has the form $r\#e$, where $r \in R^{Ac}$, $e \in E$, and $\#$ is a separator symbol. The program T_τ accepts $r\#e$ if and only if $e \in \tau(r)$. The number of moves made by T_τ is bounded by a polynomial, $p(|r|)$. Using T_τ, the set of states $\tau(r)$ can be constructed in $|E|p(|r|)$ steps.

We view *agents* as performing actions upon the environment, thus causing the state of the environment to change. In general, an agent will be attempting to "control" the environment in some way, in order to carry out some task. However, the agent has at best partial control over the environment.

DEFINITION 2. *An* agent*, Ag, in an environment $Env = \langle E, e_0, Ac, \tau\rangle$ is a mapping*

$$Ag : R_{Env} \to Ac \cup \{\otimes\}$$

The symbol $\otimes$ is used to indicate that the agent has finished its operation: an agent invokes this only on terminated runs, $r \in T_{Env}$, an event that is referred to as the agent having no allowable actions. *A* system*, Sys, is a pair $\langle Env, Ag\rangle$ comprising an environment and an agent operating in that environment. A sequence $s \in R_{Env} \cup R^{Ac}$ is called a* possible run of the agent Ag in the environment Env *if*

$$s = e_0 \cdot \alpha_0 \cdot e_1 \cdot \alpha_1 \cdots$$

satisfies

1. *e_0 is the initial state of E, and $\alpha_0 = Ag(e_0)$;*
2. *$\forall k > 0$,*

$$\begin{array}{lcl} e_k & \in & \tau(e_0\cdot\alpha_0\cdot e_1\cdot\alpha_1\cdots e_{k-1}\cdot\alpha_{k-1}) \textit{ where} \\ \alpha_k & = & Ag(e_0\cdot\alpha_0\cdot e_1\cdot\alpha_1\cdots e_k) \end{array}$$

It should be noted that, in general, the state transformer function τ (with domain R^{Ac}) is *non-deterministic*, whereas agents are *deterministic*.

An agent may have a number of different possible runs in any given environment. We will denote by $R(Ag, Env)$ the set of possible runs of agent Ag in environment Env and by $T(Ag, Env)$ the subset of these that are terminated, i.e., belong to T_{Env}. An agent, Ag, must define some allowable action in Ac, for every run in $R(Ag, Env) \setminus T_{Env}$, i.e., agents may not choose to halt arbitrarily.

We concentrate on the behaviour of agents which have some bound placed on the number of actions performed. More precisely, given $t : \mathbb{N} \to \mathbb{N}$, the set of *$t(n)$-critical runs* of an agent Ag in the environment Env is the set $C^{t(n)}(Ag, Env)$ defined by those runs of the agent in which exactly $t(n)$ actions have been performed or which have terminated after at most $t(n) - 1$ actions. Unless otherwise stated the bounding function $t(n) = n$ (with $n = |E \times Ac|$) is used.

3. BOOLEAN TASK SPECIFICATIONS

We build agents in order to carry out *tasks* for us. The task to be carried out must be *specified* by us. An obvious question is how to specify these tasks. In this paper, we will be concerned with *predicate task specifications*. Such specifications take the form of a predicate over runs, i.e., a condition that runs either satisfy or fail to satisfy. We use Ψ to denote a predicate specification, and write $\Psi(r)$ to indicate that run $r \in R$ satisfies Ψ. Intuitively, $\Psi(r)$ means that the task specified by Ψ is successfully accomplished on run r. We are concerned in this paper with the *representation* of Ψ as a logical formula.

A *task environment* is a pair $\langle Env, \Psi \rangle$, where Env is an environment, and $\Psi : R \to \{0, 1\}$ is a predicate over runs. A task environment thus specifies the properties of the system the agent will inhabit (i.e., the environment Env), and also the criteria by which an agent will be judged to have either failed or succeeded (i.e., the specification Ψ).

One final consideration may, informally, be phrased as "what limits (if any) do we wish to place on how long an agent may take to succeed?" In this context we consider two basic decision problem formulations:

a) Is there an agent that *eventually* succeeds in its task?

b) Is there an agent the succeeds after *at most* some number of actions?

We now define our Boolean task specification language and formulate the decision problems studied.

Let $X_n = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables. A propositional formula $\Psi(X_n)$ over X_n is inductively defined by the following rules:

1. $\Psi(X_n)$ consists of single literal (x or $\bar{x}$ where $x \in X_n$).
2. $\Psi(X_n) = \Phi_1(X_n)\theta\Phi_2(X_n)$, where $\theta \in \{\vee, \wedge\}$ and Φ_1, Φ_2 are propositional formulae.

