Advance Access publication on October 29, 2008
doi:10.1093/comjnl/bxn052

Designing Effective Policies for Minimal Agents

KRYSIA BRODA[1,*] AND CHRISTOPHER J. HOGGER[1]

[1]*Department of Computing, Imperial College London South Kensington Campus, London SW7 2AZ, UK*
**Corresponding author: kb@doc.ic.ac.uk*

A policy for a minimal reactive agent is a set of condition-action rules used to determine its response to perceived environmental stimuli. When the policy pre-disposes the agent to achieving a stipulated goal we call it a teleo-reactive policy. This paper presents a framework for constructing and evaluating teleo-reactive policies for one or more minimal agents, based upon discounted-reward evaluation of policy-restricted subgraphs of complete situation graphs. The main feature of the method is that it exploits explicit associations of the agent's perceptions with states. The framework allows to construct and evaluate policies for a number of cooperating agents by focusing upon the behaviour of a single representative of them. This abstraction ameliorates the potential combinatorial burden. Within the framework varied behaviours can be modelled, including communication between agents. Simulation results presented here indicate that the method affords a good degree of predictive power. The paper presents two different branch and bound algorithms used to optimize policy evaluation.

Keywords: minimal agent; multi-agent policies; policy evaluation

Received 22 January 2008; revised 26 September 2008

1. INTRODUCTION

This paper presents a framework for constructing and evaluating goal-oriented policies for particularly simple autonomous agents. It concentrates mostly upon purely reactive ones but also examines some conservative extensions to these, for instance the incorporation of hysteretic features. In the context of ambient intelligence, our work can be regarded as concentrating upon the implementation of environmentally embedded responsive agents that have been optimized for performance, having been selected by the optimization process from a multitude of candidates, each representing some percept–response relation consistent with some logical model of interaction between users and their environment. As an illustration, a personalized device intended for monitoring a user's diabetes would need to reflect some suitable logical theory describing the interplay between perceived glucose levels (or dynamics) and controlled responses of insulin delivery. Although such a theory might be embedded directly in the device as a rational or cognitive agent, it might admit as its consequences numerous alternative percept–response relations of varying efficacy. By contrast, our aim is to analyse such relations in advance to predict the most efficacious one, and then embed that as a compiled end-product into the device as a single reactive agent.

In this section we motivate our work and give an overview of the framework and of the paper. The technical presentation is deferred until Section 2.

1.1. Motivation

Our focus on very simple agents is motivated chiefly by emerging application contexts, such as remote exploration, where physical and economic constraints exclude sophisticated on-board processing. Besides their direct application to physical domains, however, these agents may also be used more generally as building-blocks in the design of algorithms performing state-transitions upon data structures, for example in sorting, searching or tiling problems.

Every agent we consider contains some policy which governs its behaviour and is intended to enable the agent to achieve some goal known to the designer of the policy. Our standard policy structure will be a set of mutually exclusive production rules of the form *perception* $\rightarrow$ *action*, usually intended to control durative behaviour: given some current perception the agent performs the corresponding action until acquiring a new perception, whereupon it reacts likewise to that. A goal for such an agent is typically some specified state(s) of the world together with some associated specified perception(s) for the agent.

We use the term *teleo-reactive (TR) policy* to signify a policy that has been designed with some particular goal in mind and, correspondingly, the term TR-agent to signify an agent governed by such a policy. The latter term was first introduced in [1] and further developed in [2, 3], but with a more specific meaning.

The characteristics of a TR-agent are, typically, that it behaves autonomously under the control of the policy stored within it, that the policy alone instructs it how to react to perceptions of the world, that it possesses very limited computing resources for program storage and interpretation and that it is predisposed by the policy to achieve some goal. Such an agent may or may not have sufficient perceptive capability to know, at any instant, the entire state of the world. An agent of the kind described in [3] is presumed at least capable of perceiving an intended goal state whenever that state arises, and is accordingly designed with that capability in mind. Its policy includes an explicit test for the goal state, whilst the nature and ordering of its rules are inferred by reductive analysis of that test. Its goal-orientedness is thus explicit in the policy. By contrast, we make two key assumptions about the kind of TR-agents studied in this paper, and first introduced in [4]; namely that they have (i) little or no access to cognitive resources, such as beliefs or reasoning systems, and (ii) only partial observational capability, in that their perceptions may not fully capture the whole world state, whether a goal state or otherwise. The design process now relies not upon goal-reductive analysis but upon comparing the extents to which alternative policies dispose the agent towards achieving a goal—as judged, for instance, by a discounted-reward principle. A policy identified on this basis is *implicitly* goal oriented.

Informally, a good policy is the one which disposes an agent to perform well in pursuit of a defined goal whatever state it is currently in. TR-agents therefore occupy a middle ground between wholly reactive agents [5] whose actions have no overall motivation, and wholly deliberative agents [6] whose actions—including on-the-fly planning—flow from an explicit stored goal. A significant advantage is the relatively low resources a TR-agent needs for its internal logic, which consists of little more than a fixed rule-set requiring minimal hardware for its execution. Unlike a deliberative agent, it does not need on-board computational facilities capable of executing arbitrarily complicated software. Nevertheless, provided that the rule-set has been constructed with appropriate regard to the desired goal, the agent is likely to achieve it.

The kind of agents we are considering are potentially suitable for many contexts requiring low-cost determination of responses to sensory inputs. One rapidly developing such context is that of wireless sensor networks (WSNs) as described in [7, 8]. There, agents are implemented as programs distributed across networks of embedded miniature sensing devices capable of communicating wirelessly. In [8] such agents are exemplified for intruder detection, whilst the Agilla framework of [7] shows their application to fire detection, cargo monitoring and robot navigation.

Agilla's agent programs, like our policies, do not deliberate with explicitly represented goals but nonetheless effectively compute percept–response relations sufficient to achieve the overall goal of the WSN. They inter-communicate by read–write operations on tuple spaces local to their own host devices or those of their neighbours, a mechanism which our own agents would implement by the kind of extended perception discussed in Section 5. Their migrations between devices can include transporting details of their internal state such as the current value of their instruction counter; our own agents can, analogously, internally record their current perception using the kind of memory extension discussed in Section 6.

A comparison of our framework with other approaches to agent design is given in Section 7. In particular we discuss the contrast with deliberative agent policies [9].

1.2. The problem

The problem we address in this paper is that of predicting optimal functional policies for single and multi-agent environments in which a stochastic (teleo) reactive agent perceives (in general) only partial information about any state. This partial-observability (PO) restriction applies also (in general) to any state that happens to be a goal for the agent. In our context, *stochastic behaviour* means that when an agent takes a particular action there can be alternative outcomes associated with some probability distribution. By *functional behaviour* we mean that an agent *always* takes the same action in response to a given perception.

Many well-tried policy-prediction frameworks exist for reactive agents but, as far as we can determine, none yet exists that is guaranteed to deliver, for large problem contexts, an optimal functional policy under the PO restriction. The various algorithms used in the PO Markov decision process (POMDP) framework, including the effective but computationally demanding Witness Algorithm [10], yield policies which are optimal but are not simple functions from perceptions to actions. Instead, those policies are essentially decision trees in which the action taken depends upon a history of prior actions.

By contrast, some other frameworks do yield policies that are functional but are not necessarily optimal, though they will often be near-optimal. An example of this is the sarsa reinforcement learning algorithm [11]. These techniques impose an MDP paradigm and ostensibly treat perceptions as states in a conventional MDP but require full consideration of the concrete state-space in order to estimate policy values. Genetic algorithms are also applicable to the problem but, again, without any guarantee of optimality.

The issue of scalability has to be confronted by all these frameworks. Typical applications present relatively small sets of possible percepts and actions but very large sets of possible states, whilst the relation between states and what can be

perceived in them has no general pattern. This had led some researchers to resort to methods of abstraction which reduce the scale of the problem by collecting perceptions or states (or both) into sets to form a smaller number of individual abstract perceptions or abstract states to which the predictive method can then be realistically applied [12].

Abstraction, however, necessarily discards some information and may not necessarily deliver optimal policies. Moreover, structural features of some abstraction methods may render it impossible to find a good policy that is also functional, instead requiring the agent to act differently in response to the same perception of different abstract states.

The difficulties outlined above become seriously compounded in the multi-agent context. None of the POMDP or genetic algorithm frameworks extend easily to this context.

This paper explores the policy-prediction problem using a fundamental representation of it that we call the situation graph framework (SGF), presented formally in Section 2. Each node of an situation graph represents a situation in the form (o, p), where the term p encodes an agent's perception and o encodes what we call the objective state, which excludes the perception. What we call a situation is elsewhere in the literature often called 'state', but in this paper we shall use the term *state* always to refer to this o-component. The arcs in the situation graph represent possible actions. This distinction between o and p offers more details to work with than the standard MDP formulation, in which perceptions are not explicitly separated from their associated states. In the SGF policy values are defined by a discounted-reward formula and, for small-scale applications, a guaranteed optimal policy can be computed using a standard equation-solving algorithm. With a modest increase in scale the optimal policy can still be computed using branch-and-bound techniques. However, if the scale is increased radically then, as with all the other frameworks, compromises must be made which forfeit the guarantee of optimality though may still produce very good policies.

For larger-scale contexts the SGF can employ *situational abstraction* in a manner that preserves the functional character of policies but renders them as functions from abstract percepts to actions. We shall not describe this further in this paper but instead refer the reader to detailed accounts of it in [13, 14].

Our treatment of multi-agent contexts can also be viewed as a form of abstraction. We expose the issues involved in this, using a novel elaboration of the situation graph, so that it continues to represent the transitions experienced by one agent but now includes not only those that are self-effected but also those effected by the exogenous actions of other agents, which we call x-transitions. This elaboration is scalable to the extent that the complexity of the graph is neutral with respect to the number of acting agents.

Besides the SGF itself, our main contributions are the following: first, we dissect the issues involved when formulating and evaluating policies in order to provide full perspective; second, we present and compare two new algorithms, both employing branch-and-bound techniques but in very different ways, for determining optimal policies in medium-scale contexts and third, for the cloned multi-agent case we show empirically that predictions based on our notion of x-arcs compare well with the values obtained by simulation.

1.3. Overview of the framework

We will now give a preliminary sketch of the framework. The principal construct employed for any given application is a primary graph, called the *unrestricted graph*, whose nodes denote situations and whose action labelled arcs denote transitions between them. In this model a situation combines an objective state with a (usually partial) perception of that state. Since policies are perception–action mappings, the number of possible policies clearly depends upon the number of possible perceptions. Moreover this dependence is generally exponential because each perception may (usually) be mapped to any of several possible actions. For instance, one of our exemplifications later in the paper is a simple application that involves just four blocks and three actions, yet potentially offers hundreds of policies to consider. The inclusion of objective state information in the nodes of the graph does not influence this number, but does increase further the burden of assessing policies. Each policy is some restricted subgraph of the primary one, the restriction being that the arc(s) emergent from a node shall be identically labelled by whatever action the policy dictates for that node's perception. Whilst the policy's perception–action mapping will be functional in all the applications considered in this paper, a node typically has several emergent arcs to different situations for the same action, which we call *a bundle*, so that whilst an agent's policy is deterministic its behaviour need not be. In order to evaluate a restricted graph we need therefore to assign a probability distribution across each such bundle of arcs.

In addition rewards must be assigned to arcs. Without this provision the evaluation could consider only how probable it was that a goal situation would be reached (if at all), but not how worthwhile it was to reach it in relation to the effort expended by the agent in doing so. The restricted graph and these assignments then form the entire basis for determining the policy's value, which is a quantitative measure calculated using a discounted-reward principle. An optimal policy is the one having greatest value among all policies.

The problem of designing optimal or near-optimal policies for a group of one or more similar TR-agents, called *clones*, operating in the context of *exogenous* events is also addressed within our framework and was first introduced in [15]. From the viewpoint of any individual agent an exogenous event is any change in the world not caused through its own actions, so includes actions of other agents according to their policy as well as serendipitous actions caused by external agents.

The multi-agent context poses further challenges of its own, since the design process must take some account of the multi-situations in which the agents find themselves and of the inter-dependent effects of their actions. This extra complexity in dealing with multiple agents makes it impractical to extend our graphs to represent comprehensively all the combinatorial possibilities arising if their nodes were generalized to denote multi-situations. Instead, for these contexts we employ a conservative extension of a restricted graph, so that it represents the behaviour of any one agent (termed *self*) together with the possible effects upon it of the behaviour of any other agent (termed *other*). Having only two notional entities—*self* and *other*—represented in this extension is another contribution towards the scalability of the design process. An additional consequence of our approach is that it can be used to model also the effects upon *self*'s possibilities of events that arise exogenously rather than from other agents. In this case *other* can be identified with whatever external source is responsible for the exogeneity. Our use of such representative entities, which we first investigated in [4], is somewhat similar in motivation to [16] in which the focus is upon MDP-based design, and to [17] in which the focus is upon design by reinforcement learning. We have so far applied it only to cloned agents, that is, ones all having the same policy. Whatever that policy is, just one restricted graph is sufficient to represent it. For n agents having distinct policies our method would require the consideration of only n restricted graphs, one for each instance of *self* with its own policy, having special arcs denoting the possible impacts of any of the other agents. This conservative treatment of differentiated agents is a further contribution to scalability.

We have also investigated the applicability of the framework to limited but potentially useful extensions of the supposed architectures of the agents, such as possession of small amounts of internal memory (besides that employed for storing policies) and communication. Several examples in this paper examine cases where an agent may broadcast some of what it perceives to all other agents, who then assimilate this information into their own perceptions. They may then cooperate to a greater degree than otherwise and so perform more effectively. This mechanism is not as powerful as those that could communicate in a more strategical fashion by, for instance, delegating subgoals or notifying discoveries of unprofitable subgoals. It is equivalent in power to having agents lacking the mechanism but having increased observability of the current state. Nevertheless, it is a conceptually useful way in which to model agent percipience.

In order to produce empirical evidence in support of the framework we test and evaluate policies on a simulator and then compare their observed values with those predicted from the SG. The main interest there is the extent to which the predicted ranking of policies agrees with the observed ranking. The paper contains a number of charts derived from these experiments, and from each one we derive a single quantity—a rank correlation statistic—that measures the predictive quality across the set of policies evaluated. In many of these examples the simulations are deliberately more concrete than the models employed in the situation graphs, in order that we may assess how well the predictions stand up in the presence of such an abstraction gap. We explain how that gap can cause the prediction method to make two distinct kinds of mistake in calculating policy values, giving rise to fluctuations in the charts.

However, our experience indicates that the gap would need to be very substantial in order to render the method incapable of broadly distinguishing between good and bad policies. It is, of course, a separate question as to how well the predicted best policies would fare if installed in real-world physical agents charged with pursuit of the same goals, which amounts to questioning the significance of a second abstraction gap between the simulator and the real-world. For instance, some of our examples seek to model the notion of an agent wandering to various locations in search of particular items. Currently the simulator represents locations and agent movements only discretely rather than continuously, whilst it does not represent agent breakdowns, conflicts, aborted actions and all the other dysfunctionalities to which real entities are susceptible.

This article is organized as follows. The basic framework is presented in Section 2 and the methods of policy valuation and framework tools are presented in Section 3. The extensions required for a multi-agent environment are presented in Section 4 and extended to include communication in Section 5. An exploration of the use of memory as perception and comparison, with an example, with methods that solve the problem of perceptual aliasing by using state estimation is given in Section 6. Section 7 positions our approach within related works and the paper concludes with Section 8.

2. BASIC ASPECTS OF FORMULATION

Our framework will be described through several different example domains, selected to illustrate its flexibility. There are two principal data structures upon which the framework is founded: situations and situation graphs. These are introduced next and we will use by way of illustration *BlocksWorld*, where agents are assumed to operate in an environment consisting of towers of blocks that can be built upon or dismantled. Initially, it is assumed there is just one agent acting. In Section 4 the framework is used to model and evaluate policies for a group of agents.

