Designing Effective Policies for Minimal Agents

Krysia Broda and Christopher J. Hogger

Department of Computing, Imperial College London
South Kensington Campus, London SW7 2AZ UK
{kb,cjh}@doc.ic.ac.uk
Technical Report 2006/3

Abstract. 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 and definite associations of the agent's perceptions with states. The combinatorial burden that would potentially ensue from such associations can be ameliorated by suitable use of abstractions. The framework allows one to plan for a number of agents by focusing upon the behaviour of a single representative of them. It allows for varied behaviour to be modelled, including communication between agents. Simulation results presented here indicate that the method affords a good degree of scalability and predictive power.

1 Introduction

This paper presents a framework for constructing and evaluating goal-oriented policies for particularly simple (i.e. minimal or near minimal) 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 this section we discuss why we study these agents and how we position our work in relation to other treatments of agent design, before proceeding to give an overview of our framework and of the paper.

1.1 Motivation

Our focus on very simple agents is motivated chiefly by emerging application contexts, such as nanomedicine and 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 environment together with some associated specified perception(s) for the agent.

We use the term *teleo-reactive policy* to signify a policy that has been designed with some particular goal in mind and, correspondingly, the term *teleo-reactive agent* (TR-agent) to signify an agent governed by such a policy. The latter term was first introduced in [26] and further developed by N. J. Nilsson in, for example, [1] and [27] but with a more specific meaning. Later in this section we contrast Nilsson's agents with our own.

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 [27] 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 [3]; 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 environmental state, whether a goal state or otherwise. In particular, a policy contains no rule specifically associated with a goal. 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 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 [8] whose actions have no overall motivation, and wholly deliberative agents [21] 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.

Apart from the framework itself, the paper makes two main contributions, both concerned with scalability and approximations. The framework is not limited to investigations of policies of a single agent and the first contribution concerns a method to deal with joint policies for several agents that does not need to explicitly consider the state of every agent, but which instead focuses on a single representative and treats the actions of other agents as exogenous behaviour from the standpoint of the representative. We show empirically that predictions of the values of policies obtained using such an approximation, whilst not being exact, are nevertheless broadly in line with the values obtained by simulation. Where there are discrepancies, we have explained their source, and make a suggestion for improving the approximation, which shows good promise in preliminary tests.

The second contribution uses abstractions to deal with large environments. A simplistic evaluation of policies gives rather random predictions, but by considering the origin of the randomness an improvement is suggested, which again gives good results in preliminary tests. Evaluation of policies makes use of Bellman's equation (see Section 4), which evaluates each situation s in which an agent can find itself in terms of the reward reaped by moving to a successor of s. When abstract situations are employed, it may be the case that apparently possible sequences of transitions are in fact not possible, and their contribution to a policy's value can be considerable. By evaluating a situation s in terms of the rewards reaped in two steps from s, better approximations of policy value are obtained, since impossible paths can be detected and eliminated.

1.2 Positioning

In introducing our framework it will be useful to outline at the start some of the ways in which it differs from some alternative approaches to agent design.

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 [20] and – in multi-agent contexts – communication mechanisms [12]. Such facilities enable agents to reason about their environment, 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 above, our own wish is to minimize that requirement in order to serve those application contexts (*e.g.* in nanomedicine) where such minimality may be physically imperative.

The second contrast is with the work of those who seek to optimize relatively simple agents, comparable to our own, by the use of Markov Decision Processes (MDPs) or – when the agents cannot perceive the entirety of the state – Partially Observable MDPs (POMDPs) [9, 18, 25]. The key assumption made in these design methods is that one's beliefs about the agent's current state can be inferred on the basis of its previous action together with one's beliefs about its previous state, thence enabling a suitable next action to be chosen. This assumption, combined with standard techniques of mathematical programming, yields algorithms capable of identifying policies that are optimal or near-optimal relative to one's ability to estimate probabilities given the agent's assumed powers of state observation. These methods are very successful when the above key assumption holds. They become much more 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 an environment potentially impacted by unpredictable exogenous events. These are the kinds of context for which we wish to design our agents. We shall present 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.