$|\Psi(X_n)|$ denotes the total number of occurrences of *literals* in Ψ, and f_Ψ the Boolean logic function (of n variables) represented by Ψ. We say that $\Psi(X_n)$ is *non-trivial* if f_Ψ is not equivalent to a (Boolean) constant.

DEFINITION 3. *Let $Env = \langle E, e_0, Ac, \tau \rangle$ be an environment with state set E and actions Ac. Let $S = \langle E_1, \ldots, E_n \rangle$ be an ordered collection of* pairwise disjoint *subsets of E (note that S does* not *have to be a partition of E) and $\Psi(X_n)$ be a non-trivial formula.*

If $r \in R_{Env}$ is a run, then the instantiation, $\beta(r) = \langle b_1, b_2, \ldots, b_n \rangle$ of Boolean values to X_n induced by r *is defined by:*

$$b_i = \begin{cases} 1 & \textit{if } r \textit{ contains some state } e \in E_i \\ 0 & \textit{if } r \textit{ does not contain any state } e \in E_i. \end{cases}$$

A run, r, succeeds with respect to Ψ *if $f_\Psi(\beta(r)) = 1$.*

A task specification *for an agent in an environment, Env consists of a pair $\langle S, \Psi \rangle$ where S is an ordered collection of n pairwise disjoint subsets of E and Ψ is a non-trivial formula of n variables.*

An agent, Ag, satisfies the task specification $\langle S, \Psi \rangle$ if and only if

$$\forall r \in T(Ag, Env) \quad \textit{we have } f_\Psi(\beta(r)) = 1$$

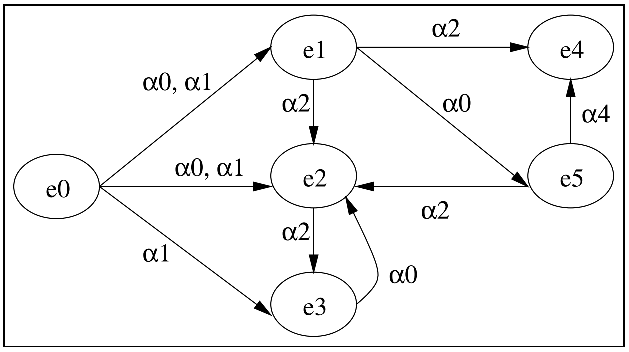

Figure 1: The state transitions of an example environment: Arcs between environment states are labelled with the sets of actions corresponding to transitions. Note that this environment is *history dependent*, because agents are not allowed to perform the same action twice. So, for example, if the agent reached state e_2 by performing α_0 then α_2, it would not be able to perform α_2 again in order to reach e_3.

Consider the environment whose state transformer function is illustrated by the graph in Figure 1. In this environment, an agent has just four available actions (α_1 to α_4 respectively), and the environment can be in any of six states (e_0 to e_5). History dependence in this environment arises because the agent is not allowed to execute the same action twice. Arcs between states in Figure 1 are labelled with the actions that cause the state transitions — note that the environment is non-deterministic. Suppose that we allow three primitive proposition letters to be used: p_1, p_2, p_3, where p_1 corresponds to environment states $\{e_2\}$, p_2 corresponds to $\{e_3\}$, and p_3 corresponds to $\{e_1\}$. Now consider the following task specifications:

- p_1

  This is an achievement task with goal states $\{e_2\}$. Since p_1 corresponds to $\{e_2\}$, an agent can reliably achieve p_1 by performing α_1, the result of which will be either e_1, e_2, or e_3. If e_1 results, the agent can perform α_0 to take it to e_5 and then α_2 to take it to e_2. If e_3 results, it can simply perform α_0.

- $\bar{p_1}$

  This is essentially a maintenance task with bad states $\{e_2\}$. Since the agent must perform either α_0 or α_1, no agent can guarantee to avoid e_2.

- $p_1 \vee p_2$

  Since there is an agent that can be guaranteed to succeed with task p_1, there is also an agent that can be guaranteed to succeed with task $p_1 \vee p_2$.

- $p_1 \wedge p_2$

  This task involves achieving *both* e_2 and e_3. There is no agent that can be guaranteed to succeed with this task.

- $p_1 \wedge (p_2 \vee p_3)$

This task involves achieving e_2 and either e_3 or e_1. There exists an agent that can successfully achieve this task.

- $(p_1 \wedge \bar{p_2}) \vee (p_2 \wedge \bar{p_3})$

  This task involves either achieving e_2 and not e_3 or else achieving e_3 and not e_1. Since there exists an agent that can achieve e_2 and not e_3, there exists an agent that can succeed with this task.

We consider two decision classes of agent design problem: design tasks with *implicit* specification and design tasks with *explicit* specification.