2.1. Situations

Any world in which our agents operate is one capable of assuming various *(objective) states*. A state in *BlocksWorld* might, for example, be represented by [1, 1, 2, 2, 2] signifying that the state comprises two towers each of size(height) 1 and three towers each of size 2 (all presumed standing on a

surface). We will denote by $\mathcal{O}$ the set of all states for a particular application. Any agent has three main features: a set $\mathcal{P}$ of *perceptions* it may have of the world, a set $\mathcal{A}$ of possible *actions* it may take and a *policy* relating perceptions to actions. In any objective state $o \in \mathcal{O}$, the agent's physically possible perceptions form some subset $P(o) \subseteq \mathcal{P}$. A perception is an observation made by the agent of some aspect of the objective state and may include introspection of itself or of other agents. For instance, an agent may be able to 'perceive' its previous action, by virtue of a limited memory or it may be able to perceive that no other agent is currently holding anything, by virtue of limited communication between agents, as discussed later. In response to any perception $p \in P(o)$ the agent's possible actions form some subset $A(p) \subseteq \mathcal{A}$ and a *policy* for the agent is any total function $f: \mathcal{P} \rightarrow \mathcal{A}$ satisfying $\forall p \in \mathcal{P}$, $f(p) \in A(p)$. The number of possible policies is the product of the cardinalities of the $A(p)$ sets for all $p \in \mathcal{P}$. A *situation* for the agent is any pair (o, p) for which $o \in \mathcal{O}$ and $p \in P(o)$. We denote the set of all situations by $\mathcal{S}$, one or more of which may be designated *goal* situations.

A perception does not, in general, capture the entire state of the world. On the contrary, our low-resource assumption entails that the agent normally perceives only a very limited amount of information about that world state. If a perception p fully described an objective state o then the situation (o, p) could be contracted simply to o, and the design process would need only to compare conventional state-transition graphs. However, the realistic position is one of partial observability. The problem is therefore how to optimize, for a goal incorporating a world state, an agent that (generally) cannot recognize that state.

It is important to the framework that the action set $\mathcal{A}$ for every agent shall include a special action which we call `wander` and denote by `w`. The `wander` action enables the agent to change its perception without altering the world state, and exemplifies 'wandering through the different perceptions of a state'. To say that an agent wanders does not necessarily entail that it literally moves about spatially. It means only that it refreshes, and perhaps changes, its perception. A thermostat sensing a sudden change in temperature is in our terms performing a wander action, as also is an agent that, seeing the surface, makes a spatial move after which it continues to see (another part of) the surface. The only difference is that the latter case is a reflexive transition of its situation, whereas the former case is not. In *BlocksWorld*, for instance, `w` corresponds to enabling the agent literally to wander around on the surface, so bringing various items into its range of vision.

DEFINITION 2.1. *A TR-application is a tuple* $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ *comprising representations of the assumed objective states* ($\mathcal{O}$), *perceptions* ($\mathcal{P}$), *actions* ($\mathcal{A}$).

DEFINITION 2.2. *Let* $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ *be a TR-application. A situation is a pair* (o, p) *where* $o \in \mathcal{O}$ *and* $p \in \mathcal{P}$ *and* p *is a perception that the agent may have of* o (*i.e.* $p \in P(o)$).

2.2. Situation graphs

Our framework for assessing policies potentially considers the full range of possibilities determined by the assumed world together with the possible perceptions and actions of the agent. It employs a structure which we refer to as the *unrestricted situation graph G.* This shows the situations that a representative agent called *self* may be in and the possible actions it may take. Each directed arc in G signifies, and is labelled by, some such action. When *self* is in a situation (o, p) its possible actions depend only upon p. Analysis of this graph enables us to extract policies appropriate to particular goals. After *self* has acted in a situation its new situation is determined by the assumed physics of the world (and of the kind of agent) being modelled. For example, if *self* is seeing a 2-tower and places a held block upon it, it is a natural choice to determine that the result is a 3-tower and that *self* is now seeing that 3-tower rather than seeing anything else. But note that the framework can easily model a situation in which the outcome was less certain; for example, the place action might not be successful and the outcome could be seeing a 2-tower still, or even seeing a 1-tower, the place action having destroyed the original tower.

DEFINITION 2.3. *Let* $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ *be a TR-application. The* unrestricted situation graph, *denoted by G, is a directed graph whose nodes are the situations. The arcs emanating from a situation* (o, p) *are precisely those corresponding to* $A(p)$ *and each one is directed to a situation that self could be in if that action were taken.*

The unrestricted situation graph discloses how any situation can (or cannot) be reached from another, in particular whether a given goal situation can be reached from a given initial situation. It may also reveal subgraphs from which a given goal could never be reached.

A key feature of our framework is the process of pruning selected arcs from the unrestricted graph G according to some policy f, to leave the *f-restricted* graph, denoted by G_f. This graph commits the agent to take, in any situation, just that action determined by policy f and shows what will actually happen.

We can use *BlocksWorld* to exemplify these constructions and some of the issues entailed in considering choices of policy for a given goal.

EXAMPLE 2.1. Suppose that there is one agent and four blocks. The blocks can be arranged into towers and the agent can hold at most one block. There are eight states, any of which can be represented by a list such as [1, 2], indicating a configuration of towers composable from the blocks not being held. Next consider what the agent might 'see' in this world. We assume an agent can see either the surface or one tower of recognizable height and sufficient descriptors for this are $s0$, $s1$, $s2$, $s3$ and $s4$. Further suppose that the agent knows whether or not it is holding a block. Two further

descriptors *h* and *nh* suffice for this. Together, the descriptors yield eight different perceptions, and valid combinations of states and perceptions give rise to 19 situations. An example valid situation is ([3], (*s*0, *h*)), whereas the perception (*s*4, *h*) is impossible in a 4-block world.

Finally, assume the agent can perform three kinds of action—`pick` (remove and hold the top block of a tower being seen), `place` (put a held block onto the surface being seen or onto a tower being seen) and `wander` (update what is being seen). The abbreviations `k`, `l` and `w` suffice to distinguish these. A possible perception–action pair (*p, a*) is ((*s*0, *h*), `w`), whereas ((*s*0, *h*), `k`) is not. This follows from the assumed logic of the chosen descriptors, which derives in turn from the assumed physics of the application. The syntactical form of the descriptors is here immaterial provided one retains awareness of what they are intended to denote, and provided that what they denote are correct properties of the original conceptualization. Whilst the syntax ([1], (*s*0, *h*)) contains helpful visual cues to aid that awareness, for analytical purposes it may as well be iconized to a simpler form such as (2, *b*) or even 2*b*. In the sequel we shall often use these simpler forms in order to reduce presentational clutter.

Figure 1 shows all the possible state and perception descriptors together with their iconic labels (1, 2, 3, ... *etc.* and a, b, c, ... *etc.*, respectively) for Example 2.1. Here, *s*0 denotes that *self* sees the surface, *sn* ($n > 0$) denotes that it sees a tower of height *n*, *h* denotes that *self* sees that it is holding a block and *nh* denotes that it sees that it is not holding a block. The figure also shows, for each perception *p*, the set $O(p)$ of associated states and the set $A(p)$ of associated actions. Figure 2 shows the resulting unrestricted SG *G*. For the sake of compactness, a situation such as (2, *f*) is shown there simply as 2*f*.

The next step is to consider what the agent might be required to achieve and what its behaviour might be. A policy for the agent in Example 2.1 might be

$$\begin{array}{lll} s1,\ nh \to \mathtt{k}, & s1,\ h \to \mathtt{l}, & s2, h \to \mathtt{l}, \\ s3,\ h \to \mathtt{l}, & s2,\ nh \to \mathtt{w}, & s3,\ nh \to \mathtt{w}, \\ s4,\ nh \to \mathtt{w}, & s0,\ h \to \mathtt{w}, & s0,\ nh \to \mathtt{w} \end{array}$$

giving the *f*-reduced graph shown in Fig. 3.

	$o \in \mathcal{O}$
1	[2, 2]
2	[1, 1, 1, 1]
3	[1, 1, 2]
4	[1, 3]
5	[4]
6	[1, 1, 1]
7	[1, 2]
8	[3]

	$p \in \mathcal{P}$	$O(p)$	$A(p)$
a	s1, h	{6,7}	{l,w}
b	s2, h	{7}	{l,w}
c	s3, h	{8}	{l,w}
d	s1, nh	{2,3,4}	{k,w}
e	s2, nh	{1,3}	{k,w}
f	s3, nh	{4}	{k,w}
g	s4, nh	{5}	{k,w}
h	s0, h	{6,7,8}	{l,w}
i	s0, nh	{1,2,3,4,5}	{w}

FIGURE 1. States, perceptions and actions (Example 2.1).

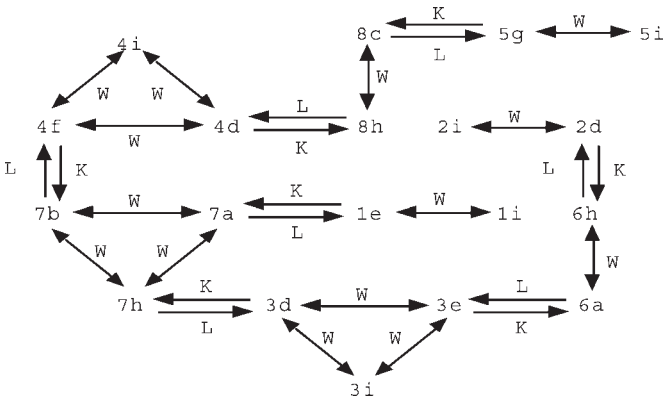

FIGURE 2. Unrestricted situation graph (Example 2.1).

In general, the result of any agent's actions may vary according to the initial world state presented to it. However, in this example the result is generally (but not always) a 4-tower (that is, a tower of size 4) provided that `w` is implemented in a fair manner—that is, allows all perceptions in the world to be experienced in the long run. Once this tower has been built the agent can only wander indefinitely unless terminated by some extraneous mechanism.

The state having a 4-tower is, in fact, the intended goal for this policy. This goal is not explicit in the policy, nor is it made known to the agent by any other means. Instead, the policy has been constructed by a procedure that takes account of the intended goal (or goals).

Even for a simple world and goal, a suitable policy can be very difficult to compose using intuition alone.

Let $\mathcal{S}$ be the set of all situations in the problem formulation (and hence in *G* and in all its policy-restricted subgraphs). Then the choice of *f* partitions $\mathcal{S}$ into two disjoint subsets N_f and T_f called the *nontrough* and the *trough*, respectively. N_f contains the goal and all situations from which the goal is reachable under policy *f*. T_f contains all the other situations. This partitioning facilitates useful economies in the operation of the framework.

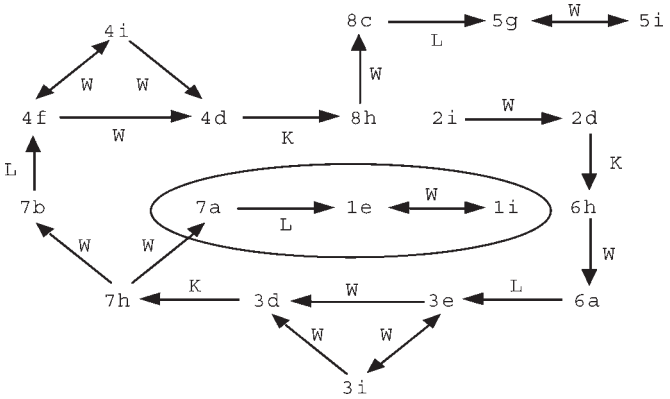

FIGURE 3. An *f*-restricted graph (Example 2.1).

In Fig. 3 the trough, circled for emphasis, is the node set {(1, *e*), (1, *i*), (7, *a*)}. From all situations outside the trough, the agent will achieve the goal unless it subsequently enters the trough. For instance, from (7, *h*) it may wander into the trough, but if it instead wanders to (7, *b*) then the goal will be achieved.

The essential problem in this example is in deciding upon the best actions for perceptions *a* and *e*. Since each one has two possible actions there are four possibilities to consider. It turns out that none of them can avoid a trough somewhere. For instance, if we modify the original policy by choosing `k` for perception *e* and `w` for perception *a*, we obtain a different *f*-restricted graph and this policy

$$\begin{array}{lll} s1,\ nh \rightarrow \texttt{k}, & s2,\ nh \rightarrow \texttt{k}, & s2,\ h \rightarrow \texttt{l}, \\ s3,\ h \rightarrow \texttt{l}, & s1,\ h \rightarrow \texttt{w}, & s3,\ nh \rightarrow \texttt{w}, \\ s4,\ nh \rightarrow \texttt{w}, & s0,\ h \rightarrow \texttt{w}, & s0,\ nh \rightarrow \texttt{w} \end{array}$$

In this case the set {(2, *d*), (2, *i*), (3, *e*), (6, *a*), (6, *h*)} is the trough. Each of perceptions *a* and *e* has the property of being associated with several states having different 'best' actions. The limited perceptions of the agent render it unable to know which world state it is in and hence which of the possible best actions to take.

Fundamentally, given the particular block-world and goal stipulated, the agent in this example is not perceptive enough to cope well with all situations. It suffers from *perceptual aliasing*, meaning that several (different) situations are perceived similarly by the agent. In [4] one way of elegantly improving the agent is to equip it with a single-register memory capable of recording whether it has ever seen a tower of size at least 2, and to treat the reading of the register's state as another perception. For this modified agent one can find a much better (though still imperfect) policy.

Using registers in this way to deal with perceptual aliasing presumes that their stored contents, at the moment they are looked up and exploited, are useful to the agent in dealing with the current situation. For instance, if an agent recalls from its register that it once saw a tower of size 2 then it may not follow that such a tower persists in the current state. Previous actions by other agents or exogenous sources may by now have reduced all towers to size 1.

Only if these possibilities can be excluded can the agent rely upon its stored memory as being a persisting truth about the world, provided also that it has not itself altered that 2-tower since it last saw and memorized it. In most of our studies we have not assumed any memorizing capability for our agents, focusing instead upon what can be achieved without the facility and thereby maintaining our minimal-hardware assumption. However, in cases where modest amounts of memory might be justified we can easily model the feature in our framework as it stands and Example 6.1 in Section 6 illustrates this.

3. POLICY EVALUATION FOR SINGLE AGENTS

The value of an agent's policy is a global measure of how well the agent, proceeding from any situation, performs in the long run under that policy. Evaluating a policy cannot, in general, be reduced to local considerations of how the agent acts at particular situations, since an action taken for a perception *p* in one situation might produce very different outcomes when taken for the same perception in a different situation. Instead, we must estimate the worth of a policy *f* as the sum of the expected values of all the situations in G_f, where the expected value of a situation *s* is the benefit to the agent of proceeding from *s*.

This section describes how this estimation is achieved in a single-agent context using the discounted-reward principle [14], defined as follows.

DEFINITION 3.1. *Let f be a policy for a TR-application* $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ *and let* $s = (o, p)$ be a situation in G_f *and let SS be the successor set of s. The* discounted reward $V(s, f)$, *measuring the benefit of proceeding from s, is given by the Bellman formula*

$$V(s,f) = \Sigma_{u \in SS}(P_{su} \times (\Upsilon_{su} + \gamma(f(p)) \times V(u,f)))$$

In the above, Υ_{su} is the immediate reward for the action $f(p)$ that takes *s* to *u*, when the agent has perception *p*. P_{su} is the probability that from *s* the agent proceeds next to *u* and the factor $\gamma(f(p))$ discounts the benefit of taking that action at *s*. Note that for a situation with no successors, this formula gives the expected value as 0. Normally, we choose $0 < \gamma(f(p)) < 1$ to reflect the cost to the agent—in time or other resources—of performing successive actions. The formula for $V(s, f)$ given in Definition 3.1 is called the *infinite horizon discounted reward* formula and is suitable when we are interested in the long-term behaviour of an agent. When the interest is in agent behaviour over shorter, fixed-length paths, a different formula is appropriate; this is the *finite horizon discounted-reward* formula as defined next.