Typically the POMDP design process results in policies that map (perception, action) pairs to actions, in contrast to policies which simply map perceptions to actions. Policies of the latter kind are termed *memoryless* to signify that their use does not entail remembering past actions [22, 30]. Insofar as our policies confirm to this form it motivates the use of a different design method,

The third contrast is with the particular species of TR-agents envisaged by Nilsson in [26, 27]. 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 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 inferred by a reductive planning process that constructs and orders rule 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. Like the MDP method, Nilsson's is able to avoid explicit consideration of all possible policies in pursuit of a suitable one whilst, unlike the POMDP method, it does not lend itself to contexts of partial observability.

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 only expected to emerge 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 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 a sub-goal. 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 sub-goal. For example, in his *Blocks World* there might be a rule $\neg clear(A) \rightarrow$ "make clear(A)" where $make - clear(A)$ is a parameterised 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 A) is not clear, the agent can busy itself in removing blocks from above A for as long as it is known that A is not clear. On the other hand, the ability only to perceive direct percepts, as generally assumed in our framework, means that in most realistic cases an action is likely to falsify the condition of the rule that triggered it.

The fourth contrast is with those methods [13, 14, 19, 24, 29] 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 quantititative support – advantageous associations between actions and their consequences upon the state, as far as perceptual observability permits [1, 2]. 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.

1.3 Overview of the Framework

Given this positioning, 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 a world 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 examples later in the paper will be a very simple application that involves just four blocks and three actions, yet potentially offers many thousands of policies to consider. The inclusion of state information in the nodes of the graph does not influence this number, but does increase further the burden of assessing policies. Each possible policy is some restricted subgraph of the primary one, the restriction being that the arc(s) emergent from a node shall be 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 we consider in this paper, a node will typically have several emergent arcs to different situations for the same action, which we will 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 it is advantageous also to assign rewards/penalties 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 one having greater value, under this scheme, than all other policies.

This paper demonstrates our framework with examples and, in particular, it considers the various aspects which address its scalability. One way of aiming for scalability is through the use of abstractions, which we first explored in [5]. This employs generic descriptors denoting classes of perceptions or of states or of both to reduce substantially the burden of policy assessment without sacrificing too much by way of predictive accuracy.

Another way is to employ precise or heuristic filters capable of excluding certain classes of policy from need of evaluation.

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 [4, 6]. 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 interdependent 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 [3], is somewhat similar in motivation to [10] in which the focus is upon MDP-based design, and to [16] 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 situation graphs. 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.

The material is organized as follows. The basic framework is presented in Section 2 and the method 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. In Section 6 we introduce the notion of generic situations. A comparison, with examples, with methods that solve the problem of perceptual aliasing by using state estimation is given in Section 7.1 and, finally, Section 7.2 discusses future work.

2 Basic Aspects of Formulation

Any world in which our agents operate is one capable of assuming various *states*. There is an assumed language in which such states can be represented. A state in *BlocksWorld* might, for example, be represented by $table, towers(size(1), 2),\ towers(size(2), 3)$ signifying that the state comprises a table (on which towers may stand) together with two towers each of size (height) 1 and three towers each of size 2.

The formulation does not actually require such detailed descriptors of the world – it is sufficient for different states to be distinguished by simple atomic identifiers. 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*[1] it may have of the world, a set $\mathcal{A}$ of possible *actions* it may take and a *policy* relating perceptions to actions. We here restrict the language of states, perceptions and actions to be propositional. In any state $o \in \mathcal{O}$, the agent's possible perceptions form some subset $P(o) \subseteq \mathcal{P}$. A perception is an observation made by the agent of some aspect of the world's 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. 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 $f : \mathcal{P} \to \mathcal{A}$ 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 by $\mathcal{S}$ the set of all possible situations, 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 state. If a perception p fully described a 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 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. To say that an agent wanders (in our terms) 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 table, so bringing various items into its range of vision.