DEFINITION 4. *Let Ψ be a non-trivial propositional formula of n variables. The decision problem* finite *Ψ-Design (Ψ-FD) takes as an instance: an environment Env, and S, an ordered collection of n pairwise-disjoint subsets of E. Ψ-$FD(\langle Env, S\rangle)$ returns "true" if and only if*

$$\exists Ag\ \forall r \in C^{|E\times Ac|}(Ag, Env) \quad : \quad f_\Psi(\beta(r)) = 1$$

It is important to note that Ψ is *not* part of the instance: each specific Ψ defines a design task for agents in Env. In contrast, we have

DEFINITION 5. *The decision problem* Finite Agent Design *with* Explicit Task Specification *(FED) takes as an instance: a* non-trivial *propositional formula $\Psi(X_n)$, an environment Env, and S, an ordered collection of n pairwise-disjoint subsets of E. $FED(\langle \Psi, Env, S\rangle)$ returns "true" if and only if*

$$\exists Ag\ \forall r \in C^{|E\times Ac|}(Ag, Env) \quad : \quad f_\Psi(\beta(r)) = 1.$$

We employ the following shorthand in order to simplify subsequent notation. For $Q \in \{\Psi\text{-}FD, FED\}$ and $X \in \{det, non\text{-}det\}$ we denote by Q_X the decision problem Q with τ having the property X when given as an instance. For example, FED_{det} denotes the decision problem *Finite Agent Design with Explicit Task Specification for deterministic τ.*

3.1 Computational Complexity Results

Wooldridge and Dunne [8, 9, 3] considered the special cases $\Psi(x) = x$ and $\Psi(x) = \bar{x}$. In [3] the setting in which no restriction is placed on the number of actions that a successful agent is allowed, was also considered, i.e., the problem $\Psi(x)$-*Design* whose instances are pairs $\langle Env, S\rangle$ with S a subset of E concerned with the question of whether an agent exists that is guaranteed to reach at least one (respectively, avoid all) of the states in S.

Theorem 1 summarises the results proved in [8, 9, 3] for these special cases.

THEOREM 1. *Let $\Psi(x)$-Design$^\infty$ (resp., $\Psi(x)$-Design$^{<\infty}$) denote the cases in which unbounded environments may (resp., may not) occur as instances:*

a) x-Design$^\infty$ is r.e. but not recursive.

b) $\bar{x}$-Design$^\infty$ is not r.e.

c) $\Psi(x)$-Design$^{<\infty}$ is recursive.

For the case of finite *design problems.*

d) $\forall X \in \{det, non\text{-}det\}$ x-$FD_X \equiv_{\log} \bar{x}$-$FD_X$.

e) $\Psi(x)$-$FD_{non\text{-}det}$ is PSPACE*-complete.*

f) $\Psi(x)$-FD_{det} is NP*-complete.*

Finally, if τ is history independent, *i.e. the next environment state is dependent on the current state and chosen action only (not on the sequence of states and actions by which it was reached).*

g) $\Psi(x)$-$FD_{non\text{-}det}$ is NL*-complete.*

h) $\Psi(x)$-FD_{det} is P*-complete.*

Our principal concern in this paper is to generalize Theorem 1 to arbitrary n variable propositional functions.

To begin we note the following generalisation of Theorem 1 (a,b), in the case when the definition of Ψ-*Design* is modified to apply in unbounded environments with no restriction on the number of actions an agent can perform.

THEOREM 2. *Let Ψ-Design$^\infty$ denote the decision problem Ψ-Design with instances of* unbounded *environments allowed.*

a) Ψ-Design$^\infty$ is recursive if and only if Ψ is trivial.

b) Ψ-Design$^\infty$ is recursively enumerable if and only if f_Ψ is monotone.

We now show that the decision problems Ψ-FD_X and FED_X are polynomial-time equivalent to the decision problems x-FD_X and $\bar{x}$-FD_X, irrespective of the specific non-trivial Ψ concerned.

THEOREM 3. *$\forall X \in \{det, non\text{-}det\}$, If Ψ is* any *non-trivial formula, Ψ-$FD_X \equiv_p x$-FD_X; $FED_X \equiv_p x$-FD_X.*

PROOF. We omit the full technical details and simply outline the construction establishing Ψ-$FD_X \leq_p x$-FD_X, i.e., that Ψ-FD_X is "no harder" than x-FD_X for any non-trivial Ψ. Given an instance $\langle Env, S\rangle$ of Ψ-FD_X, form an instance $\langle Env_A, G\rangle$ of x-FD_X by adding a new state $\{accept\}$ and action $\{test\}$. The set G contains the single state $\{accept\}$. We then modify the transition function for the instance of Ψ-FD_X so that if a run r has $f_\Psi(\beta(r)) = 1$ then $\tau_A(r \cdot test) = \{accept\}$. In this way any instance of Ψ-FD_X would have true returned if and only if the constructed instance also has true returned. $\square$