DEFINITION 3.2. *Let f be a policy for a TR-application* $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ *and let* $s = (o, p)$ *be a situation in* G_f *and SS be the successor set of s. The* finite discounted reward $V(s, f, k)$, *effectively measuring the benefit of the agent proceeding from s and having at most k steps available is given by the formula*

$$\begin{aligned} V(s,\ f,\ 0) = &\quad 0 \\ V(s,\ f,\ k) = &\quad (\textit{if } k \geq 1) \\ &\Sigma_{u \in SS}(P_{su} \times (\Upsilon_{su} + \gamma(f(p)) \times V(u,\ f,\ k-1))) \end{aligned}$$

Unless one has good reason to distinguish rewards for actions from different situations, the fine tuning provided for in the above definitions can be dispensed with. It then suffices

to use some fixed values r, R and T and a fixed discount factor $0 < \gamma < 1$ such that

$\Upsilon_{su} = R$ if the action $f(p)$ taken at s leads immediately to a goal

$\Upsilon_{su} = T$ if the action $f(p)$ taken at s leads immediately to a node in the trough (for f)

$\Upsilon_{su} = r$ otherwise

For the infinite horizon case the situations' values are related by a set of linear equations which, since $\gamma < 1$, have unique finite solutions even when G_f contains cycles signifying non-terminating behaviour.

Since we are interested in policies that perform well, on average, from whatever state an agent may find itself in, we define the (predicted) value of a policy f, denoted $V_{\text{pre}}(f)$ to be the weighted average of $V(s, f)$ over all situations s in the f-reduced graph. In this paper we assume all nodes are equiprobable as initial nodes or as nodes resulting from unpredictable, but rare, exogenous behaviour, so the average is the usual mean.

DEFINITION 3.3. *Let f be a policy for a TR-application $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ and S be the set of situations in G_f. The value $V_{\text{pre}}(f)$ of policy f is given by*

$$V_{\text{pre}}(f) = \Sigma_{s \in S}(i_s \times V(s, f))$$

where i_s is the probability that an agent may initially be in situation s.

The relative values assigned to R, r and T govern the extent to which the method accords merit to the agent for reaching a goal rather than not doing so, and penalty for entering a trough rather than not doing so. In general, choosing $R \gg r$ ranks more highly those policies well-disposed to the reaching of the goal and choosing $r \gg T$ will rank steps leading to the trough very lowly. The value of γ controls the separation (but not, in general, the ranks) of the policies' values. Both the finite horizon formula and the infinite horizon formula give the value of a policy as a sum of the form $\alpha r + \beta R + \delta T$. For the experiments reported in this paper we have used the infinite horizon method with values of $r = T = -1$, $R = 100$ and $\gamma = 0.9$. It is shown in the Appendix that the finite discounted reward for a situation (given by Definition 3.2) tends, in the limit for k, to the discounted reward for a situation (given by Definition 3.1).

3.1. Policy prediction

As the examples in this paper demonstrate, even quite simple problems can determine large and complex situation graphs whose manual characterization would be highly tedious and error-prone. To reduce the scope for erroneous formulation we employ, for *BlocksWorld* (and, potentially, other scenarios), an *situation graph generator* program which, for any designated set of blocks, agents, actions and goal, autonomously constructs the unrestricted situation graph and further checks it against robust integrity constraints. We further employ *policy predictor* programs to compute policy values according to Definition 3.2, and also to compute the upper bounds on their success-rates. In fact, in most of our experiments we found that, over the infinite horizon, the values of policies which tend to cause the agent to enter a trough turn out to be much lower than those that do not, even when $T = r$. Moreover, the ranking of policies is largely insensitive to the choice of values for r, R and γ, provided that $R \gg r$. The smaller γ is, the less important is the dominance of R over r. Thus the method does not require domain-oriented intuitions about these parameters beyond giving prominence to goal situations.

We note here that if the only arc for some situation s is a reflexive arc, then s is necessarily in the trough and the value of s is $r/(1 - \gamma)$. However, it turns out that if a node is in the trough and has at least one exit arc, its value is also $r/(1 - \gamma)$. This is proved in [13].

Returning to Example 2.1, the $A(p)$ sets determine that there are 256 possible policies. There are 19 situations, and varying the policy merely varies the arcs connecting them. A test run was made for the case $r = -1$, $R = 100$, $\gamma = 0.9$ which identified two optimal policies

$a \to \mathtt{w}$, $b \to \mathtt{l}$, $c \to \mathtt{l}$, $d \to \mathtt{k}$, $e \to \mathtt{k}$,
$f \to \mathtt{w}$, $g \to \mathtt{k}$, $h \to \mathtt{w}$, $i \to \mathtt{w}$

$a \to \mathtt{w}$, $b \to \mathtt{l}$, $c \to \mathtt{l}$, $d \to \mathtt{k}$, $e \to \mathtt{k}$,
$f \to \mathtt{w}$, $g \to \mathtt{w}$, $h \to \mathtt{w}$, $i \to \mathtt{w}$

each having an overall value of 37.3. The policy corresponding to the f-reduced graph G_f in Fig. 3 for this example, namely

$a \to \mathtt{l}$, $b \to \mathtt{l}$, $c \to \mathtt{l}$, $d \to \mathtt{k}$, $e \to \mathtt{w}$,
$f \to \mathtt{w}$, $g \to \mathtt{w}$, $h \to \mathtt{w}$, $i \to \mathtt{w}$

is predicted to be the third-best one, having overall value 35.5. The graph G_f has a trough, containing three situations, which can be entered from eight exterior ones. By contrast, for each of the two optimal policies G_f has a trough, containing five situations, which can be entered from just one exterior one. The policy values, therefore, correctly reflect the propensity, on an average, of the agent failing to reach the goal by entering a trough.

In small-scale contexts an optimal policy can be determined by simply evaluating all policies and extracting the best of them. Evaluating any particular policy amounts only to solving a set of simultaneous linear equations whose variables

are the situation values. We shall use the acronym *EPE* (exhaustive policy evaluator) to denote this simplest of optimization algorithms. *EPE* is particularly useful when it is desirable to know the values of all policies, for instance to draw comparisons with simulation results. Later in the paper we will apply *EPE* to several examples for that specific purpose, that is, to measure how well the predictions across the entire policy spectrum correlate with observed (simulated) behaviour.

Our software was developed primarily using LPA-Prolog[1] through transparent and easily-adapted representations. Despite the relative tardiness of an interpretive formalism such as Prolog, it takes only a minute or so for *EPE* to evaluate 10 000 policies over 20 situations on an Apple G4 platform.

In mid-scale contexts, which might typically offer some millions of policies over some hundreds of situations, it is more appropriate to employ one or other of our new branch-and-bound optimizers that will be described presently. These two were developed first in Prolog but subsequently reprogrammed in Java whose execution speed we found to be approximately 300 times faster than Prolog. These new optimizers are named iterative branch and bound (*IBB*) and fixed stratification branch and bound (*FSBB*). Both of them expand a tree whose leaves are partial policies and whose value bounds are compared on-the-fly in order to prune out sub-optimal sub-trees. The efficiency of *IBB* derives in part from its use of value iteration to evaluate leaves, whilst *FSBB* does no such iteration but instead derives its efficiency from applying the Bellman formula to highly localized situation subsets rather than to the entire situation set. The relative efficiency of these two algorithms turns out to depend critically upon features of the situation graph and of the chosen goal, as we will show presently for particular examples.

3.2. Iterative branch and bound IBB

In this section we describe a particular instance of a branch and bound method, modified and adapted for the SG framework from an algorithm of Littman [18]. In order to describe it we define some new terms.

DEFINITION 3.4. *Let* $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ *be a TR-application. A* full-policy *is the one with a rule for every perception. A* part-policy *is the one which has rules for a (possibly empty) subset of the perceptions. A perception p for which* $\mathcal{A}(p)$ *is a singleton is called* fixed. *A perception p which is associated uniquely with an objective state o is called free. All other perceptions are* constrained. *A part-policy that gives a rule for every fixed or constrained perception is called an* effective *policy*.

It follows from Definition 3.4 that free perceptions are necessarily associated uniquely with a situation. The set of (full) policies can be structured using a tree, called a *policy tree* and denoted $\mathcal{T}$, in which each node is labelled by a perception p and each arc by an action in $A(p)$. If $\mathcal{P}$ has size n, then each branch of a tree developed to depth n represents a full-policy and each branch of a tree developed to a depth of $<n$ represents a part-policy. Recall from Definition 3.3 that the value $V_{\text{pre}}(f)$ of a full-policy f is the average of the values $V(f, s)$ for each situation s. This value is approximated by iteration using the recursive Equation (1)

$$V_{i+1}(f, s) = \Sigma_{u \in SS(s)P_{su}}(\Upsilon_{su} + V_i(f, u)) \tag{1}$$

where $SS(s)$ is the set of successor states of s, P_{su} the probability of the transition s to u and Υ_{su} the reward for taking it, all as prescribed by the policy f.

The value of a part-policy f is given an upper bound, computed using a similar formula to that in Equation (1), except that an optimistic value is used for those situations where actions have not been specified. That is, Equation (2) is used.

$$V_{i+1}(f, s) = \begin{array}{l} \Sigma_{u \in SS_1(s)} P_{su}(\Upsilon_{su} + V_i(f, u)) + \\ \max_a \Sigma_{u \in SS_2(a,s)}(P_{su}(\Upsilon_{su} + V_i(f, u))) \end{array} \tag{2}$$

where $SS_1(s)$ gives successor states for situations for which policy f gives an action and $SS_2(a, s)$ gives the successor situations of s assuming action a is taken. The value computed by Equation (2) will be an upper bound for the value of any full-policy that extends part-policy f. In the policy tree $\mathcal{T}$ each leaf node can be annotated by the policy value as computed by Equation (1) and each non-leaf node can be annotated by an upper bound for the policy value, as computed by Equation (2). In what follows we will assume that the best predicted policy value only is required although it is easy to keep a beam of size m if the best m predicted policy values are required. A policy tree is searched left-right and depth-first and nodes that remain to be evaluated can conveniently be maintained on a stack. A global optimum policy value B is maintained; initially B is set to a value of some known (reasonably good) policy. Thereafter, the branch and bound procedure *IBB*, whose pseudocode is given in Fig. 4, is used. Note that in the algorithm a node is associated with a part-policy or a full-policy.

The *IBB* procedure can be improved by some simple steps described next.

(i) Clearly, if a perception p is fixed with action a the rule must be $p \rightarrow a$ and the initial part-policy *cPi* can include this. If, through some means or other, actions for any other perceptions are also fixed, the appropriate rules can be added to *cPi*.
(ii) There is no need to select free perceptions (see Definition 3.4), since the value of a part-policy *Pi* which has rules for all constrained perceptions (i.e. an effective policy) is equal to the optimistic value of *Pi*, since there

[1]Logic Programming Associates Ltd., London, UK, (http://www.lpa.co.uk).

```
procedure IBB(c:node, b:node):g:node
//S is a stack of nodes that may need evaluation
//c is an arbitrary initial node with policy cPi
//   cPi usually, but need not be empty
//b is initial guess for best policy with value bv
//returns g with optimal policy for constrained
//   perceptions gPi and value gv
S:= [c];
while S is not empty
  {cs:= top node on S with part-policy Pi;
  p:= a constrained perception not yet fixed in Pi;
  for all actions ai in A(p) compute
    {newPi:=Pi extended by p->ai;
    vi:= optimistic value for newPi;
    }
    if newPi have fixed all constrained perceptions
    then if (vj=max{vi}) and vj> value(b) then
      {value(b):=vj; policy(b)=newPj;
      remove all nodes s in S with value(s)<=vj;}
    else for each newPi
      if (vi>bv) add node to S with policy newPi;
//can add these nodes at front of S in any order
  }
return node with policy bPi and value bv;
```

FIGURE 4. Branch and bound procedure *IBB*.

is no constraint on which action to select for the free perceptions, allowing the action which maximizes the node value to be selected and assuming the optimum actions are taken thereafter for all free perceptions.

(iii) In many cases the iterated values computed by Equation (2) increase monotonically. Therefore, when computing the optimistic value for a part-policy *Pi* if an interim value is greater than *bv* the iterations can be terminated since the computed value could not be less than *bv*.

In case step (ii) above is implemented the returned (effective) policy *bPi* may not be a full-policy, although the returned value *bv* is the value of the optimal policy. The optimal policy can be easily obtained by running *IBB* starting with the effective policy *bPi* and treating all free perceptions as constrained perceptions. As an illustration the tree for Example 2.1, including the efficiency improvements, is shown in Fig. 5. Constrained perceptions are {*a, d, e, h*}, *i* is fixed and {*b, c, f, g*} are free. The initial part-policy is {$d \to$ k, $h \to$ w, $c \to$ l}. An initial estimate is 376 for this policy extended by setting other actions to w.

IBB, with the simple improvements above has been roughly implemented in both Prolog and Java and some experimental results are given in Table 1 and commented on below. The reported tests include two examples from *BlocksWorld* and one example from a *GridWorld* adapted from [19]. Both Examples 3.1 and 3.2 make use of a standard 10-block world including reflexive wander, for which there are 20 perceptions. Of these, the perception (*s*0, *nh*) is fixed with the only possible action of wander, and three perceptions are free, namely (*s*10, *nh*), (*s*9, *nh*) and (*s*9, *h*), leaving 16 constrained perceptions. The parameters used in all cases were $r = -1$, $R = 100$ and $\gamma = 0.9$.

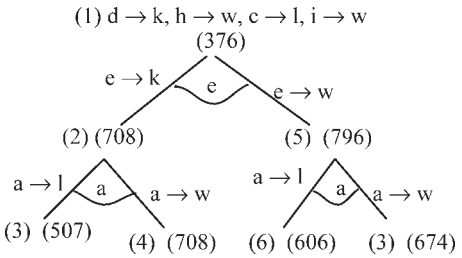

FIGURE 5. Procedure *IBB* policy tree $\mathcal{T}$ for Example 2.1: (1) Initial value is 376, search starts with a or e, choose e; (2) Best value = 708, continue with a; (3) Best value = 507 = new optimum; (4) Best value = 708 = new optimum; (5) Best value = 796 − continue; (6) and (7) are abandoned as 606 < 708 and 674 < 708.

EXAMPLE 3.1. The goal is to see a 10-tower. It is the one that is recognizable by the agent and for which it is fairly easy to construct a likely optimal policy. In fact, the optimal policy is

$$\{s1, nh \to \texttt{k}, s2, nh \to \texttt{k}, s3, nh \to \texttt{k}, s4, nh \to \texttt{k}, s10, nh \to \texttt{stop}, s4, h \to \texttt{l}, s5, h \to \texttt{l}, s6, h \to \texttt{l}, s7, h \to \texttt{l}, s8, h \to \texttt{l}, s9, h \to \texttt{l}\}$$

and all other rules are to wander.

EXAMPLE 3.2. The goal is a multi-goal (seven separate goals) and one which the agent cannot recognize when it is achieved. It is described as 'any state in which there are exactly three towers of equal height (and not also two towers of equal height), and the agent is seeing the surface'. This multi-goal includes the seven single goals ([3, 3, 3, 1], *nh*), ([2, 2, 2, 4], *nh*), ([1, 1, 1, 7], *nh*), ([2, 2, 2, 1, 3], *nh*), ([1, 1, 1, 2, 5], *nh*), ([1, 1, 1, 3, 4], *nh*), ([2, 2, 2, 1, 1, 1, 1], *nh*) and is one, we claim, for which it is not obvious what the optimal policy might be. The optimal policy is, in fact,

$$\{s2, nh \to \texttt{k}, s4, nh \to \texttt{k}, s5, nh \to \texttt{k}, s6, nh \to \texttt{k}, s8, nh \to \texttt{k}, s9, nh \to \texttt{k}, s10, nh \to \texttt{k}, s0, h \to \texttt{l}, s2, h \to \texttt{l}, s6, h \to \texttt{l}\}$$

TABLE 1. Table of results.

Expt	IBB(sec)		FSSB(sec)	Nodes
	J	P	P	*IBB*
Ex3.1a	22	6000	100	385
Ex3.1b	225	N/A		3893
Ex3.2a	20.2	5400	79200	363
Ex3.2b	591	N/A		9483
Ex3.3	0.16	240	N/A	209

and all other rules are to wander. The diffuse nature of the goal situations in Example 3.2, and the large number of situations that are partially similar to one or other of them means that most policies turn out to be pretty bad. For although a policy may be designed to achieve one part of a goal situation, that same policy may also target a similar, but non-goal situation. Even the optimal policy does not reach a goal situation from every situation. For example, in the situation ([1, 1, 1, 1, 1, 1, 1, 1, 1], *nh*) an agent can only wander and so never make any progress. Nevertheless, the optimal policy does not have any trivial 'loops', such as {*s*2, *nh* → `k`, *s*1, *h* → `l`}, which would have an agent in situation (*s*2, *nh*) forever picking up and putting down.