Definition 1. *A* TR-application *is a tuple $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{R} \rangle$, comprising representations of the assumed states ($\mathcal{O}$), perceptions ($\mathcal{P}$), actions ($\mathcal{A}$) and agents ($\mathcal{R}$).*

If $\mathcal{R}$ denotes a set of differentiated agents then the associations subsequently made between the other sets will need to consider which perceptions and actions apply to which agent. Since,

[1] We use the word 'perception' to refer to a set of percepts.

in this paper, we intend to consider only single-agent or cloned-agent cases this obligation is eliminated and $\mathcal{R}$ becomes a singleton. In such circumstances a TR-application takes the simpler form $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$.

Definition 2. *Let $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ be a TR-application. A* situation admitted by the application *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. $p \in P(o)$*).*

We can use *BlocksWorld* to illustrate these constructions.

Example 1. Suppose there is one agent, two blocks and the table, that blocks can be arranged into towers of the conventional kind and that the agent can hold at most one block. Then there are just three states and these can be represented by the lists $[1]$, $[1, 1]$ and $[2]$, each one being a configuration of towers composable from the blocks not being held. Next consider what the agent might "see" in this world. Let us restrict the possibilities to seeing either the table or one tower of recognizable height. Sufficient descriptors to distinguish these are $s0$, $s1$ and $s2$. Further suppose that the agent knows whether or not it is holding a block. Two further descriptors h and nh suffice for this. 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 table 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.

In our *BlocksWorld* example a fully-formed perception for the agent must combine what it sees with what it knows about its holding status. An example is $(s0, h)$. By contrast $(s2, h)$ is impossible in a 2-block world. An admissible situation (o, p) is $([1], (s0, h))$. Turning now to the actions, a possible perception-action pair (p, a) is $((s0, h), \mathtt{w})$, whereas $((s0, h), \mathtt{k})$ 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 $([1], (s0, h))$ 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 $2b$. In the sequel we shall often use these simpler forms in order to reduce presentational clutter.

This example yields altogether 6 situations and 9 perception-action pairs, determined by the assumed physics. The next step is to consider what the agent might be required to achieve and what its behaviour might be.

2.1 Goals and Policies

Taking a slightly richer example, we outline some of the issues entailed in considering choices of policy for a given goal.

Example 2. Here the world has a table and three blocks, and there is just one agent. A policy for this agent might be

$$\begin{array}{l} s1, nh \rightarrow \mathtt{k}, \quad s1, h \rightarrow \mathtt{l}, \quad s2, h \rightarrow \mathtt{l}, \quad s2, nh \rightarrow \mathtt{w}, \\ s3 \rightarrow \mathtt{w}, \quad s0, h \rightarrow \mathtt{w}, \quad s0, nh \rightarrow \mathtt{w} \end{array}$$

In general, the result of any agent's actions may vary according to the initial state of the world presented to it. However, in this example the result is inevitably a 3-tower (that is, a tower of size 3) whatever the initial state, 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 3-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). Note that it does not enable the agent to deconstruct a tower of size > 1 even though the agent is physically capable of doing so.

Even for a simple world and goal, a suitable policy can be very difficult to compose using intuition alone. Consider, for instance, a modest extension to this example whereby the world has four blocks and the goal remains a 3-tower. The above policy can still achieve the goal from most initial states. However, if the initial state has all four blocks arranged as a 4-tower or as two 2-towers then the goal is unreachable. It is not immediately obvious which alternative policy would best cope with those possibilities whilst remaining effective for the other initial states.

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 and form a set denoted by $A(p)$. 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 1-tower and places a held block upon it, it is a natural choice to determine that the result is a 2-tower and that *self* is now seeing that 2-tower rather than seeing anything else.

Definition 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 all the situations admitted by the given application. 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 graph G can be viewed as an elaboration of a conventional state-transition graph, dealing with situations rather than with states alone. Rather than showing what will happen, it shows what could happen.

For Example 2, 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). Here, $s0$ denotes that *self* sees the table, 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.