COROLLARY 1. *For any non-trivial formula Ψ,*

a) Ψ-$FD_{non\text{-}det}$ is PSPACE*-complete.*

b) Ψ-FD_{det} is NP*-complete.*

PROOF. Immediate from Theorem 1(e,f) and Theorem 3. $\square$

Theorems 2 and 3 extend the results of Theorem 1 (a,b,e,f) as regards environments in which the state transition function is *history-dependent*. For the remainder of this paper we deal with the remaining cases involving *history independent* τ. In a *history-independent* environment, the transition function, τ, acts upon the current state and action to determine the next state of the environment. In such environments τ presented as a directed graph, $H(V, A)$, whose vertices, V, are labelled with environment states and in which there is an edge labelled $\alpha \in Ac$ from e_i to e_j if $e_j \in \tau(e_i, \alpha)$. In retaining the assumption that environments are bounded, it follows that $H(V, A)$ is an *acyclic* directed graph.

For the case of arbitrary non-trivial formulae, Ψ, we have:

THEOREM 4. *If τ is history-independent, then:*

a) $\forall\Psi$, Ψ-$FD_{det} \in$ P.

b) $\forall\Psi$, Ψ-$FD_{non\text{-}det} \in$ NP.

c) FED_{det} *is* NP*-complete*[1].

PROOF. For part (a) the fact that the specification Ψ is implicit is significant. Since our concern is simply to exhibit the existence of *a* polynomial-time algorithm, we do not attempt to minimise the degree of the polynomial used.

Let $\langle Env, S\rangle$ be an instance of $\Psi(X_k)$-FD for some non-trivial propositional formula $\Psi(X_k)$. To begin we observe the following:

a) The size of an instance of $\Psi(X_k)$-FD, is *independent* of k and $|\Psi|$. Thus, $|\Psi(X_k)|$ enters into the run-time of any algorithm recognising $\Psi(X_k)$-FD only as a *constant* factor.

b) It follows from (a) that, without loss of generality, $\Psi(X_k)$ may be considered to be represented in disjunctive normal form (DNF), as

$$\Psi(X_k) = \bigvee_{i=1}^{r} \bigwedge_{y\in m_i} y \quad m_i \subseteq \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_k\}$$

For an arbitrary term $m = \wedge_{j=1}^{k} y_j$ in the DNF form of $\Psi(X_k)$, $m^{(-)}$ denotes the set of variables from X_k that occur as negated literals in m, and similarly $m^{(+)}$ those occurring as positive literals. An agent that satisfies the specification given by m *alone*, must avoid every state $e \in S_j$ for which $x_j \in m^{(-)}$ and attain at least one state in S_j for each $x_j \in m^{(+)}$. Furthermore an agent satisfying $\Psi(X_k)$ exists for $\langle Env, S\rangle$ if and only if there is some term m for which an agent realising m exists in $\langle Env, S\rangle$. It should be noted that this property does not, in general, hold for *non-deterministic* τ.

An algorithm to decide $\Psi(X_k)$-FD proceeds by considering each m in turn. First those vertices of $H(V, A)$ corresponding to states in S_j for which $x_j \in m^{(-)}$ are deleted, together with their incident edges. Let $D_m(W, B)$ denote the resulting graph. Without loss of generality let $m^{(+)} = \{x_1, \ldots x_p\}$, where $p \leq k$. An agent satisfying the specification $\wedge_{i=1}^{p} x_i$ is defined from any directed *path* P in $D_m(W, B)$ that starts with the vertex labelled e_0 and for each i contains at least one vertex labelled with some $e \in S_i$. There are at most

$$\sum_{\langle\pi_1,\pi_2,\ldots,\pi_p\rangle} \prod_{j=1}^{p} |S_{\pi_j}| \;\leq |E|^k k!$$

(where the summation ranges over all possible orderings, of $\langle x_1, \ldots, x_p\rangle$) ways of forming such a path in D_m. Each can be tested in $O(|E|)$ steps. Since the number of terms in the DNF representation $\Psi(X_k)$ is $\leq 2^k$, any instance $\langle Env, S\rangle$ of $\Psi(X_k)$-FD within a history-independent, deterministic environment, can be tested in $O(|E|^{k+1})$ steps

For part (b), use an algorithm which first guesses the action α_i to use in each state e_i. These guesses induce a (directed, acyclic) subgraph of $H(V, A)$ which defines a successful agent if every path from e_0 to a sink satisfies the specification Ψ. This can be tested using a simple depth-first search.

For part (c) we first show that instances $\langle\Psi(X_n), Env, S\rangle$ of FED_{det} having a history independent transition function can be decided in NP. An NP algorithm simply guesses which action $\alpha \in Ac$ (if any) to perform in each state. In the graph $H(V, A)$ these choices determine a single path (since τ is deterministic). Evaluating $f_{\Psi}(X_n)$ for this path is easily done in polynomially many steps in $|\Psi| + |Env|$.