Table 1 gives runtimes in Java and Prolog for two different selection orders for perceptions in *IBB*. It also gives the total number of nodes in *IBB* for which a (optimistic) policy value is computed.

EXAMPLE 3.3. The grid world has 47 squares and is shown in Fig. 6. The agent can move in any of the four compass directions (N,S,E,W) and the goal is marked '*'. An attempted move into a wall results in no movement. There are just seven constrained perceptions, which are indicated by the labels {*a*, ..., *g*} in the figure, which also labels the squares ({11, ..., 47} and shows the directions proposed by the optimal policy.

This example has many free perceptions, giving it a different character from those of *BlocksWorld*.

The optimal policy is shown in Fig. 6. It is not the only one; e.g. the north pointing arrows in the bottom row could be east pointing. Example 3.3 was also run using the parameters $r = 0$, $R = 0$ and $\gamma = 0.9$ and the optimal policy was the same (although it had a different value).

The number of nodes considered by *IBB* depends on the initial estimate of best value, but is very sensitive on the choice of perception *p* selected within the `while` loop at each step (as can be seen from Table 1). The particular implementation of *IBB* required that the perception order for every branch of the tree was the same, and for both Examples 3.2 and 3.3, a good order of the constrained perceptions was found (by trial and error) to be

*s*4, *nh s*3, *nh s*2, *nh s*1, *nh s*0, *h s*1, *h s*2, *h s*3, *h s*5, *nh s*7, *h s*6, *nh s*6, *nh s*4, *h s*5, *h s*8, *nh s*7, *nh*

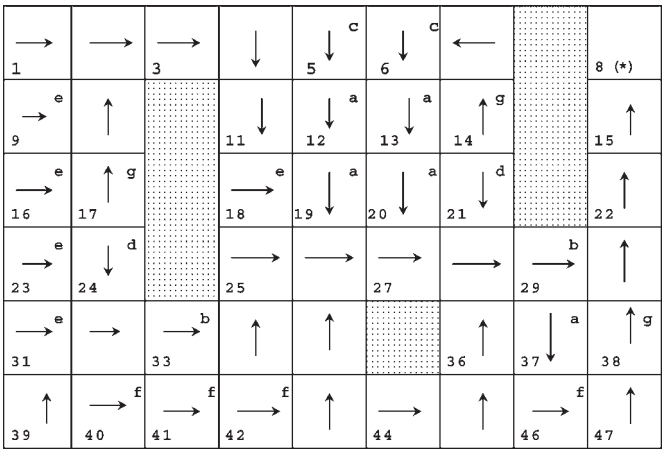

FIGURE 6. Grid for Example 3.3.

In Table 1 J and P abbreviate Java and Prolog, and the node counts are for the Java implementations. Ex3.1a and Ex3.1b use a good order for perceptions in *IBB* and Ex3.1b and Ex3.2b use a bad order. The (longer) Prolog timings for *IBB* and *FSBB* are very approximate and the timings for Prolog for Ex3.1b and Ex3.2b were not found.

Currently we are investigating how to find a good perception order for *IBB*. Another approach makes use of an extension of Dijkstra's algorithm to compute a set of partial policies $\mathcal{C}$, such that any policy *Pi* for which the goal is reachable from at least one situation is a superset of one of the policies in $\mathcal{C}$. For Examples 3.1 and 3.2 $\mathcal{C}$ has size 5 and 97, respectively. $\mathcal{C}$ might be used in *IBB* to choose a good perception order using the notion of Information Gain [20] and for filtering policies evaluated by either *IBB* or *FSBB*.

3.3. Fixed stratification branch and bound (FSBB)

The *FSBB* algorithm generates a tree in which each leaf is a partial policy π, the root being the empty policy {}. A leaf is marked as complete if it commits some action to every possible perception. The leaves awaiting expansion to their child leaves are held in a prioritized queue as determined by some selection function. If a leaf is selected for expansion then it is deleted from the queue and its children are added to the queue.

Every leaf π is accompanied by a lower bound *L* and an upper bound *U* upon the values of all completed policies that are supersets of π, these bounds being calculated when the leaf is created. For the root, $L = Tr$ and $U = R$ where *Tr* is the trough value $r/(1 - \gamma)$. A leaf is selectable for expansion only if $L \neq U$. After each expansion step the queue is pruned to remove any leaf whose upper bound is less than some other leaf's lower bound.

The policy extensions entailed when creating children are determined by a fixed stratification (partitioning) of the situations prior to generating the tree. For $k > 0$ each *kth* stratum, denoted N_k, contains every situation from which the shortest topological path (ignoring the arcs' action labels) in the unrestricted situation graph *G* to a goal has length *k*. Stratum N_0 contains only the goal(s). Such stratification presumes *G* to be connected, which holds for all domains used in this paper. A leaf at depth $k > 0$ in the tree is a commitment of actions to just those perceptions occurring in strata N_j, $j < k$.

The expansion of an incomplete leaf π at depth k generates a child $\pi \cup \pi'$ at depth $k+1$ for each distinct way of choosing a set π' of commitments of actions to those perceptions that occur in N_{k+1} but in no N_j, $j < k$. If there are no such perceptions then π' can be only $\{\}$ so that expanding this π produces a single child π at depth $k+1$. If a complete leaf π at depth k is selected for expansion then (as dictated above) it must have distinct lower and upper bounds. Its expansion consists of creating a single child π at depth $k+1$ whose lower and upper bounds are identically the true value V of π, this being calculated by a standard Bellman evaluation of the entire situation set. Such a child is called a Bellman leaf and—since its bounds are not distinct—is not itself selectable for expansion. After the usual pruning that follows any expansion step the queue must contain at most one Bellman leaf.

Our use of *FSBB* has mostly employed a selection function which keeps the queue size relatively low in order to reduce overheads in the tasks of selection and pruning. If the queue contains no Bellman leaf then this function selects, from the leaves of greatest depth, one with a maximal upper bound. Otherwise, if a Bellman leaf with value V exists, this function selects a leaf whose upper bound U gives a minimal value of $U - V$. Under this selection function the queue must eventually contract to a single Bellman leaf and this must be the optimal complete policy.

The efficiency of *FSBB* depends upon its capacity to prune leaves and hence the subtrees rooted at them, and this depends in turn upon how tightly the bounds assigned to the leaves can be drawn and at what computational expense. Before describing how the bounds are calculated we illustrate the features of *FSBB* described so far using the simple 4-block problem of Example 2.1. The stratification of its graph determines that $N_0 = \{5g\}$, $N_1 = \{5i, 8c\}$, $N2 = \{8h\}$, etc. Figure 7 shows the developing policy tree down to depth 2, as determined by the action choices for the perceptions in those first three strata, together with their bounds and pruning as computed using *FSBB*.

The bounds for a selected leaf π at depth k are calculated in a manner which aims to minimize the time expended on evaluating situations. Specifically, the only situations evaluated using the Bellman formula are those in stratum N_k, so that the only arcs considered in that evaluation are those that emerge from situations in that stratum. In general the subgraph so defined is much smaller than the total graph dealt with by a standard Bellman evaluation of all situations.

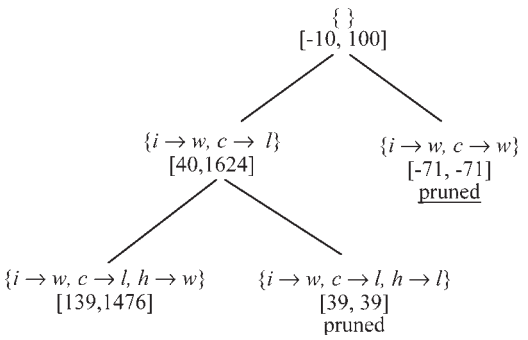

FIGURE 7. *FSBB* policy tree $\mathcal{T}$ for Example 2.1.

The stratification criterion determines that any arc from N_k in the subgraph must be directed either into N_{k-1} or into N_k or into some N_j, $j > k$. Each such arc is assigned a reward rwd(s) representing an estimate of the value of the situation s to which it is directed, to compensate for the fact that s is treated by the subgraph's Bellman evaluation as having value 0 since it has no successors in that subgraph.

This reward estimation for the subgraph exploits a prior estimation of values of situations in N_j, $j > k$. Each such situation s is examined to determine whether or not there exists a path from s to a goal having the property that, wherever it makes a commitment $p \rightarrow \mathtt{a}$, (i) every other commitment it makes to p also chooses a and (ii)if π commits an action to p then that action is a. If s has such a path then it is termed π-good. The quest for such a path is undertaken by depth-first search, optimized using lemmas and loop-avoidance. Additionally, two parameters R_{low_ϕ} and R_{high_ϕ} are needed for the reward estimation; these are inherited from the leaf's parent ϕ and will be defined presently.

Given the above, each estimate rwd(s) in the subgraph is calculated as follows:

for the lower-bounding case:

$$\begin{aligned} &s \in N_{k-1}: && \mathrm{rwd}(s) = R_{\mathrm{low}_\phi} \\ &s \in N_k: && \mathrm{rwd}(s) = r \\ &s \in N_j, j > k: && \mathrm{rwd}(s) = Tr \end{aligned}$$

for the upper-bounding case:

$$\begin{aligned} &s \in N_{k-1}: && \mathrm{rwd}(s) = R_{\mathrm{high}_\phi} \text{ if } s \text{ is } \pi\text{-good} \\ & && \mathrm{rwd}(s) = Tr \text{ otherwise} \\ &s \in N_k: && \mathrm{rwd}(s) = r \\ &s \in N_j, j > k: && \mathrm{rwd}(s) = r + \gamma((1-\gamma^{j-k})Tr \\ & && \qquad + \gamma^{j-k} R_{\mathrm{high}_\phi}) \text{ if } s \text{ is } \pi\text{-good} \\ & && \mathrm{rwd}(s) = Tr \text{ otherwise} \end{aligned}$$

Thus, the subgraph is Bellman - evaluated twice, first to estimate lower bounds for the situations in N_k and then to estimate their upper bounds.

The bounding of the leaf as a whole must also estimate all the other situations. For any situation in N_j, $j > k$, its lower bound is Tr, whilst its upper bound is $(1 - \gamma^{j-k})Tr + \gamma^{j-k} R_{\mathrm{high}_\phi}$ if it is π-good but Tr otherwise. For any situation in N_j, $j < k$ its bounds are the same as were calculated for it in the leaf's parent ϕ. Then, the leaf's overall (lower or upper)

bound (L or U) is the mean of the (lower or upper) bounds of all the situations. With L and U thus determined, our newly created leaf π is attended also by values for R_{low_π} and R_{high_π} (for use when bounding its own children), calculated as follows: if no situation in N_k is π-good then $R_{\text{low}_\pi} = R_{\text{high}_\pi} = Tr$, but otherwise R_{low_π} is the least of the lower bounds of the π-good situations in N_k and R_{high_π} is the greatest of their upper bounds. For the root node, $R_{\text{low}_\pi} = R_{\text{high}_\pi} = R$.

FSBB performs at its best for cases where not many distinct perceptions are concentrated in the first few strata. Thus it solves Example 3.1 in 100 s with Prolog, much faster than *IBB*. By contrast Example 3.2, where there are many perceptions in the first few strata, is solvable by *FSBB* in about 22 h with Prolog, much slower than the best-case run using *IBB*. In examples like this, where there are many perceptions in the first few strata, the policy tree is initially very bushy. Experiments showed that these initial leaves are insufficiently discriminated to allow judicious selection.

3.4. Experimental validation

We tested the quality of the *policy predictor* by using a *Simulator* to simulate agents operating under the various policies. In the introduction it was stated that the simulator is the sole comparator against which the model using situation graphs is evaluated. When the simulator uses exactly those descriptors that appear in the model it precisely simulates traversing the f-restricted situation graph.

We call this *exact modelling*, since the model (situation graph) and the simulated world describe in exactly the same way the world transitions that can occur. This mode of modelling is useful for validating the software, in that the policy values obtained by prediction are expected to agree exactly with those obtained by simulation.

More generally, a simulated world may contain and exploit more detail than that expressed in the model. This is because the desired level of detail in the simulated world, intended to represent some real world, may be too demanding in scale to be expressed practicably in the model. In this mode, which we call *abstract modelling*, the model's situation descriptors are less detailed than those employed by the simulator.

For example, a simulated BlocksWorld might assign towers to cells in a grid. A particular state might have two 1-towers and one 2-tower, each with its own cell position. The descriptor of such a state in the simulator might be a data structure such as [(1,(3,5)),(1,(2,2)),(2,(1,6))]. The descriptor of the agent's perception of this state might be (s(2, 2), nh), denoting that the agent is seeing whatever is in cell (2, 2) and is not holding. This level of detailing makes it possible for the simulator to 'know' that the agent is seeing the particular 1-tower in cell (2, 2). By contrast, the model might use a simpler descriptor such as [1, 1, 2] for the state and a simpler one such as (s1, nh) for the perception, thus forming a more abstract description of the situation. In this description it is not possible to express which particular 1-tower is being seen by the agent. The abstraction gap between the two levels of detailing is such that events capable of being distinguished in the simulator cannot be distinguished in the model. For instance, if the agent now wanders in the simulation to acquire the perception (s(3, 5), nh) then it is again seeing a 1-tower but a different one from that previously seen. This model, however, does not represent this detail in the situation graph.

A simulation traverses part of an *implicit* situation graph $\mathcal{G}^W$ which, if made explicit, would employ these more detailed descriptors to represent the situations of the simulated world W. The part traversed, denoted $\mathcal{G}_f^W$, is the f-restriction of $\mathcal{G}^W$ as determined by the policy f. Reference to this implicit graph enables us to state some useful principles governing the correctness of our modelling with the (generally) simpler and explicit graph G_f.

3.4.1. *Modelling principles*

Let f be a policy for the TR-application $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ and G_f be an f-restricted situation graph. Let $\langle \mathcal{O}^W, \mathcal{P}^W, \mathcal{A}^W \rangle$ be the corresponding simulated application and $\mathcal{G}^W$ be the simulated situation graph and $\mathcal{G}_f^W$ its f-restriction.

Let TR be the TR-application $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ and G_f be an f-restricted situation graph for TR. Then the following principles hold:

Soundness of actions in G_f: Every action $a \in \mathcal{A}$ models a simulated action in $\mathcal{A}^W$.
Completeness of actions in G_f: Every simulated action in $\mathcal{A}^W$ is modelled by some action $a \in \mathcal{A}$.
Soundness of situations in G_f: Every situation $(o, p) \in \mathcal{O} \times \mathcal{P}$ models a situation in $\mathcal{G}^W$.
Completeness of situations in G_f: Every situation in $\mathcal{G}^W$ is modelled by some situation $(o, p) \in \mathcal{O} \times \mathcal{P}$.
Soundness of arcs in G_f: Every arc in G_f corresponds to an arc in $\mathcal{G}_f^W$.
Completeness of arcs in G_f: Every arc in $\mathcal{G}_f^W$ is modelled by some arc in G_f.

Informally, in exact modelling the correspondence between situations in G_f and $\mathcal{G}^W$ is one-one, whereas in abstract modelling this is not so, since the G_f is a further abstraction of the simulated world.