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

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

Fig. 1. States, perceptions and actions (Example 2)

Figure 2 shows the resulting unrestricted situation graph G. For the sake of compactness, a situation such as $(2, f)$ is shown there simply as $2f$.

The 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.

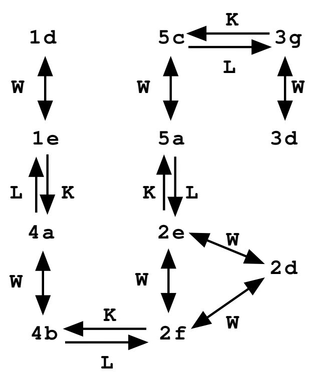

Fig. 2. Unrestricted situation graph (Example 2)

2.3 Policy-restricted Situation Graphs

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.

Example 3. The agent is the same as before but now operates in a world having four blocks, and the goal now is to build a 4-tower. The states and the associated perceptions and actions are shown in Figure 3 and the unrestricted situation graph is shown in Figure 4.

	$o \in \mathcal{O}$
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, nh	{6, 7, 8}	{`l`, `w`}
i	s0, nh	{1, 2, 3, 4, 5}	{`w`}

Fig. 3. States, perceptions and actions (Example 3)

For the policy

$$s1, nh \rightarrow \texttt{k}, \quad s1, h \rightarrow \texttt{l}, \quad s2, h \rightarrow \texttt{l}, \quad s3, h \rightarrow \texttt{l}, \quad s2, nh \rightarrow \texttt{w},$$
$$s3, nh \rightarrow \texttt{w}, \quad s4, nh \rightarrow \texttt{w}, \quad s0, h \rightarrow \texttt{w}, \quad s0, nh \rightarrow \texttt{w}$$

the f-reduced graph is shown in Figure 5.

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 *non-trough* 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.

In Figure 5 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

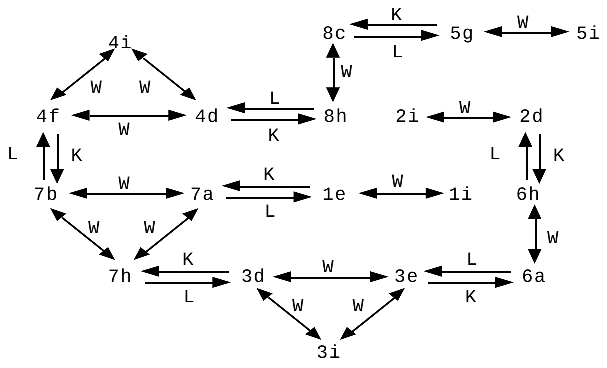

Fig. 4. Unrestricted situation graph (Example 3)

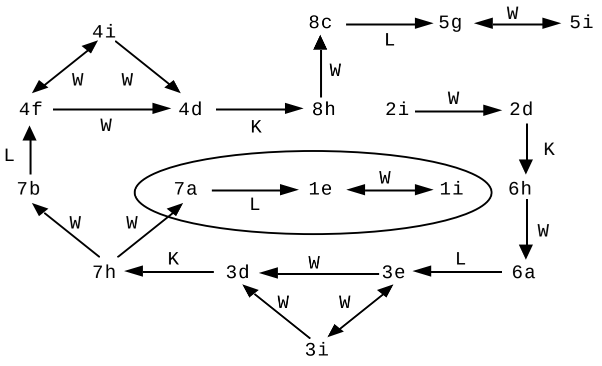

Fig. 5. An f-restricted graph (Example 3)

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

$$s1, nh \rightarrow \texttt{k}, \quad s2, nh \rightarrow \texttt{k}, \quad s2, h \rightarrow \texttt{l}, \quad s3, h \rightarrow \texttt{l}, \quad s1, h \rightarrow \texttt{w}, \quad s3, nh \rightarrow \texttt{w},$$
$$s4, nh \rightarrow \texttt{w}, \quad s0, h \rightarrow \texttt{w}, \quad s0, nh \rightarrow \texttt{w}$$

In this case the trough is $\{(2, d), (2, i), (3, e), (6, a), (6, h)\}$. 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 state the world 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 [3] 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 9 in Section 7.1 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 [6], defined as follows.