To show that FED_{det} for history independent τ is NP-hard we use a reduction from *Paths with Forbidden Pairs* restricted to directed acyclic graphs — PFP-DAG. An instance of PFP-DAG consists of a directed acyclic graph $G(V, A)$, two specified vertices $s, t \in V$, and a collection

$$C = \{\langle a_1, b_1\rangle,\ \langle a_2, b_2\rangle,\ \ldots,\ \langle a_n, b_n\rangle\}$$

of pairs of vertices in V. An instance is accepted if there is a path from s to t in $G(V, A)$ that contains *at most one* vertex from each pair in C.

Given an instance $\langle G(V, A), s, t, C\rangle$ of PFP-DAG we construct, in time polynomial in $|A| + |C|$, an instance

$$\langle\Psi(X_n)_{G,C}, Env, S\rangle$$

of FED_{det} for which τ is history independent and there is an agent satisfying $\Psi(X_n)$ in Env if and only if there is a path from s to t in $G(V, A)$ that contains at most one vertex from each pair in C.

We set $Env = \langle E, e_0, Ac, \tau\rangle$ where $E = V \cup \{e_t\}$; $Ac = \{\alpha_1, \ldots, \alpha_d, \alpha_{d+1}\}$, where d is the maximum out-degree of $G(V, A)$. For v with out-degree k in $G(V, A)$ and each v_i such that $\langle v, v_i\rangle$ is an edge of $G(V, A)$ we set $\tau(v, \alpha_i)$ to be v_i. An additional action influences only the vertex t: for this we have a single action, α_{d+1} entering the final state e_t. Finally the initial state, $e_0 \in E$ is set to the state corresponding to vertex s of $G(V, A)$. The graph, $H(V \cup \{e_t\}, A)$ defined is identical to $G(V, A)$ (with the addition of the $\langle t, e_t\rangle$ edge), and has at most one out-going edge from any vertex labelled with a given action.

The formula Ψ constructed for the instance uses $|\cup_{i=1}^{n} \{a_i, b_i\}| + 1$ variables, where n is the number of pairs in C Let $\langle A_n, B_n, z\rangle$ denote the resulting variables (where it may be the case that $|A_n|, |B_n| < n$ or $A_n \cap B_n \neq \emptyset$). With these,

$$\Psi(A_n, B_n, z) \;=\; z \;\wedge\; \bigwedge_{i=1}^{n} (\bar{a}_i \vee \bar{b}_i)$$

The final stage of the construction is to define the set of (at most) $2n + 1$ pairwise disjoint subsets of E that form S. Let m be the total number of distinct vertices referred to in C (so that Ψ depends on exactly $m + 1$ variables). Then: $S_i = \{a_i\}$ (if i is odd and the vertex a_i does not occur in any pair $C_j \in C$ with $j < i$); $S_i = \{b_i\}$ (if i is even and the vertex b_i does not occur in any pair $C_j \in C$ with $j < i$); and

[1]To be strictly accurate, we define $\langle\Psi_n\rangle_{n\geq 1}$, a sequence of non-trivial n-variable propositional formulae Ψ_n, such that the decision problem $FED_{det}(\Psi_n, Env, S_n)$ is NP-complete.

$S_{m+1} = \{e_t\}$. The construction ensures that any state (i.e., vertex) occurs in at most one S_i set.

We claim that an agent satisfying the specification $\Psi(X_{m+1})$ in the environment just defined if and only if there is a directed path from s to t containing at most one vertex from each pair in C for the instance of PFP-DAG.

Suppose such an agent exists. Its actions define a directed path from the initial state (s) to some final state. This path must contain the state corresponding to e_t (since this is the $m+1$'st variable, z, used in defining Ψ). Consider the corresponding path from s in $G(V, A)$. This path must contain the vertex t since the state e_t reached by the agent can only be accessed from the state corresponding to t in the environment. Thus the agent describes some path from s to t in $G(V, A)$. This path cannot contains both vertices from some pair $\langle a_i, b_i \rangle$: for if x_i, x_j were the propositional variables associated with these, then $\Psi(X_{m+1})$ would be false under the instantiation induced by the agent due to the presence of a clause equivalent to $\bar{x}_i \vee \bar{x}_j$. It follows that from an agent satisfying the specification we can find a path from s to t in $G(V, A)$ meeting the forbidden pair requirements.