The simulator operates for a single agent in the following way. A single run begins with the agent assigned to some chosen situation s in G_f. In the case that s corresponds to several situations in $\mathcal{G}_f^W$, one of those situations is chosen randomly. The simulator then drives the agent's subsequent activity in a stepwise fashion, in each step making it perform the action dictated for its current perception by the policy f, thus updating its simulated situation. In the case of a non-deterministic action the next simulated situation is chosen randomly from those that are possible. The agent implicitly traverses a path in G_f from s. The run terminates when the agent either reaches the goal or exceeds a prescribed

bound B on the number of transitions performed. As the path is traversed, the value of the run is computed incrementally on the same basis as used in the *policy predictor*. The value of a run of n steps is given by the formula

$$V = r_1 + r_2\gamma + \cdots + r_{n-1}\gamma^{n-2} + r_n\gamma^{n-1}$$

where r_i is the reward for the ith step. If the agent reaches a trough situation, then the final value of the run will always be $r/(1-\gamma)$ and so the run can be terminated prematurely without any loss of information. Equal numbers of runs are executed for each initial situation s. The mean of observed values V over all runs for all s then gives the observed policy value $V_{\text{obs}}(f)$. For exact modelling and a G_f with no loops it can be shown by induction on the maximum number of steps in G_f that the Principles of Formulation imply that $V_{\text{obs}}(f)$ is equal to $V_{\text{pre}}(f)$. This property uses the fact that, in exact mode, each arc for a non-deterministic action taken from a situation s has equal probability. When G_f has loops a similar result holds, namely that the expected value of $V_{\text{obs}}(f)$ approaches $V_{\text{pre}}(f)$, as the maximum number of steps allowed in a simulation run (i.e. the bound B) tends to infinity. For abstract modelling these properties cannot hold in general since the situation graph is less detailed than the behaviour of the simulator. In particular, the probabilities on the arcs for a non-deterministic action can only be estimated in the model.

The simulator also reports the observed success rate $SR_{\text{obs}}(f)$ for the policy, measured as the percentage of runs that reach the goal. The success rate can be predicted independently of the simulator by considering the reachability of the goal in G_f. By definition, there cannot exist any arc in G_f directed from T_f to N_f. However, there may exist one or more in the opposite direction, in which case we describe G_f as *NT*-bridged. The *policy predictor* has the secondary function of determining N_f, T_f and the *NT*-bridged status for any policy f.

If G_f is not *NT*-bridged then the agent can reach the goal if and only if its initial situation is in N_f, so that its predicted success rate $SR_{\text{pre}}(f)$ (as a percentage) is exactly $\lambda = 100 \times |N_f|/|\mathcal{S}|$. If G_f is *NT*-bridged then we have only the weaker relationship $SR_{\text{pre}}(f) < \lambda$. In either case λ provides an upper bound on $SR_{\text{pre}}(f)$, which we may then compare with $SR_{\text{obs}}(f)$. Simulated runs curtailed by the bound B may not attain a reachable goal. So in general $SR_{\text{pre}}(f)$ will overestimate $SR_{\text{obs}}(f)$ by an extent that depends on B.

For a set $\mathcal{F}$ of n policies we can measure the correlation between their observed and predicted values as follows. A pair $(f, g) \in \mathcal{F} \times \mathcal{F}$ (distinct f, g) for which $V_{\text{pre}}(f) \leq V_{\text{pre}}(g)$ is *concordant* if $V_{\text{obs}}(f) \leq V_{\text{obs}}(g)$, but is otherwise *discordant*. If C is the number of concordant pairs and D the number of discordant pairs, then the *Kendall rank-correlation coefficient* [21] $\tau_{\mathcal{F}}$ for $\mathcal{F}$ is $2 \times (C-D)/(n \times (n-1))$. We can re-express this measure as a percentage $Q_{\mathcal{F}} = 50 \times (1 + \tau_{\mathcal{F}})$. In the best case $Q_{\mathcal{F}} = 100\%$, when the predicted and observed ranks of all policies agree perfectly. In the worst case $Q_{\mathcal{F}} = 0\%$, when they disagree maximally. The $Q_{\mathcal{F}}$ values cited in the case studies we report here all imply, with $> 99.75\%$ confidence, that the observed and predicted policy values are correlated.

To visualize the correlation of predictions with test outcomes for a set $\mathcal{F}$ of n policies, those policies' observed values are charted against the ranks of their predicted values (observed policy values are measured along the vertical axis, and predicted ranks along the horizontal one). Overall predictive quality is reflected by the extent to which the chart exhibits a monotonically decreasing profile. For Example 2.1 if the observed policy values, $V_{\text{obs}}(f)$, are charted against the ranks of their predicted ones, $V_{\text{pre}}(f)$, the chart obtained, shown in Fig. 8, decreases monotonically, showing the predicted ranks to be in accordance with the ranks obtained by simulation. The Kendall measure $Q_{\mathcal{F}}$ is 99.56% in this case and from 1007 simulated runs per policy (thus, 53 for each initial situation) with bound $B = 100$, the same two policies are identically observed as optimal.

When computing the predicted policy values using the predictor program we suppress any arcs emergent from the goal in order to mirror the simulator's behaviour in terminating a run when the agent reaches the goal.

Although the prediction parameter values used throughout are $R = 100$, $r = -1$ and $\gamma = 0.9$, we determined by prior experiments that, provided that $R \gg r$ and $0 < \gamma < 1$, other choices would not have altered the results or conclusions reported here. Similarly, allowing the value of r to vary from one transition to another, particularly for transitions that enter the trough, might better distinguish between various 'bad' policies. However, the nature of the discounted-reward formula is that policies that reach the goal at all have much higher values than those that do not, and we were primarily interested in the former policies. The contribution to the policy value of trough nodes depends on the closeness of such situations to the start node.

3.5. Predictive quality

As explained above, in the single agent case the policy rankings obtained from the predicted policy values for exact modelling were very close to those obtained from the observed policy values obtained from a simulation. The small variations occur in graphs with loops, since the predictor

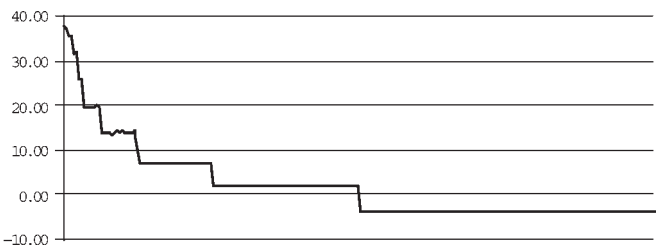

FIGURE 8. Observed policy values in Example 2.1 (exact mode).

uses the infinite horizon formula whereas the exact simulator uses a finite horizon given by the bound B on the maximum number of steps.

In the case of abstract modelling the predicted policy values show more volatility with respect to the values obtained by the simulator. This is due to the fact that the f-reduced situation graph G_f used by the policy predictor is only a model of the situations and transitions used by the simulator. There are several ramifications of this abstraction, which are illustrated by considering how the `w` action is treated in a grid-based simulation of *BlocksWorld*. The key decision for the `w`-action is what the agent shall see afterwards. We decided this as follows. A `w`-action moves the agent randomly to any vacant cell in its (immediate) neighbourhood (comprising eight cells, unless the agent is next to a grid boundary). The agent then sees the surface if its new neighbourhood is entirely vacant, but otherwise randomly sees any one tower in that new neighbourhood. This decision fixes the probability distribution over the possible `w`-transitions between the prior situation and its successors. The *policy predictor* may attempt to estimate these probabilities or may default them to be equi-probable.

The above consideration shows that we must expect some shortfall in the predictive power of the policy predictor when applied to a position-sensitive world, and the question is how serious (or not) that shortfall is. We simulated Example 2.1 on a 6×6 grid with a bound B on the number of steps increased to 200 to allow for more wandering. The predicted optimal policies are the same as before, but with a slightly reduced predicted value 30.47 (reduced precisely because of the negative rewards of reflexive wanders). However, they are not quite optimal among the observed values. The observed optimal policy is a little different:

$$a \to \mathtt{w},\ b \to \mathtt{l},\ c \to \mathtt{l},\ d \to \mathtt{k},\ e \to \mathtt{w},$$
$$f \to \mathtt{w},\ g \to \mathtt{k},\ h \to \mathtt{w},\ i \to \mathtt{w}$$

On seeing a 2-tower when not holding (perception e), it marginally favours `w` over `k`. Figure 9 shows the ranking chart, for which $Q_{\mathcal{F}} = 66.91\%$. Minor sampling variations among many policies of similar value are the root cause of this reduced value. If we contract $\mathcal{F}$ to just the 20 best policies then $Q\mathcal{F} = 76.32\%$, so the predictive quality is rather better in the region that matters.

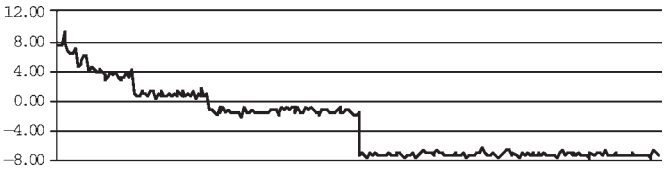

FIGURE 9. Observed policy values in Example 2.1 (abstract mode).

The observed values in the chart are much lower than the predicted ones, since the simulated agent is wandering (in the sparse regions of the grid) much more than the prediction considers probable. However, it is the relative rather than the absolute values that are our main interest. The observed and predicted success rates remain in close agreement, since excess wandering does not alter the reachability of the goal, but merely delays its discovery.

In predicting the policy values for the above grid-based example we used only default probability distributions. The volatility in Fig. 9 would have been reduced had we applied better estimated probabilities. Rather than writing specific software to make such estimates for a grid-based *BlocksWorld*, we can more easily demonstrate here the scope for improvement by exploiting results from the simulator. We made the simulator count the number of times each arc in the f-reduced SG was traversed (by a corresponding transition in $\mathcal{G}^W$) in order to estimate the relative frequency of non-deterministic arcs. These frequencies were used to better estimate probabilities in the policy predictor. The results using these better probability estimates for the best 18 policies of Fig. 9, superimposed with the original results for the same policies, are shown in Fig. 10, whose chart is significantly smoother than the corresponding region of the chart in Fig. 9.

4. MULTIPLE AGENTS

In any TR-application there can be one or more agent types with one or more agents of each type. Agents of the same type are called *clones*. Whether or not it is useful to employ multiple agents in the pursuit of a goal depends upon the nature of the world and of the agents' interactions with the world and with each other. With limited perceptions, incognizance of the goal and lack of communication, simple TR-agents of the kind we have considered may cooperate advantageously only by serendipity. Predicting accurately the behaviour of multiple agents presents not only analytical difficulties, in attempting to assess the overall impact of their interactions, but also problems of scale—for just a modest case of a 4-block world with two agents there are potentially more than 13 000 policies to consider.

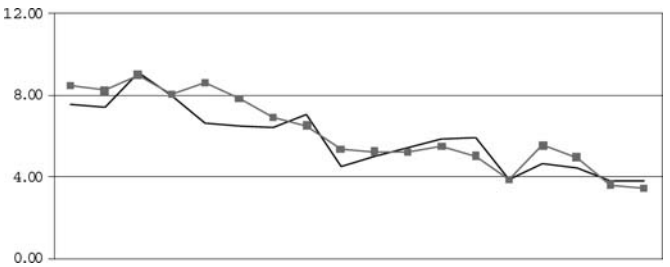

FIGURE 10. Improved observed policy values in Example 2.1 (abstract mode).

This section considers how our framework can be adapted to plan for a community of agents. In this paper we restrict agents to be clones, that is to have identical policies. The framework is flexible enough to deal with applications in which every agent has the same capabilities and follows the same policy, through agents with the same capabilities but following a different policy (for example by virtue of having differing goals), to agents with different capabilities.

4.1. Exogenous actions

Our use of unrestricted situation graph G has so far assumed just one agent. When dealing with several agents we again use the situation graph representing the point of view of a single archetypal agent called *self*. Every transition experienced by the agent must have some corresponding arc in the situation graph. When an agent is alone these transitions are caused only by its own actions, but if other agents are present then the transitions it experiences may be due to their actions. A new action x for this kind of exogenous activity is needed to label these other transitions. An x-arc is interpreted as *self* waiting.

Such an arc may be drawn from any situation $s1$ to any other $s2$ when we wish to entertain the possibility that another agent effects the transition from $s1$ to $s2$. Therefore *self*'s action set now contains x, which opens the possibility for a policy to prescribe act x. If *self*'s policy prescribes action x when in situation $s1$ the effect is to make *self* wait, until and if its perception is changed by some other agent.

When *self*'s policy prescribes some action other than x, *self* may alternatively undergo an exogenous transition caused by the action of another agent. We call this *passive updating* of *self*.

If an exogenous agent behaves unpredictably then, in general, it is hard to obtain a significant predictive benefit. By contrast, if this other agent is predictable—in particular, is itself an agent with a known policy—then it can be exploited to advantage in circumstances where *self* acting alone would be inefficient in achieving the goal or be unable to reach the goal at all. Our consideration of exogenous actions from now on will therefore concentrate upon multiple-agent scenarios in which the policies of exogenous agents are either known or are to be determined. Actions that arise from serendipitous actions of the environment are not considered when finding or evaluating policies. Such actions are, by their nature, presumed rare and so their consequences will not have a large impact on the policy values. However, if such actions were more probable then they would perhaps be better modelled by an explicit agent.

4.2. Policies for multiple agents

The key question we address in this section is whether a graph focusing upon one agent enables prediction of good policies for a group of agents without requiring explicit analysis of all the combinations of the situations they occupy. First we explain how such a graph is constructed and the role played by the x-action. The notion of a TR-application is already general enough to deal with multiple agents. If $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{R} \rangle$ is a TR-application, then all the agents in R will be clones and have identical policies unless otherwise stated. The set R can be represented by a positive integer indicating the number of such cloned agents. The set $\mathcal{A}$ of action repertoires will include both the `wander` and the `wait` actions.

The changes to unrestricted graphs to include the effects of several agents are:

- the additional agents may affect the set of objective states $\mathcal{O}$, and possibly the perceptions $\mathcal{P}$ of *self*; however, since these are dependent only upon the capabilities of a single agent, it is less likely they will be affected, unless, for example, the agents are themselves upgraded to be able to perceive other agents. The change in $\mathcal{O}$ could be larger; however, unless *self* needs to identify the other agents by name, states can be described using implicit anonymous references to those agents;
- from each situation there may be additional arcs for exogenous actions by other agents that will affect the situation of *self*. These additional arcs are labelled by x and are interpreted from *self*'s point of view as a `wait` action.

In *BlocksWorld*, for example, instead of requiring states that describe *self* and three other agents holding a block, it may be sufficient to record that *self* is holding a block and that *some* other agent is also doing so. Which one, or how many, may not be important. Already, this feature allows us to represent situations involving several agents compactly. For instance, instead of the situation ([1, 2, 3], (s0, *nh*), (*s*3, *h*), (*s*3, *nh*)), indicating seven blocks and three agents, two of which are seeing the 3-tower whilst the third is seeing the surface, it might be sufficient to represent the situation as ([1, 2, 3], (s3, nh)), assuming *self* is seeing the 3-tower. The perceptions of the other two agents are not relevant to the design of *self*'s policy except insofar as they may offer opportunities for *self* to be passively updated—for example, the agent holding a block may place it on the 3-tower resulting in *self* being passively updated to seeing a 4-tower.

The unrestricted graph formed for *self* will be called a *self graph*, $\mathcal{G}^s$, [13], which is a projection onto *self* of an *implicit*, much larger graph, called the *group graph*, G^g. $\mathcal{G}^g$ represents situations as described above, namely by a tuple including the objective state and one perception for each agent. The relation between $\mathcal{G}^g$ and $\mathcal{G}^s$ is formalized in Definition 4.1. $\mathcal{G}^g$ represents situations as described above, namely by a tuple including the objective state and one perception for each agent.