Definition 4. *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 $V(s, f)$, *effectively measuring the benefit of the agent proceeding from s, is given by the 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 that takes s to u, 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 4 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 *discounted reward* formula as defined next.

Definition 5. *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*

$$V(s, f, 0) = 0$$
$$V(s, f, k) = \Sigma_{u \in SS}(P_{su} \times (\Upsilon_{su} + \gamma(f(p)) \times V(u, f, k-1))), k \geq 1$$

A node with no successors is covered by the case when when $k = 0$ and consequently, for any node s at maximum distance from a goal node of m_s steps, $V(s, f, k) = V(s, f, m_s)$, for all $k \geq m_s$. Unless one has good reason to distinguish the rewards for actions from different situations, the fine tuning provided for in the above definitions can be dispensed with. It then suffices to use a fixed discount factor $0 < \gamma < 1$ and some fixed values r, R and T such that

$rwd(s, f) = R$ if the action $f(p)$ taken at s leads immediately to a goal
$rwd(s, f) = T$ if the action $f(p)$ taken at s leads immediately to a node in the trough (for f)
$rwd(s, f) = r$ 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, these solutions are used to compute the overall value of f, denoted by $V_{\text{pre}}(f)$, given by the (weighted) average of $V(s, f)$ taken over all nodes s in the f-reduced graph.

In this paper we assume all nodes are equi-probable as initial nodes or as nodes resulting from unpredictable, but rare, exogenous behaviour, so the average is the usual mean.

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 to the agent 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 $\gamma = 0.9$.

The finite horizon formula given in Definition 5 can be computed iteratively using the appropriate transition and reward matrices.The entries P_{su} in the *probability matrix* P are given by the probability of traversing the arc between situations s and u, and the entries Υ_{su} in the *reward matrix* are given by the reward for traversing that arc. For a given policy f, its value is given by

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

where SS is the successor set of s and 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 $V(f, 0) = [0, \ldots, 0]^T$, 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

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

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

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

which is the same as the formula for (Vs, f) given in Definition 4.

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.

3.1 Policy Predictor

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 *Blocks World* (and, potentially, other scenarios) a *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 a *policy predictor* program to compute policy values according to Definition 5, and also to compute the upper bounds (λ) 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 strongly dominates r in magnitude. 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.

With potentially thousands of policies to consider, the policy predictor needs to be as efficient as possible. A key optimization here is to reuse reward values for individual nodes. As each policy (after the first) is evaluated, the program compares this policy with the preceding one to identify those nodes whose rewards remain invariant under this policy change. Under any policy f each node has an associated subgraph (in the corresponding G_f) containing the nodes reachable from it. If this subgraph is topologically invariant upon switching to a new policy then the latter inherits the earlier rewards for that node and for all others in the subgraph. The policy predictor performs appropriate graph traversals to test this property efficiently. In practice this tactic reduces enormously the time spent on computing node rewards.

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 [7].

All these programs are currently written in LPA-Prolog[2], enabling the reshaping of examples to be undertaken through transparent and easily-adapted representations. Despite the relative tardiness of an interpretive formalism such as Prolog, it takes only a minute or so to evaluate 10,000 policies on an Apple G4 platform.

Returning to Example 3, 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. Identifying the best policy requires solving 256 sets of linear equations in 19 unknowns. A test run was made for the case r=-1, R=100, γ=0.9 which identified two optimal policies

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

each having an overall value of 241.22. The policy corresponding to the f-reduced graph G_f in Figure 5 for this example, namely

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

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

is predicted to be the third-best one, having overall value 215.19. The graph G_f has a trough, containing 3 situations, which can be entered from 8 exterior ones. By contrast, for each of the two optimal policies G_f has a trough, containing 5 situations, which can be entered from just 1 exterior one. The policy values therefore correctly reflect the propensity, on average, of the agent failing to reach the goal by entering a trough.