On the other hand, suppose $G(V, A)$ contains a path from s to t containing at most one vertex from each pair in C. From this path we can construct a sequence of actions for each state, (the action corresponding to the out-going edge from each vertex on the path). The agent thereby defined reaches the state t and carrying out the action α_{d+1} completes the transition into the required final state e_t. $\Box$

4. STOCHASTIC AGENT DESIGN

So far, we have been considering a rather pessimistic notion of success with respect to tasks: an agent must be guaranteed to satisfy its task on *all* runs, otherwise it is deemed to be no good. At the other extreme, a rather too *optimistic* notion of success was studied in [9], where an agent was considered acceptable if it succeeded to accomplish its designated task on *at least one* run. It was shown in [9] that the optimistic agent design problem was "easier" than pessimistic agent design: versions of the agent design problem that are PSPACE-complete are when considering pessimistic agent design are "merely" NP-complete in the optimistic case.

Whereas the strict, pessimistic notion of success that we have discussed so far in this paper is perhaps too strong, the optimistic agent design problem is too weak. In practice, we would probably want to know whether the agent could be *expected* to succeed — whether it has a better than even chance of succeeding. We refer to this problem — of determining whether or not an agent has a better than even chance of succeeding with respect to its task in a particular environment — as *stochastic agent design*. The critical assumption that we must make in order to study this problem is that all (immediate) outcomes of an action are equiprobable.

> *Stochastic agent design*:
> *Given*: Achievement agent design task with environment $Env = \langle E, e_0, \tau \rangle$ and goal states $S \subseteq E$.
> *Answer*: "yes" if there is an agent that succeeds with this task with probability greater than 1/2, "no" otherwise.

It should be noted that an agent having a probability of success greater than 1/2 is *not equivalent* to an agent succeeding on *the majority* of its runs. Too see this, consider an environment in which there is a single action α which leads to 2 states from the initial state; one of these has no available actions; the other (for some value k) leads to a binary computation tree of height k in which all but one of the states at the leaves is a state in S: the unique agent in this environment has $2^k + 1$ runs of which the majority $(2^k - 2)$ succeed. The probability of success, however, is less than 1/2: the state with further available actions is chosen with probability 1/2, but with non-zero probability a run from this state does not succeed.

For an environment Env and goal states S, $FSD(\langle Env, S \rangle)$ denotes an instance of stochastic finite achievement design; in addition, we denote the problem in general by FSD_X, where $X \in \{det, non\text{-}det\}$ is interpreted as above.

THEOREM 5. *$FSD_{non\text{-}det}$ is* PSPACE*-complete.*

PROOF. Let $\langle Env, S \rangle$ be an instance of $FSD_{non\text{-}det}$ and $n = |E \times Ac|$. To show that $FSD_{non\text{-}det} \in$ PSPACE, we consider a function – Ext – that maps possible runs (ending in a state) from Env to *rational* values, p, in the range $[0, 1]$. This function $Ext(r \cdot e)$ has value 1 if $e \in S$; 0 if there are no available actions for $r \cdot e$ or if the length of $r \cdot e$ exceeds n. In all other case, $Ext(r \cdot e)$ is:

$$\max_{\alpha : \tau(r \cdot e \cdot \alpha) \neq \emptyset} \frac{1}{|\tau(r \cdot e \cdot \alpha)|} \sum_{d \in \tau(r \cdot e \cdot \alpha)} Ext(r \cdot e \cdot \alpha \cdot d)$$

The function $Ext(r \cdot e)$ is the (maximum) probability that an agent continuing a run $r \cdot e$ attains one of the goal states. It follows that $\langle Env, S \rangle$ is a positive instance of $FSD_{non\text{-}det}$ if and only if $Ext(e_0) > 0.5$ and so it suffices to show that the value $Ext(e_0)$ can be computed in PSPACE. In order to simplify the algorithm presentation we use a *non-deterministic* machine to "guess" the maximising action for its input run: certainly there is (at least) one pattern of such guesses which will correctly compute $Ext(e_0)$. The algorithm is recursive and given a run $r \cdot e$ proceeds as follows:

1. If $e \in S$ return 1 else

- if $\forall \alpha$, $\tau(r \cdot e \cdot \alpha) = \emptyset$ return 0 else
- if $|r \cdot e| > n$ return 0 else

2. Non-determininistically choose an available action α.
   2.1 $count := 0$.
   2.2 for each $d \in \tau(r \cdot e \cdot \alpha)$,
   $$count := count + \frac{Ext(r \cdot e \cdot \alpha \cdot d)}{|\tau(r \cdot e \cdot \alpha)|}$$
   2.3 return $count$

It remains to observe that this computation can be carried out in NPSPACE: only rational values are involved, and using representation of these by two integer values (i.e. numerator and denominator), the value of the denominator never exceeds n^n and so can be stored in $O(n \log n)$ bits. The depth of the recursion is $O(n)$ and so the entire procedure can be realised using only polynomial (in n) space. Since NPSPACE = PSPACE the membership proof is completed.