DEFINITION 4.1. *Let $\mathcal{G}^g$ be a group graph for n agents, based on the set $\mathcal{S}_g$ of situations of the form $(o, p_1, \ldots, p_n)$ and having*

the set of transitions $\mathcal{T}_g$. The self graph $\mathcal{G}^s$ is the graph for self = k obtained from $\mathcal{G}^g$ as follows. The situations of $\mathcal{G}^s$ are projections of those in $\mathcal{G}^g$ and have the form (o, p_k). The set of transitions $\mathcal{T}_s$ in $\mathcal{G}^s$ is given by $\mathcal{T}_s = \{((o, p_k), (o', p_k))\}$ for each transition $(o, p_1, \ldots, p_k, \ldots, p_n)$ to $(o', p'_1, \ldots, p'_k, \ldots, p'_n)$ in $\mathcal{G}^g$. The action labelling a transition is x *if the transition is not due to the action of agent k, otherwise it is the action taken by agent k.*

The situations from which the self graph is constructed are exactly those in which *self* could possibly find itself—in other words those situations (o, p) which are consistent with all those situations (o, p') in which other agents could be, given that the objective state is o and that *self* has a perception p. Notice that the group graph would include many 'equivalent' situations. For example, the situation ([1, 2, 3], (s0, *nh*), (s3, *h*), (s3, *nh*)) could occur in six different ways by permuting the perceptions of each agent. Two of those permutations would map to ([1, 2, 3], (s3, *h*)) and two more would map to ([1, 2, 3], (s0, *nh*)). Moreover, the self graph should include exactly those transitions which *self* could make from any situation. Transitions which are due to *self* taking an action a ≠ x appear also in $\mathcal{G}^s$ labelled by the same action a, whilst those due to some other agent acting will appear in $\mathcal{G}^s$ as action x. In the above example, if the policy were to pick when not holding and seeing a 3-tower, there would be an x-arc from ([1, 2, 3], (s3, *h*)) to ([1, 2, 2], (s2, *h*)), corresponding to the possibility of an agent other than *self* being in situation ([1, 2, 3], (s3, *nh*)) and performing the k action. This transition corresponds to several transitions in the group graph: for example, assuming *self* is the second agent, such transitions would include ([1, 2, 3], (s3, *nh*), (s3, *h*), (s0, *nh*)) to ([1, 2, 2], (s2, *h*), (s2, *h*), (s0, *nh*)) and ([1, 2, 3], (s1, *nh*), (s3, *h*), (s3, *nh*)) to ([1, 2, 2], (s1, *nh*), (s2, *h*), (s2, *h*)). If the perception of *self* remains unchanged by such an action there is a reflexive x-arc in the self graph.

An x-arc in the graph can thus represent two different notions. On the one hand it indicates how *self* can be impacted by the actions of others. On the other hand it may represent the action of deliberate waiting in accordance with *self*'s own policy. In this case the transitions still represent the impact on *self* of the actions of others.

The group graph is never actually constructed. The presence of x-arcs can be detected directly from the policy. To see what this means, consider the situation ([1, 2, 3], (s3, *nh*)) and the x-arc transition to ([1, 2, 2], (s2, *nh*)) taken from the above example. This transition occurs as it is *enabled* by the policy. That is, when *self* is in situation ([1, 2, 3], (s3, *nh*)) there could be another agent in a situation which, according to the policy, will cause *self* to move to ([1,2,2],(s2,nh)). In this case, assuming the policy is to pick when seeing a 3-tower, the transition is enabled since another agent could be in the situation ([1,2,3],(s3,*nh*)). Notice that there are also some group situations when this transition could not occur, for instance, if neither of the other agents was seeing the 3-tower. If the policy prescribes a different action for perception (s3,*nh*) then the transition to ([1,2,2],(s2,*nh*)) could *not* occur, for in order to remove the top block of the 3-tower an agent would have to be seeing (s3,*nh*)) and the policy should choose the k action. But that is not so in this case and the x-arc would not be present.

The number of policies that need to be considered is somewhat larger for agent communities than for a single agent, due to the extra possibility of the x action. For instance, for Example 2.1 there were 256 possible policies, whereas if there are two agents there are more than 13 000 policies. In practice, most of those policies have low value. This raises the general problem of economically identifying low-value policies. We can apply various heuristic filters to this problem; for example, an intuitive filter removes those policies which afford poor prospects of reaching the goal. Another filter might remove policies which present the potential for deadlock. As with all heuristics these filters are fallible, in that they may be expensive to compute and may reject some worthwhile policies.

We will now illustrate some of the issues involved using a multi-agent version of the 'deadlock filter', which we call the *clone consistency principle* and is defined in Definition 4.2. This is a useful filter since it not only reduces the possibility for deadlock, but also reduces very significantly the number of policies requiring to be examined. The clone consistency principle seeks to eliminate policies with the following pathology. The policy chooses x for some perception and in some situation $s = (o, p)$ there are no x-arcs enabled by the policy other than a reflexive arc. In this case an agent in situation s will necessarily wait forever. All other agents would also necessarily be in state o, but possibly with perceptions different from p. Because *self* has no enabled x-arcs, no actions made by other agents can change the state, so all agents are locked into state o. If they had the same perception as *self* then deadlock would ensue. For instance, in the 7-block example above, if the action on seeing a 1-tower and not holding is x, then *self* must wait in the situation ([1, 1, 1, 2, 2], (s1, *nh*)). If all other agents also happen to be seeing 1-towers, then all would have to perform wait and there would be deadlock. However, this situation is not necessarily deadlocked since some agents could, on the other hand, be seeing a 2-tower and perhaps the action in that case is k, giving rise to an x-arc to the situation ([1, 1, 1, 1, 2], (s1, *nh*)). Although *self* would still perform x, the action prescribed for the agent other (now having perception (s1, *h*)) may be to wander and place the held block on the 1-tower being observed by *self*, in which case *self*'s perception would change to (s2, *nh*) and it would be able to perform k.

In what follows a *valid group*, also called a *multi-situation*, is a set of situations $\{(o_1, p_1), \ldots, (o_n, p_n)\}$ derived from the situation $(o, p_1, \ldots, p_n)$ in the group graph. In other words, a *multi-situation* is a *physically possible* assignment of the

agents to situations that share a common state and differ (if at all) only in perceptions.

DEFINITION 4.2. *Let f be a policy containing the rule* $p \to$ x. *Then the clone consistency principle requires that, for each situation* (o, p), *there is a non-reflexive* x*-arc to a state* o', G_f^s *must have an arc from* (o, q) *to* (o', q'), *f must contain* $q \to$ a, a $\neq$ x *and the situations* (o, p) *and* (o, q) *must belong to a valid group*.

EXAMPLE 4.1. For the examples in the remainder of this section and in the next section we use a different world scenario, consisting of planks and two or more cloned agents. A plank may be held at either end by one agent or at both ends by two different agents. In the latter case it can be disposed of using the dispose action, denoted by di, which reduces the number of planks by 1. Agents may otherwise wander, or lift or drop one end of a plank. If only one end of a plank is being held and the agent holding it attempts a dispose action, then that action will leave the agent's situation unchanged.

The states, perceptions and actions we use to model such a world containing two planks and two agents are shown in Fig. 11. Each state is represented by a tuple $[r, t, f]$, where r is the number of (raised) planks held at both ends, t the number of (tilted) planks held only at one end and f the number of (flat) planks held at neither end. Each agent may perceive whether it sees an unheld end (*su*), a held end (*sh*) or no end (*s*0), and whether it is holding an end or not.[2] This model is an abstraction of *Planks World* as represented in the simulator, in that it contains no detail concerning which ends of which planks are perceived by the agents. The presumed goal is the situation (0, *a*) in which there are no planks and all agents are therefore neither seeing nor holding any plank.

Each $A(p)$ set now contains the exogenous action x. The graph G has 16 situations and there are 36 possible policies, of which just 20 are clone-consistent. It is shown in Fig. 12. Notice particularly the x-arc from situation (7, *f*), which corresponds to the passive update of *self* when another clone initiates a di action.

Consider a clone *r*1 governed by the policy

$$a \to \texttt{w}, \quad c \to \texttt{li}, \quad e \to \texttt{x}, \quad f \to \texttt{dr}$$

In the situation (5, *e*) there is one flat plank and one tilted plank and *r*1 is not holding but is seeing the held end. For this perception the policy requires that *r*1 shall wait. In due course another agent *r*2 holding the held end might effect the transition from (5, *e*) to (4, *c*), that is, might drop that end. From *r*2s point of view, this is a transition from (5, *f*) to (4, *c*), requiring a drop action. The clone consistency principle states that the policy requires $f \to$ dr.

[2]It is assumed that if an agent is holding an end then it can see a held end.

	$o \in \mathcal{O}$ [r, t, f]
0	[0, 0, 0]
1	[0, 0, 1]
2	[0, 1, 0]
3	[1, 0, 0]
4	[0, 0, 2]
5	[0, 1, 1]
6	[0, 2, 0]
7	[1, 0, 1]

	$p \in \mathcal{P}$	$O(p)$	$A(p)$
a	s0, nh	{0,1,2,4,5}	{w,x}
c	su, nh	{1,2,4,5}	{li,w,x}
e	sh, nh	{2,5}	{w,x}
f	sh, h	{3,5,6,7}	{dr,di,x}

FIGURE 11. States, perceptions and actions (Example 4.1).

The best policy for the stated goal is

$$a \to \texttt{w}, \quad c \to \texttt{li}, \quad e \to \texttt{w}, \quad f \to \texttt{di}$$

This policy happens not to require any agent to wait. On the other hand, a suboptimal policy that does require waiting, such as

$$a \to \texttt{x}, \quad c \to \texttt{li}, \quad e \to \texttt{w}, \quad f \to \texttt{di}$$

can be both clone-consistent and yet admit deadlock. It admits deadlock because both agents may be in (1, *a*). It is clone consistent because, in every situation an agent may occupy, some other agent could act so as to change the former's situation. For instance, if one agent is in (1, *a*) then the other agent may be in (1, *c*) and would perform a li action causing the former agent to change its situation to (2, *a*).

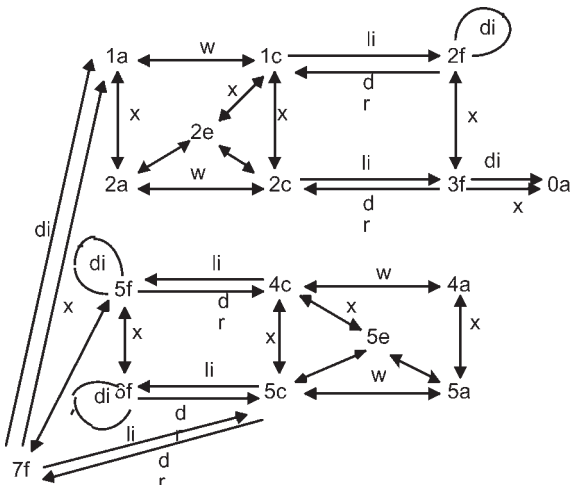

FIGURE 12. Unrestricted situation graph (Example 4.1).

4.3. Changes to the policy predictor

The policy predictor estimates the probability distribution of non-deterministic transitions in the single agent case. In the multiple agent case it must also estimate the probability distribution for x-arcs, which will depend in part on the number of additional agents that are to effect those transitions.

When there are $n > 1$ agents, the predictor assigns probabilities to the arcs of the graph G_f as follows. Each node $s = (o, p)$ has emergent arcs for the action a that *self* performs according to the rule $p \rightarrow$ a in its policy f. In the case that a $\neq$ x the emergent arcs from s comprise both a-arcs and x-arcs (denoting passive updating). The normalized relative probabilities of the a-arcs from $s$are either estimated or defaulted as equi-probable. The normalized relative probabilities of the x-arcs from s are likewise determined.

The absolute probability distribution at s is then computed by multiplying the relative probability of each a-arc by $1/n$ and multiplying the relative probability of each x-arc by $(n - 1)/n$.

Consequently, if Σx and Σa denote the sums of the absolute probabilities of the x-arcs and of the a-arcs, respectively, their relationship is that $\Sigma\mathtt{x} + \Sigma\mathtt{a} = 1$ (to renormalize) and $\Sigma\mathtt{x} = (n - 1)\Sigma\mathtt{a}$. The latter reflects the supposition that it is $(n - 1)$ times more probable that any agent will be passively updated (by some other agent acting) than that it will itself act. This matches the behaviour of our serialized simulator which, in each step, randomly chooses one agent to act and then passively updates the other $(n - 1)$ agents.

Accurate estimation of x-arc probabilities would be achievable in principle by exhaustive analysis of the complete group graph. However, the very purpose of the self graph is to obviate explicit construction of the group graph. In our own studies we have estimated x-arc probabilities in only a limited fashion by forming the groups explicitly and using them to make approximate counts of the potential transitions between them.

As an illustration, suppose in Example 4.1 that the chosen policy is

$$a \rightarrow \mathtt{x}, \quad c \rightarrow \mathtt{li}, \quad e \rightarrow \mathtt{w}, \quad f \rightarrow \mathtt{di}$$

There are then three x-arcs emanating from situation $(5, f)$, one to $(7, f)$, one to $(6, f)$, due, respectively, to a second agent being in $(5, c)$ and either lifting the same plank as *self* or the other plank, and one reflexive x-arc dueto the second agent wandering (from $(5, e)$), or proactively (i.e. according to the policy) waiting (from $(5, a)$). The probability on the di arc from $(5, f)$ would be computed to have a value of 0.5, whilst the probabilities on each of the non-reflexive x-arcs would be 1/8, whilst that on the reflexive x-arc is 1/4.

4.4. Changes to the simulator

In this section the operation of the simulator in a multi-agent environment is described. In this context the simulator effects transitions between the multi-situations used in the group graph.

A single run of the simulator begins with the agents assigned to some chosen multi-situation s in G^g and, as with the single agent case, if s corresponds to several simulated multi-situations then one of those is chosen randomly. Following this initialization the simulator executes steps until either the goal is reached (by some agent), or the simulation depth bound B is reached, when the run terminates. The value of the run is calculated in the same way as for the single agent case. Execution of a step involves several operations

(i) an agent is randomly selected as the one to act;
(ii) this agent performs the action prescribed by the policy and its own situation is accordingly updated;
(iii) the situations of all other agents are appropriately passively updated.

If the action performed in (ii) is a pro-active wait (x-action), the step merely leaves all agents unaltered.

For example, a step from the multi-situation $(4, c, c)$ in Example 4.1 might be: $r1$ is selected, its policy (say) dictates action li, and the new multi-situation would be either $(5, c, f)$ or $(5, f, e)$, depending on whether $r2$ is seeing the same end that $r1$ has just lifted or not. The situation change for $r2$ in this transition is an example of a passive update corresponding to an x-arc from either $(4, c)$ to $(5, c)$ or from $(4, c)$ to $(5, e)$.

The simulator's successive random choosing of which agent acts next provides an adequate approximation to the more realistic scenario in which they would all be acting concurrently. Owing to the physical constraint that there can be only one world state at any instant, any set of concurrent actions that produced that state can be serialized in one way or another to achieve the same outcome. The simulator's randomness effectively covers all such possible serializations.

As in the single agent case, the simulator can take advantage of recognizing when a particular multi-situation is in the group trough, provided the group graph is feasible to explore.

5. COMMUNICATING AGENTS

The extension of the framework to allow several agents operating at once has so far excluded any communication between them. If non-communicating agents appear to cooperate in achieving a task then this is merely a fortituitous manifestation of emergent behaviour, which we call 'as-if' co-operation. In this section we show how deliberate (planned) cooperation can be obtained by enabling agents to communicate. We avoid the need to devise special languages and protocols for this by restricting the communicable elements to be perceptions of the kind already employed. If there is an atomic perception p then we can allow *self* to have the atomic perception kp

(representing 'knows' p), whose meaning is that one or more other agents are perceiving, and communicating to *self*, that p is true. We assume that the content of p is instantly transmissible from those other agents to *self* by some suitable broadcasting mechanism. With this provision in place, policy formation for r_i can then take account of what it receives from other agents in addition to its own direct perceptions of the world.

Consider again the two-plank two-agent problem of Example 4.1. Neither agent can perceive what the other is seeing, and so cannot distinguish between states 6 and 7 when holding one end of a plank. This means the `dispose` action is liable to be unsuccessful. Adding additional agents, but maintaining their limited perception, increases the chance of success when attempting a `dispose` action. Some of the unsuccessful `dispose` attempts can be avoided if some communication between agents is allowed.

We can allow for agents to communicate to each other, in a limited way, by giving them certain useful percepts of the same form as those *self* could have. In this example, we choose to let an agent perceive whether no other agent is holding a plank end (*knh*), or whether at least one other agent is doing so (*kh*).

EXAMPLE 5.1. This example extends Example 4.1 in two ways: (i) by allowing communicated perceptions, and (ii) by allowing either two or three cloned agents. The particular states and perceptions for this formulation for two agents are given in Fig. 13. The *self*-restricted graph G_f for two agents and the policy

$$a \to \texttt{x}, \quad b \to \texttt{w}, \quad c \to \texttt{li}, \quad d \to \texttt{li}, \quad e \to \texttt{w},$$
$$f \to \texttt{x}, \quad g \to \texttt{di}$$

is shown in Fig. 14. Note that reflexive `x`-arcs are omitted to avoid clutter.