3.2 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 will 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, is 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 Blocks World 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 implicitly traverses part of a 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 graph G_f.

Principles of Formulation 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.
(Completeness of actions in G_f) Every simulated action is modelled by some action $a \in \mathcal{A}$.
(Soundness of situations in G_f) Every admissible 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 admissible 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.

(Exact model G_f) Every transition in $\mathcal{G}_f^W$ is between situations in $\mathcal{G}_f^W$ modelled by an arc between *distinct* situations in G_f.

(Abstract model G_f) For any arc between situation s and itself in G_f (also called a *reflexive arc*) there is at least one transition in $\mathcal{G}_f^W$ between different situations in $\mathcal{G}_f^W$.

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

Two consequences of these principles follow immediately.

1. A situation graph of an exact model cannot possess a reflexive arc, which means that the `w` action will always be modelled by an arc in G_f that changes the perception.
2. A situation graph of an exact model may possess a reflexive arc. For instance, an agent may wander between two different 2-towers in *BlocksWorld* which would be modelled by a reflexive wander about the appropriate situation s.

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 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

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

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 $V_{\text{obs}}(f)$.

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 thereby fail to reach a goal that may have been reachable in principle. 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* [32] $\tau_{\mathcal{F}}$ 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 $> 99.8\%$ 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 3, 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 Figure 6, 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.

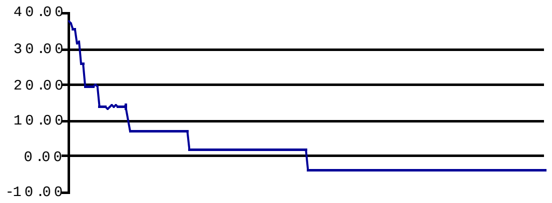

Fig. 6. Observed policy values in Example 3 (exact mode)

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 γ=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.

3.3 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 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 8 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 (or any alternative) 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 3 on a 6 x 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 7 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.

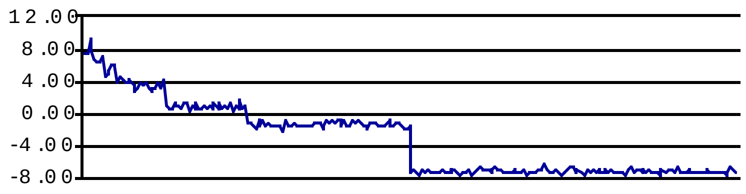

Fig. 7. Observed policy values in Example 3 (abstract mode)

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 Figure 7 would have been reduced had we applied better estimated probabilities. Rather than writing specific software to make such estimates for a grid-based *Blocks World*, 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 situation graph 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 for the best 18 policies are shown in Figure 8, whose chart is significantly smoother than the corresponding region of the chart in Figure 7.

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

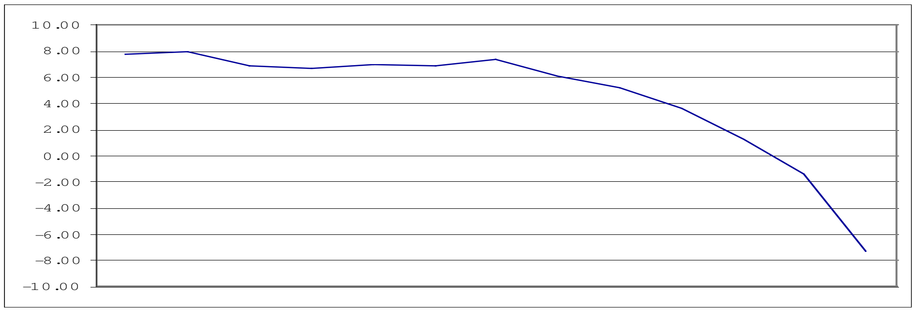

Fig. 8. Improved observed policy values in Example 3 (abstract mode)

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 2 agents there are potentially more than 13,000 policies to consider.