To prove PSPACE-hardness, we show how the problem of *stochastic satisfiability* (SSAT) can be reduced to $FSD_{non\text{-}det}$. An instance of SSAT is given by a formula with the form:

$$\exists x_1.\mathsf{R}x_2.\exists x_3.\mathsf{R}x_4 \cdots Qx_n.\ \varphi(x_1,\ldots,x_n) \tag{1}$$

where:

- each x_i is a boolean variable (it does *not* need to be a collection of variables);
- R is a "random" quantifier, with the intended interpretation "with some randomly selected value"; and
- Q is $\exists$ if n is odd, and R otherwise.

The goal of SSAT is to determine whether there is a strategy for assigning values to existentially quantified variables that makes (1) true with probability great than $\frac{1}{2}$, i.e., if

$$\exists x_1.\mathsf{R}x_2.\exists x_3.\mathsf{R}x_4 \cdots Qx_n.\ \mathsf{prob}[\varphi(x_1,\ldots,x_n) = \top] > \frac{1}{2}$$

We give an outline of the reduction first. To reduce an instance (1) to $FSD_{non\text{-}det}$, we proceed as follows. The idea is to create an environment which forces the agent and environment to take it in turns, starting with the agent, to assign values (truth or falsity) to the variables in (1) — the agent assigns values to $\exists$-quantified variables, the environment assigns values to R-quantified variables. The environment is non-deterministic, with the actual assignment of values done at random. After all variables have been assigned values, the environment returns the goal state, $e^\top$, if the formula $\varphi(x_1, x_2, x_3, \ldots, x_n)$ is made true by the valuation traced out by agent and environment, and a dummy, "fail" state, $e^\perp$, otherwise. Each run thus corresponds to a possible valuation for the variables, and all possible valuations will be runs. Recall that the goal of the SSAT problem is to determine whether there is a strategy for assigning values to $\exists$-variables that makes the formula true in the majority of cases. This will clearly be the case just in case there is an agent that can succeed with the task (make the formula true) with probability greater than 1/2.

The details are as follows. For each $\exists$-quantified variable x_i, we create two actions, $\alpha_i^\top$ and $\alpha_i^\perp$, corresponding to the assignment of truth or falsity to x_i, respectively. For each R-quantified variable x_j, we create two environment states, $e_j^\top$ and $e_j^\perp$, again corresponding to an assignment of truth or falsity to x_j. We also create three additional environment states, e_0, $e^\top$ and $e^\perp$, let the initial state be e_0, and let the set of goal states S be a singleton containing $e^\top$. The state transformer function of the environment behaves as follows. The agent and environment alternate to give valuations to the quantified variables, with the agent going first, working from outermost variable to innermost. The agent is first allowed to pick a value for x_1 by performing $\alpha_1^\top$ or $\alpha_1^\perp$, to which the environment responds with either $e_2^\top$ or $e_2^\perp$, to which the agent must respond with either $\alpha_3^\top$ or $\alpha_3^\perp$, and so on. When all variables have been given values, the environment responds with either $e^\top$ if $\varphi(x_1, x_2, x_3, \ldots, x_n)$ is true under the valuation traced out, and $e^\perp$ otherwise. For example, the following run

$$e_0 \xrightarrow{\alpha_1^\top} e_2^\top \xrightarrow{\alpha_3^\perp} e^\top$$

is a run in which the variables $x_1, \ldots, x_3$ are assigned the values true, true, and false respectively; the environment indicates that the formula under question is true under this valuation. To ensure that the environment can always respond with a final valuation, we can "pad" runs by allowing an additional dummy action by the agent. □

You might now expect us to consider the FSD_{det}, i.e., the stochastic finite agent design problem for deterministic environments. For deterministic environments, however, the stochastic agent design problem is not meaningful: an agent will only ever have one run in a deterministic environment, and it thus makes no sense to ask whether the agent succeeds with probability exceeding 1/2 — its probability of success is either 0 or 1.

However, we can ask a closely related question, namely, do most *agents* satisfy the task in a deterministic environment. That is, given a deterministic environment and a particular (achievement) task, do the majority of agents satisfy this task in the environment? Another way of looking at this problem is to say that if we selected an agent at random, are the chances better than even that this agent would succeed with the task in the environment? We call this problem *majority agent design* ($MAJD$):

> *Majority agent design*:
> *Given*: Achievement agent design task with environment $Env = \langle E, e_0, \tau \rangle$ and goal states $S \subseteq E$.
> *Answer*: "yes" *the majority* of agents achieve the task, "no" otherwise.

We can immediately show:

THEOREM 6. *$MAJD_{det}$ is* PP*-complete.*

PROOF. We must first show the problem is in PP. A PP algorithm deciding the problem (see [6, pp.256–257] for a definition of the class PP) is:

1. a computation non-deterministically "guesses" a run consistent with the state transformer function τ of the environment;
2. a computation "accepts" if the task is accomplished on this run, otherwise it "rejects".