In case there are three agents, the unrestricted graph has an additional state, namely $8 = [1, 1, 0]$, i.e. the case in which one plank is raised and the second tilted, and nine extra situations, namely $(3, b)$, $(3, e)$, $(6, b)$, $(6, d)$, $(6, e)$, $(7, b)$, $(7, d)$, $(7, e)$ and $(8, g)$ and numerous additional `x`-arcs. The simulator was run for the problems of disposing of two planks with either two or three agents. The particular policy illustrated in Fig. 14 was ranked by the predictor as second for both the 2-agent case and the 3-agent case. The simulator ranked the policy, respectively, as fourth and third.

More significantly, the observed (simulated) values of the policy were, respectively, 18.33 and 30.55. Informally, the policy is to `wait`, rather than `wander`, except when there is a chance some agent may be holding the other end of a plank, in which case the policy is to `wander`. If the agent finds itself holding an end, it waits for another agent to lift the remaining end, ready for a `dispose` action. When there are more agents this policy has a better chance of success, hence the higher observed value (30.55) for three agents. The charts for the two-agent and three-agent case are shown in Figs 15 and 16, respectively.

For non-communicating agents, the closest policy to that in Fig. 14 is

$$a \to \texttt{x}, \quad c \to \texttt{li}, \quad e \to \texttt{w}, \quad f \to \texttt{di}$$

which is now shown for two agents in Fig. 17. This offers few situations the opportunity to reach the goal and has a poor observed policy value 8.70. By contrast, the use of communication as reflected in Fig. 14 confers upon many more nodes the possibility of reaching the goal and has a correspondingly higher observed policy value, as noted earlier, of 18.33. The

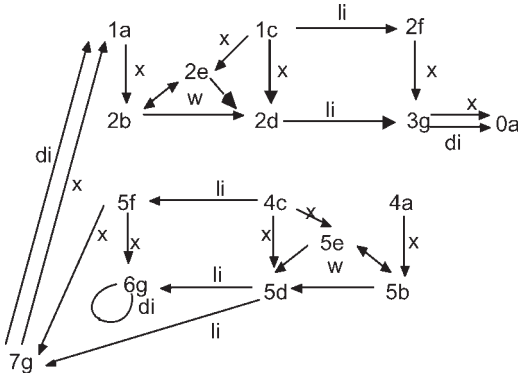

FIGURE 14. Policy graph G_c for 2 agents (Example 5.1).

	$o \in \mathcal{O}$ [r, t, f]
0	[0, 0, 0]
1	[0, 0, 1]
2	[0, 1, 0]
3	[1, 0, 0]
4	[0, 0, 2]
5	[0, 1, 1]
6	[0, 2, 0]
7	[1, 0, 1]

	$p \in \mathcal{P}$	$O(p)$	$A(p)$
a	s0, nh, knh	{0,1,4}	{w,x}
b	s0, nh, kh	{2,5}	{w,x}
c	su, nh, knh	{1,4}	{li,w,x}
d	su, nh, kh	{ 2,5}	{li,w,x}
e	sh, nh, kh	{2,5}	{w,x}
f	sh, h, knh	{2,5}	{dr,x}
g	sh, h, kh	{3,6,7}	{di,dr,x}

FIGURE 13. States, perceptions and actions (Example 5.1).

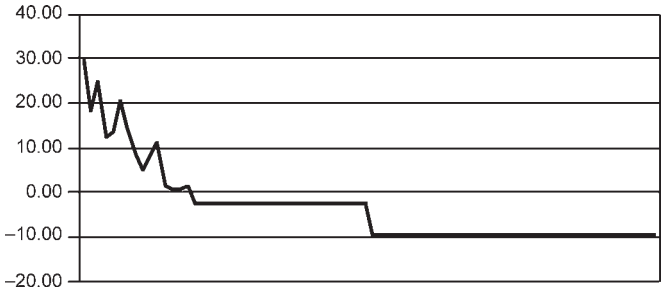

FIGURE 15. Policy chart for two agents (Example 5.1).

communication has allowed *self* to distinguish between situations 1*a* and 2*b* and consequently to form a different action for the two situations, leading to an improved policy.

Using communication in this manner is semantically equivalent to the alternative of simply increasing the ability of an agent to perceive by its own means more of the state. Viewed in this way, we could have cast the new perceptions *a*: (*s*0, *nh*, *knh*) and *b*: (*s*0, *nh*, *kh*) as

$$a : (s0, nh, nh1) \quad \text{and} \quad b : (s0, nh, h1)$$

where *h*1 and *nh*1 stand, respectively, for the other agent is, or is not, holding, on the assumption that an agent was physically equipped to know what the other agent was holding. In engineering terms, however, it is more practical to broadcast perceptions through one uniform technology than to equip agents with a diverse range of sensors. For more than one agent this approach introduces more complex perceptions, including disjunctions, so although the two views are equivalent, we prefer the one we interpret as communication.

Both the self-graphs of Examples 4.1 and 5.1 demonstrate an *incoherence* effect due to the presence of x-arcs. A path may exist using certain x-arcs, each of its corresponding transitions being possible, but the group structure of agents required for the first transition does not lead to the group structure necessary for the next transition. We call this phenomenon *group incoherence*. For instance, in Example 5.1 and for two agents, consider the policy

$$a \to \texttt{x}, \quad b \to \texttt{w}, \quad c \to \texttt{li}, \quad d \to \texttt{li}, \quad e \to \texttt{x},$$
$$f \to \texttt{dr}, \quad g \to \texttt{di}$$

and the apparent path through G_f from (2, *e*) to (0, *a*) via (1, *c*), (2, *f*) and (3, *g*) taking actions by *self* of x, li, x, di. Assume *self* is agent *r*1 and the other agent is *r*2. In situation (2, *e*) *r*1 is looking at a held end that it is not itself holding. It would enter (1, *c*) if *r*2 could drop the end whilst being in (2, *f*). *r*2 would then enter situation (1, *c*) as well, both agents seeing the same end of the plank. *r*1 would then lift the end and move to situation (2, *f*), while *r*2 would be passively updated to (2, *e*). It is clear there is no way for *r*1 to get to (3, *g*), for this would require *r*2 to lift the other end of the plank to the one *r*1 is holding. This would require *r*2 to be in situation (2, *d*) rather than (2, *e*).

There does not seem to be any easy way to avoid the effects on policy values due to group incoherence when present, although the use of communication means that agents can be more aware of others and be able to distinguish between apparently identical world states, as was illustrated for Examples 4.1 and 5.1 discussed earlier. An analysis of the relation between a group graph and a corresponding self graph is given in [13].

In future work we will investigate a modification to the Bellman formula of Definition 3 which, for a given policy *f*, makes the value of a situation in the self graph not only depend upon the expected reward of its successor situations, but also take into account its predecessor situations. In particular, when a successor situation s' of a situation s occurs via an x-arc the modification will consider whether, given a predecessor situation b of s, another agent could be in a perception to effect the x-arc.

That is, assuming *self* is agent 1 and $s = (o, p_1)$, $s' = (o', p'_1)$ and $b = (c, d_1)$, if the group graph would have had arcs from some multi- situation $(c, d_1, \ldots, d_k)$ to $(o', p'_1, \ldots, p'_k)$ via $(o, p_1, \ldots, p_k)$, whether the second arc projects onto the x-arc in the self graph, and the first projects onto some arc in the self graph from situation b to situation s. If this is the case, then a more reliable estimate of the probability distribution from s to its successors can be made and the value of the policy f obtained using the self graph will be better predictor of the value of policy f. This is similar to an improvement described in [14].

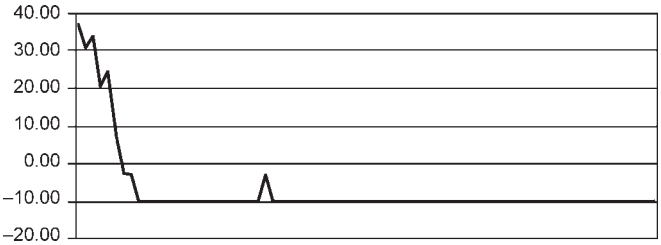

FIGURE 16. Policy chart for three agents (Example 5.1).

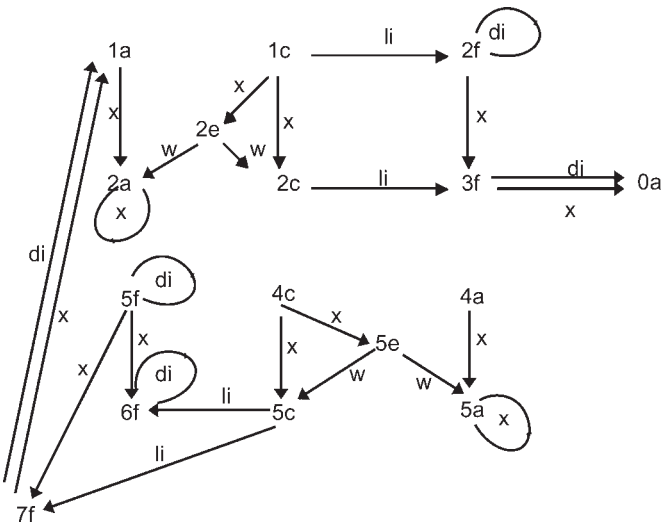

FIGURE 17. Policy graph for two (non-communicating) agents (Example 4.1).

6. MEMORY AND PERCEPTION

In this section we explore the idea of interpreting memory as a percept. We illustrate this by considering the star-finding problem taken from [22].

EXAMPLE 6.1. The problem concerns an agent that can move left or right through a linear sequence of four cells to look for a star which is always in the same place, in this case at cell 3. The set of states $=\{1, 2, 3, 4\}$ and each perception is modelled as a triple $[o, a1, a2]$, where o is either 'e(mpty)' or 's(tar)' and $a1$ and $a2$ are the two most recent actions, either 'move left' (L) or 'move right'(R). In this representation the first element o is the agent's observation and the remaining elements constitute a memory of several past actions.

If the agent's behaviour is to stop when it sees the star and stay put when attempting to move left(right) from cell 1(4) then the predicted optimal policy is

$$[e, L, L] \rightarrow R, \quad [e, L, R] \rightarrow R, \quad [e, R, L] \rightarrow L,$$
$$[e, R, R] \rightarrow L, \quad [s, -, -] \rightarrow stop$$

This is shown in Fig. 18 and corresponds to the policy in [22]. Some situations shown in Fig. 18 are not reachable from any other, e.g. (2,[*e*, *L*, *R*]), but they are included since they might exogenously become the current situation through corruption of memory.

The optimal policy (for the star in cell 3) can be given an algorithmic interpretation, which is 'Move right until reaching either the star or the right-hand cell, then move left until reach the star'. If the star is closer to the left-hand cell the dual policy that first moves left is better. If the position of the star is not static, then either policy is the best. It is worth noting that the calculation using the SGF is particularly simple since the determinism of the actions obviates the need for probability estimation.

This is a POMDP problem and so typically solved using belief state estimation, as applied in [10, 22]. There, the vector [0.25, 0, 0.5, 0.25], for instance, represents that the agent believes it is more likely to be in state 3 than in state 1 or 4 and definitely is not in state 2. In order to obtain an optimal policy using this representation an iterative algorithm is used to find an approximation by computing $V(b) = \max_{a \in A} (r + \gamma \times e)$, where r is the expected reward for action a from belief state b and e is the estimated expected value of successor states, using the current estimates. In [23] it was shown that the optimal value function can be represented using some finite set of vectors V, such that the value of a node is computed as $V(b) = \max_{v \in V} b.v$. In [10, 22] it is shown how a policy can be found using this result.

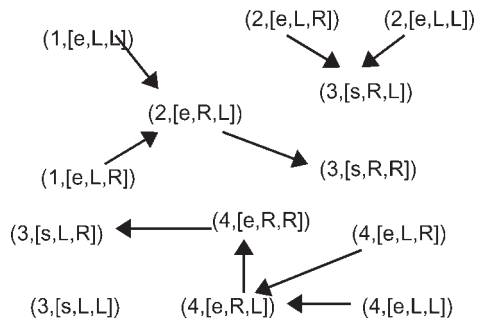

FIGURE 18. G_f for optimal policy of Example 6.1.

Belief states in this context are essentially an encoding of the past. As illustrated in Example 6.1 an alternative treatment of memories is to interpret them as percepts. The method of belief states clearly relies upon the assumption that the world changes only under the actions of the agent. Otherwise, there would be no reason why the agent should assume its previous belief state is any guide to the recent past and hence to the current world state. For example, if several agents are acting then this assumption is not valid.

For multi-agent contexts our use of the *self*-oriented x-arc appears simpler and more scaleable than the corresponding 'decentralized' POMDP approach of [24], which has to analyse conjointly how all the agents are acting upon the world.

Due to exogenous actions an agent's memory may not reflect facts true of the current world. One way to recover from such a case is to allow an agent to forget, which could be modelled by arranging for an agent's memory to revert to some neutral value after a certain number of steps. For example, in *BlocksWorld* an agent might remember the height of the tallest tower it had seen, but such a memory may be incorrect if that tower were built upon or dismantled by another agent. If the memory reflected a true perception of the world it is likely it would be reinforced by the agent either seeing the tallest tower again or an even taller one. Otherwise, the memory could be forgotten after a number of steps, freeing the agent from believing incorrect facts of the world. This is implicitly how noisy perceptions are dealt with. If a wrong perception is made, an agent will soon obtain another perception that will be correct, assuming that the level of noise is not very great.

7. POSITIONING WITH RELATED WORK

In this Section we outline some ways in which the SGF differs from some alternative methods of optimal policy determination.

The first contrast is with the work of those who seek to design agents equipped with internal reasoning resources such as knowledge bases, theorem-provers and planners [25] and—in multi-agent contexts—communication mechanisms [26]. Such facilities enable agents to reason about their world, about their experiences and about the consequences and merits of their possible courses of action and interaction. It is intuitive that, armed with such powerful capabilities, they should be generally more successful in achieving their goals

than primitive agents able only to map perceptions directly to actions. The price potentially paid for this, however, lies in the physical complexity of the hardware required to accommodate and operate those facilities. As indicated in Section 1, our own wish is to minimize that requirement in order to serve those application contexts where such minimality may be physically imperative.

In particular, we can differentiate our framework from that of deliberative agents as follows. A deliberative agent generally follows a sense-think-act cycle [27]. During the 'think' part of the cycle the agent will evaluate what it has sensed together with its previous beliefs and, if necessary, update those beliefs and propose an action based on those updated beliefs. The so-called BDI deliberative agents [28] have more structure, in that in addition to beliefs B they have a current goal D and an intention I which is essentially its current plan to achieve D. To our knowledge the deliberation activity does not usually quantify over the whole space of policies, but rather attempts only to find the best policy with respect to the current beliefs.

In our framework, the 'think' part of the cycle is carried out off-line. It takes a goal and a situation graph model and finds from them the policy P_{best} that has the best discounted-reward estimate averaged over all start situations. This policy is at least as good as the following, intuitively appealing, policy P_{good}: from the current perception cp assume that the best policy from any state consistent with cp has first action cp_a and choose cp_a as the next action, then continue in the same way.

Notice that for each observed perception p the policy P_{good} will always pick the same action p_a, as long as the agent cannot distinguish which of several possible situations that all have perception p it is actually in. Therefore, P_{good} is one of the policies considered in our framework for finding P_{best}. In fact, P_{best} must be at least as good as P_{good}, since if it were not, then for some situations $\mathcal{S}$ and percepts p_S for which P_{best} and P_{good} differed, the value of P_{best} would not be optimal, counter to assumption.

The differences in approach between policy finding in the SGF and for deliberating agents (DA) can be briefly summarized as: (i) the evaluation of policies is made off-line in SGF and on-line in DA, and (ii) the DA can take into account more structured beliefs. However, as we showed in Sections 5 and 6, SG can take into account communication between agents and memory by giving agents enhanced perceptions, which to some extent overcomes the limitations of basic agents.