This section considers how our framework can be adapted to plan for a community of agents, each operating according to its own policy. 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. Of course, whatever the capabilities, all agents operate in the same environment. There is no *a priori* assumption that the agents are controlled from the outside, although there is no reason why one of the agents may not have a policy that causes it to act as a controller. We first restrict the agents to behave in an independent way and in a later section we illustrate the technique for clones that may communicate with each other.

The description of unrestricted situation graphs G has so far assumed just one agent. The changes required to G to accommodate several agents are of two kinds, but in any event in our framework any G is always constructed from the point of view of one agent. It is this feature that allows us to achieve a measure of scalability in the multi-agent case. It requires the use of a second special kind of action, called `wait` and denoted by `x`. This is in contrast to representing each agent's perception explicitly in a situation.

4.1 Exogenous Actions

The `wait` action enables the agent to wait until its perception is updated by some other agent acting. The agent that is waiting then reacts to its new perception according to its policy. The existence of a `wait` action only makes sense, therefore, if the world may be acted upon by agents other than the agent *self* under consideration. Such actions, called *exogenous*, are not necessarily benign and are not necessarily predictable. In *Blocks World*, for instance, we may imagine that while *self* is following its own agenda, some other agent is randomly knocking the top blocks off towers and leaving them scattered around singly upon the table. This may or not be helpful to *self*, depending upon the circumstances.

Exogenous activity is represented in the unrestricted situation graph G by appropriate arcs labelled `x`. 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 $s2$. *Self*'s action set now contains `x`. If *self*'s policy prescribes action `x` when in situation $s1$ the effect is to make *self* wait for some other agent to change *self*'s perception. When *self*'s policy prescribes some action other than `x` when in situation $s1$ *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. In this paper, if $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{R} \rangle$ is a TR-application, then all the agents in $\mathcal{R}$ will be clones and have identical policies unless otherwise stated. The set $\mathcal{R}$ 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 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 *Blocks World*, 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 $([1,2,3],(s0,nh),(s3,h),(s3,nh))$, indicating 7 blocks and 3 agents, two of which are seeing the 3-tower whilst the third is seeing the table, 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* and denoted by $\mathcal{G}^s$ [7]. The corresponding policy-restricted graph will be denoted by $\mathcal{G}^s_f$. The graph $\mathcal{G}^s$ can be viewed as a projection onto *self* of an implicit, much larger graph, called the *group graph*. The group graph represents situations as described above, namely by a tuple including the state and one perception for each agent. In this graph, there is a transition between situations (o,p) and (o',p') if, by the action of one of the agents in (o,p), the outcome (o',p') is possible. Given a group graph $\mathcal{G}^g$, the graph $\mathcal{G}^s$ in which we are interested is obtained by projecting situations in $\mathcal{G}^g$ to situations in $\mathcal{G}^s$ and transitions in $\mathcal{G}^g$ to transitions in $\mathcal{G}^s$ as given in Definition 6.

Definition 6. *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

$$\mathcal{T}_s = \{((o, p_k), (o', p'_k))\} \textit{ for each transition } (o, p_1, \ldots, p_k, \ldots, p_n) \textit{ to } (o', p'_1, \ldots, p'_k, \ldots, p'_n) \textit{ in } \mathcal{G}^g$$

where 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 state is o and that *self* has 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 6 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` $\neq$ `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 can occur only if 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.

If the policy chooses `x` for some perception, then it might happen that in some situation $s = (o, p)$ there are no `x`-arcs enabled by the policy other than a reflexive arc. We call such a node *isolated*. In this case an agent in situation s would be forever waiting. All other agents would necessarily be in state o also, 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. Avoidance of the possibility of deadlock might be a desirable feature. If so, then the policy predictor can filter the polices it considers by means of the *clone consistency principle*, which ignores policies that give rise to isolated situations. However, the clone consistency principle does not completely eliminate deadlock, since it could still be the case that all agents happened to be observing a perception (o, p) that required them to wait, even though there was an enabling `x`-arc from (o, p). 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 isolated 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 enabled `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 perform `k`.