The PP machine accepts if the majority of its computations accept. Each different run corresponds to a different agent. Since we are considering finite runs, each run is guaranteed to have length at most polynomial in $|E \times Ac|$, the machine decides the problem.

We reduce from $MAJSAT$ [6, p.256] to show the problem is PP-hard. An instance of $MAJSAT$ is simply a propositional logic formula φ over boolean variables $x_1, \ldots, x_n$. The goal is to answer "yes" if the formula is true under most valuations of these variables, "no" otherwise. We create an instance of $MAJD_{det}$ as follows. The idea is that any given agent simply traces out a valuation for the variables: every different agent thus corresponds to a different valuation. Formally, the environment we create has four states: $\{e_0, e^\top, e^\perp\}$, with the initial state being e_0, and the states $e^\top$ and $e^\perp$ respectively used to indicate whether or not the valuation traced out makes the formula true or false respectively; the set of goals states is a singleton containing

$e^{\top}$. Until the agent has completed assigning values to variables, the environment responds to all assignments with e_0. When all Boolean variables have been assigned values, the environment responds with $e^{\top}$ (if the formula is true under the assignment traced out), or $e^{\perp}$ otherwise; the run then ends. $\square$

The $MAJD$ problem gives us a (somewhat crude) measure of the inherent hardness or easiness of a task: if there is a better than evens chance that an agent selected at random will succeed with the task, then the task itself must be easy to solve.

5. CONCLUSIONS

In this paper we have considered the complexity of agent design problems when a task is specified by an arbitrary propositional logical formula, thus extending results of [8, 9, 3]. It is shown that with history-dependent environments the the decision problem with arbitrary non-trivial formulae is "no harder" than that for single variable formula specifications. In contrast, one setting of these problems in history-independent environments gives rise to NP-complete decision problems. We also considered stochastic variants of agent design problems, In the AI literature, the most closely related work to our own is on the complexity of the planning problem: Bylander was probably the first to undertake a systematic study of the complexity of the planning problem; he showed that the (propositional) STRIPS planning problem is PSPACE-complete [2]. Building on his work, many other variants of the planning problem have been studied — recent examples include [5, 1]. The main differences between our work and the work on AI planning is as follows:

- The notion of an agent in our work (as a function that maps runs to selected actions) is more general than the notion of a plan as it commonly appears in the planning literature. Our agents are more akin to the notion of a *strategy* in game theory. The obvious advantage of our approach is that our results are not bound to a particular plan representation. The obvious disadvantage is that having a positive answer to one of our agent design problems does not imply that an agent to carry out the task will be *implementable* (cf. [7]).
- Most complexity results in the planning literature are bound to particular *representations* of goals and actions. The STRIPS notation in Bylander's work is one example [2]; Baral *et al.* use the action description language $\mathcal{A}$ [1]; in the work of Littman *et al*, the representation chosen is ST [5]. In some cases, it is not clear whether results reflect the complexity of the decision problem, or whether they are at least in part an artifact of the representation. We have adopted the most general representations possible, for example representing achievement and maintenance tasks through sets of states, rather than through a particular (e.g., logical) formalism, and consider several possible representations of the environment.
- Most complexity results in planning assume simple (deterministic, history independent) environments.
- The focus in the planning literature has been almost exclusively on achievement tasks. We have also considered maintenance tasks, and elsewhere, have considered richer task specifications [9].

In the computational complexity literature, the most relevant problems are those of determining whether or not a given player has a winning strategy in a particular two-player game. PSPACE-completeness appears to be the characteristic complexity result for such problems [6, pp459–480].

There are many issues that demand attention in future work. One is the relationship of our work to that of solving Markov decision problems [4]. Another key problem is that of determining the extent to which our positive (polynomial time) results can be exploited in practice. Yet another is on extending our task specification framework to allow richer and more complex tasks.

6. ACKNOWLEDGEMENTS

The work reported in this paper was carried out with the support of EPSRC Grant GR/R60836.

7. REFERENCES

[1] C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in presence of incompleteness. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden, 1999.

[2] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[3] P. E. Dunne and M. Wooldridge. The complexity of agent design problems: Determinism and history dependence, 2001. Submitted.

[4] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[5] M. L. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of AI Research*, 9:1–36, 1998.

[6] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley: Reading, MA, 1994.

[7] S. Russell and D. Subramanian. Provably bounded-optimal agents. *Journal of AI Research*, 2:575–609, 1995.

[8] M. Wooldridge. The computational complexity of agent design problems. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 341–348, Boston, MA, 2000.

[9] M. Wooldridge and P. E. Dunne. Optimistic and disjunctive agent design problems. In Y. Lespérance and C. Castelfranchi, editors, *Proceedings of the Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, 2000.