The second contrast is with the work of those who seek to optimize relatively simple agents, comparable with our own, by the use of POMDPs where the agents cannot perceive the entirety of the state [10, 22, 24]. Both cases ultimately desire the agent to determine its next action on the basis of its current perception. Both the POMDP and situation graph design frameworks acknowledge the relevance of state to this decision. However, POMDP considers state during the design process in terms of beliefs about the state. The outcome is a policy that is represented by a tree whose nodes and arcs are, respectively, labelled by actions and perceptions. As a result, the action that an agent takes for a given perception depends on where it is in the tree. This, combined with standard techniques of mathematical programming, yields algorithms capable of identifying tree-structured policies that are optimal or near-optimal relative to one's ability to estimate probabilities given the agent's assumed powers of state observation.

The SGF uses complete knowledge of the state in its design process and the outcome is that an agent decides its next action on the basis of perception alone. Policies of the latter kind have been termed *memoryless* to signify that their use does not entail remembering past actions [18, 29].

For both frameworks, the more the state becomes directly observable, the more they take on the character of an MDP. The POMDP methods become complicated to apply, however, in the multi-agent context where the updating of each agent's beliefs has to consider the combinatorial impact of the other agents' actions upon the state. These complications become compounded further if the agents are having to survive in a world potentially impacted by unpredictable exogenous events. These are the kinds of context for which we wish to design our agents. We have presented evidence that in those contexts our method, whilst by no means perfect in its predictive accuracy or free from its own complexity concerns, is nonetheless comparatively simple to apply and yields reasonably good policies for the given goals.

The third contrast is with the particular species of TR-agents envisaged by Nilsson in [1, 3]. Nilsson's policy structure differs from ours in that its rules are assumed to be ordered, which lays the basis for his so-called *regression property*. This requires that the effect of any rule $p \rightarrow a$ shall be to cause the condition of some earlier rule in order to become satisfied and that p shall be weaker (i.e. implied by) the condition of any other rule having that same effect. Additionally, these arrangements must lead ultimately to the achievement of the goal. Nilsson's design process therefore relies on assuming that states—in particular the goal state—are totally observable. In this architecture the content and ordering of the rules constituting the desired policy can then be inferred by a reductive planning process that constructs and orders rules in such a way that the operation of each one may suitably enable the operation of others, the whole intended to ensure that the goal state eventually becomes achievable.

The regression property combines considerations that are unrelated to one another—the capacity of rules to enable the conditions of other rules, and the significance of rule ordering. In our framework the disposition of rules to become enabled and of their actions to promote progress towards the goal is not enforced *a priori*, but is instead expected to emerge only as a natural outcome of the design process. Moreover, our

treatment attaches no goal-related significance to rule ordering. Indeed, the ordering of rules is immaterial if perceptions are pairwise mutually exclusive. For us, the only benefit of rule ordering is that it caters for the suppression within any rule of those perceptual conjuncts known, by default, to be satisfied by virtue of some earlier rule having not been enabled with their logical complements satisfied. It is, therefore, merely the analogue of the default 'negation-by-failure' rule employed to confer programming economy in formalisms like Prolog. Altogether, our view is that the regression property is overly restrictive as an *a priori* constraint upon agent design.

A further consequence of Nilsson's agents' greater percipience is that it favours the use of hierarchical policies in which actions can themselves be policies with sub-goals. Whenever a policy is executed as an action, in order for it to have any effect under normal circumstances the perception that triggered it must be knowable by the agent long enough to achieve the policy's subgoal. For example, in his *BlocksWorld* there might be a rule $clear(A) \rightarrow makeclear(A)$ where $makeclear(A)$ is a parametrized call to another policy. Since Nilsson does not require his agent to be observing a tower in order to know that its top block (say $\mathcal{A}$) is not clear, the agent can busy itself in removing blocks from above $\mathcal{A}$ for as long as it is known that $\mathcal{A}$ is not clear. On the other hand, the ability only to perceive direct percepts, as generally assumed in our framework, means that in many, if not most cases an action is likely to falsify the condition of the rule that triggered it.

The fourth contrast is with those methods [20, 31–33] that rely upon learning, that is, upon training agents to perform well in simulated environments. Here the evolving experience of the agent is effectively translated into merit-oriented weightings of the alternative actions available to each perception. The outcome is typically a non-deterministic policy allowing the agent to choose, for its current perception, between alternative actions according to the weightings, which may be interpreted as the relative probabilities of those actions being the best to perform. Among the methods used for learning such policies is inductive logic programming, employed to determine—by a combination of logical inference and quantitative support—advantageous associations between actions and their consequences upon the state,as far as perceptual observability permits [2, 34]. All such methods differ from our own in their reliance upon a training regime and, usually, in the structure of the policies they produce. Like POMDPs they can be very successful, although it is not yet clear how their overall complexity compares with that of POMDPs or with that of our own framework employing just a static model of the agent's possible behaviours.

8. EXTENSIONS TO THE FRAMEWORK AND FURTHER INVESTIGATIONS

In this paper we have restricted policies to be functions. That is, each perception is associated with exactly one action. This is in contrast to *relational (or stochastic) policies*, in which a perception may be associated with more than one action and the agent chooses between them according to some probability distribution. We have also assumed that all agents in a group follow the same policy. However, our framework does not rule out either the possibility of relational policies or the modelling of a group of non-homogeneous agents. Some preliminary experiments show that a relational policy which linearly combines two or more functional policies can be better than any of the functional policies and that two agents following different policies may complement each other. In particular, we will explore the following. First, we shall consider groups of non-homogeneous functional agents having differing capabilities and/or perceptions. For this we would like to discover principles for evaluating the efficacy of these co-implemented non-cloned agents. Secondly, we shall apply the framework to relational policies in the expectation that it will entail no significant changes to the framework.

Our focus on simple agents containing only a functional policy is motivated partly by anticipation of application domains demanding minimal agent hardware and partly by the desire to explore agent capability incrementally, starting with the simplest architectures. A policy of the kind we have considered can be viewed as a compiled consequence of some theory *Th* about the behaviour of the agent. If made explicit, *Th* would either be embedded within the agent or else be deployed only externally in the design process. In the former case the agent would possess sufficient on-board processing capability to accommodate *Th* and to elicit, using some internal reasoning system, the consequences of *Th* needed to determine the run-time behaviour. This is the general nature of an autonomous cognitive agent. Our future work will examine how our design framework can deal with progressive enhancements to the assumed agent architecture, such as bounded memory, subgoal-reduction, relational mapping of percepts to actions and other cognition-related extensions.

Finally, mechanisms by which an agent may learn to improve its policy are more suited to finding relational policies since the linear coefficients may be estimated in standard ways (for example using an artificial neural network and back-propagation). An investigation of such adaptive agents is an exciting area for future work.

9. CONCLUSIONS

In Section 1 we stated that one of our aims in this paper was to dissect the various issues involved in policy optimization for minimal agents capable only of partially observing situations

and under functional restrictions. This dissection has contributed to the development of the SGF.

We conclude by restating the other main contributions: first, two new branch and bound type algorithms for improving the search for optimal policies; second, our method of accommodating exogenous behaviour, including multi-agent interaction, solely through the special x-action and its associated clone-consistency principle and an empirical demonstration of its predictive merits and third, our method of supporting inter-agent communication solely through suitable enrichment of perceptions.

All of these provisions extend substantially the power of the framework without requiring any extension of the agents' basic structure or any additional mechanisms for their implementation.

ACKNOWLEDGEMENTS

The authors are grateful to Michael Cook for the Java programs, which he wrote during his undergraduate summer placement in 2007 while supported by an EPSRC student bursary. They also thank the anonymous referees for their helpful comments.

REFERENCES

[1] Nilsson, N.J. (1994) Teleo-Reactive Programs for Agent Control. *J. Artif. Intell. Res.*, **1**, 139–158.

[2] Benson, S. (1995) Inductive Learning of Reactive Action Models. *Proc. 12th Int. Conf. Machine Learning*, Tahoe City, California, pp. 47–54.

[3] Nilsson, N.J. (2001) Teleo-Reactive Programs and the Triple-Tower Architecture. *Electron. Trans. Artif. Intell.*, **5**, 99–110.

[4] Broda, K., Hogger, C.J. and Watson, S. (2000) Constructing Teleo-Reactive Robot Programs. *Proc. 14th European Conf. Artificial Intelligence (ECAI-2000)*, Berlin, pp. 653–657.

[5] Brooks, R.A. (1991) Intelligence without Reason. *Proc. 12th Int. Joint Conf. Artificial Intelligence (IJCAI-91)*, Sydney, Australia, pp. 569–595.

[6] Kowalski, R.A. and Sadri, F. (1999) From logic programming to multi-agent systems. *Ann. Math. Artif. Intell.*, **25**, 391–419.

[7] Fok, C.-L., Roman, G.-C. and Lu, C. (2006) Agilla A Mobile Agent Middleware for Sensor Networks, Washington University in St. Louis, Technical Report WUCSE-2006-16.

[8] Marsh, D., Tynan, R., O'Kane, D. and O'Hare, G.M.P. (2004) Autonomic Wireless Sensor Networks. *Eng. Appl. Artif. Intell.*, **17**, 741–748.

[9] Wooldridge, M.J. (2000) *Reasoning About Rational Agents*. MIT press.

[10] Kaelbling, L.P., Littman, M.L. and Cassandra, A.R. (1998) Planning and acting in partially observable stochastic domains. *Artif. Intell.*, **101**, 99–134.

[11] Sutton, R.S. and Barto, A.G. (1998) *Reinforcement Learning: An Introduction*. MIT Press.

[12] Sabbadin, R. (2002) Graph Partitioning Techniques for Markov Decision Processes Decomposition. *Proc. 15th European Conf. Artificial Intelligence*, Lyon, France, pp. 670–674.

[13] Broda, K. and Hogger, C.J. (2005) Abstract Policy Evaluation for Reactive Agents. *SARA-05, 6th Symp. Abstraction, Reformulation and Approximation*, Springer, LNAI 3607, Edinburgh, UK, pp. 44–59.

[14] Broda, K. and Hogger, C.J. (2007) Abstraction as a Tool for Multi-Agent Policy Evaluation. *ASAMI'07, Symp. Artificial Societies for Ambient Intelligence*, *Proc. AISB'07*, Newcastle, UK, pp. 20–26.

[15] Broda, K. and Hogger, C.J. (2005) Determining and verifying good policies for cloned teleo-reactive agents. *Comput. Syst. Sci. Eng. CSSE*, **20**, 249–258.

[16] Chades, I., Scherrer, B. and Charpillet, F. (2003) Planning Cooperative Homogeneous Multiagent Systems Using Markov Decision Processes. *Proc. 5th Int. Conf. Enterprise Information Systems (ICEIS 2003)*, pp. 426–429.

[17] Guestrin, C., Lagoudakis, M. and Parr, R. (2002) Coordinated Reinforcement Learning. *Proc. 19th Int. Conf. Machine Learning, ICML-02*, Sydney, Australia, pp. 227–234.

[18] Littman, M.L. (1994) Memoryless Policies: Theoretical Limitations and Practical Results. *Proc. 3rd Int. Conf. Simulation of Adaptive Behaviour*, MIT Press, pp. 297–305.

[19] Loch, J. and Singh, S. (1998) Using Eligibility Traces to find the Best Memoryless Policy in Partially Observable Markov Decision Processes. *Proc. 15th Int. Conf. Machine Learning*, pp. 323–331.

[20] Mitchell, T. (1997) *Machine Learning*, McGraw Hill.

[21] Snedecor, G.W. and Cochran, W.G. (1972) *Statistical Methods*. Iowa State University Press.

[22] Cassandra, A.R., Kaelbling, L.P. and Littman, M. (1994) Acting Optimally in Partially Observable Stochastic Domains. *Proc. 12th National Conf. AI (AAAI-94)*, Seattle, USA, pp. 183–188.

[23] Sondik, E.J. (1978) The optimal control of partially observable Markov processes over the infinite horizon: discounted costs. *Oper. Res.*, **26**, 282–304.

[24] Nair, R., Tambe, M., Yokoo, M., Pynadath, D. and Marsella, M. (2003) Taming Decentralised POMDPs: Towards Efficient Policy Computation for Multiagent Settings. *Proc. 18th Int. Joint Conf. Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, pp. 705–711.

[25] Kowalski, R.A., Sadri, F. and Toni, F. (1998) An Agent Architecture that Combines Backward and Forward Reasoning. *CADE-15 Workshop on Strategies in Automated Deduction*, Lindau, Germany, pp. 49–56.

[26] Clark, K.L. and McCabe, F.G. (2004) Go!—A multi-paradigm programming language for implementing multi-threaded agents. *Ann. Math. Artif. Intell.*, **41** 171–206.

[27] Pfeifer, R. and Scheier, C. (1999) *Understanding Intelligence*. MIT Press.

[28] Rao, A.S. and Georgeff, M.P. (1992) An Abstract Architecture for Rational Agents. *Proc. 3rd Int. Conf. Principles of Knowledge Representation and Reasoning*, Cambridge, Massachusetts, pp. 439–449.

[29] Singh, S., Jaakkola, T. and Jordan, M.I. (1994) Learning Without State-Estimation in Partially Observable Markovian Decision Processes. *Proc. 11th Int. Conf. Machine Learning*, New Brunswick, New Jersey, USA, pp. 284–293.
[30] Kersting, K., Van Otterlo, M. and de Raedt, L. (2004) Bellman goes Relational. *Proc. 21st Int. Conf. Machine Learning*, Banff, Canada, pp. 465–472 ACM.
[31] Dickens, L. (2004) Learning through exploration, MSc Dissertation, Department of Computing, Imperial College.
[32] Kochenderfer, M. (2003) Evolving Hierarchical and Recursive Teleo-Reactive Programs Through Genetic Programming. *EuroGP 2003*, LNCS 2610, Essex, UK, pp. 83–92.
[33] Ryan, M.R.K. and Pendrith, M.D. (1998) An Architecture for Modularity and Re-Use in Reinforcement Learning. *Proc. 15th Int. Conf. Machine Learning*, Madison, Wisconsin, USA, pp. 481–487.
[34] Benson, S. and Nilsson, N.J. (1995) Reacting, Planning and Learning in an Autonomous Agent. Furukawa, K., Michie, D. and Muggleton, S. (eds.), *Machine Intelligence 14*, Clarendon Press, Oxford.

APPENDIX

The finite horizon formula given in Definition 3.2 can be computed iteratively using the appropriate transition and reward matrices, where P_{su} is an entry in the *probability matrix* $\mathcal{P}$ and Υ_{su} is an entry in the *reward matrix* Υ. For a given policy f, its value is given by

$$V(s,f,k) = \Sigma_{u\in SS}(\gamma \times P_{su} \times V(u,f,k-1)) + \Sigma_{u\in SS}(P_{su} \times \Upsilon_{su})$$

The inner product term $\Sigma_{u\in SS}\ (P_{su} \times \Upsilon_{su}) = \langle P_s \cdot \Upsilon_s \rangle^T$ is also called the reward from s. The *value matrix* $V(f, k)$ can be computed iteratively from the n-ary vector $V(f,0) = [0, \ldots, 0]^{\mathrm{T}}$, where n is the number of situations in the f-restricted graph, and the formula

$$V(f,k) = \gamma P \times V(f,k-1) + \langle P \cdot \Upsilon \rangle^T$$

in which $\langle P \cdot \Upsilon \rangle^T$ is the n-dimensional vector of inner products $\langle P_s \cdot \Upsilon_s \rangle$. Because the value of $\langle P \cdot \Upsilon \rangle^T$ is a constant, here denoted by C, iterative expansion yields

$$V(f,i) = (\gamma^{j-1}P^{i-1} + \cdots + \gamma P + I)C$$

which converges for $0 < \gamma < 1$. In the limit, as k tends to ∞, $V(f, k)$ tends to $V(f) = (I - \gamma P)^{-1}C$, or $V(f) = \gamma P \times V(f) + C$. Therefore

$$V(s,f) = \Sigma_{u\in SS}(\gamma P_{su} \times V(f)_u) + \Sigma_{u\in SS}(P_{su} \times \Upsilon_{su})$$

which is the same as the formula for $V(s, f)$ given in Definition 3.1.

The rate of convergence of $V(f, k)$ to $V(f)$ depends upon the value of γ and to a lesser extent upon the topology of the graph G_f.