Definition 7. *Let f be a policy containing the rule $p \rightarrow$ `x`. Then the* Clone Consistency Principle *requires that for each situation (o,p) there is a non-reflexive* `x`*-arc to a state o', $\mathcal{G}_f^s$ must have an arc from (o,q) to (o',q'), f must contain $q \rightarrow$ `a`, `a` $\neq$ `x` and the situations (o,p) and (o,q) must belong to a valid group.*

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 3 there were 256 possible policies, whereas if there are two agents there are more than 13,000 policies. The clone consistency principle 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 is just one of several possible heuristic filters that could be used to reduce the number of policies considered. It is likely that more than one policy filter would be useful in practice; for example policies that admit deadlock may not necessarily be worse than some policies with no possibility of deadlock, but which, for example, only reach the goal occasionally.

Example 6. For the examples in the remainder of this section and in the next section we use a different world scenario, consisting of planks and 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 in the same situation as before the attempt.

The states, perceptions and actions we use to model such an environment containing two planks and two agents are shown in Figure 9. Each state is represented by a tuple of the form $[r,t,f]$, where r is the number of (raised) planks held at both ends, t is the number of (tilted) planks held only at one end, and f is the number of (flat) planks held at neither end. Each agent may perceive whether it is seeing an unheld end (su), a held end (sh) or no end ($s0$), and whether it is holding an end or not.[3] 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.

	$o \in \mathcal{O}$
	[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`}

Fig. 9. States, perceptions and actions (Example 6)

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

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 Figure 10. Notice particularly the `x`-arc from situation $(7, f)$, which corresponds to the passive update of *self* when another clone initiates a `di` action.

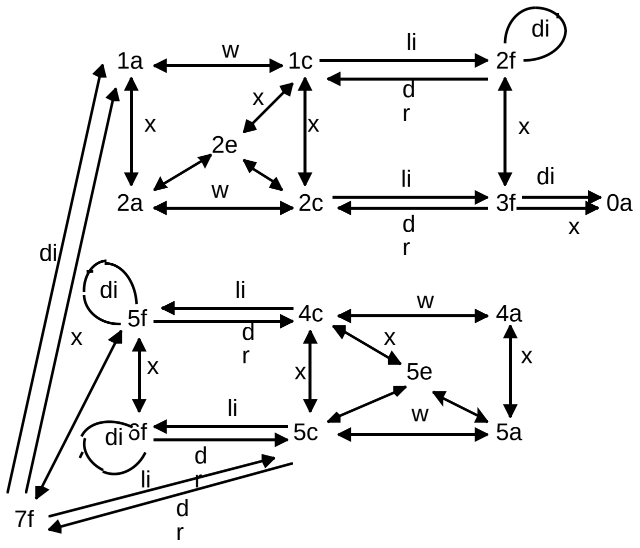

Fig. 10. The unrestricted situation graph (Example 6)

Consider a clone $r1$ governed by the policy

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

In the situation $(5, e)$ there is one flat plank and one tilted plank and $r1$ is not holding but is seeing the held end. For this perception the policy requires that $r1$ wait for another clone, say $r2$, to be holding the held end and effect the transition from $(5, e)$ to $(4, c)$, that is, to drop that end. From $r2$'s point of view, this is a transition from $(2, f)$ to $(4, c)$, requiring a `drop` action. So $r2$'s policy requires $f \to \texttt{dr}$. Since clones share the same policy, it follows that $r1$'s policy must also contain $f \to \texttt{dr}$ – which it does. This precisely illustrates what we mean by the policy being consistent for the clones. 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 sub-optimal 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 $(2, a)$.

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 there 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 $p \to$ 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 $\Sigma\mathtt{x}$ and $\Sigma\mathtt{a}$ 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 accordingly.

In the case that a = x the emergent arcs from s are all x-arcs and their probabilities can be either estimated or defaulted as equi-probable.

In the multi-agent context every node has at least one x-arc, namely a reflexive one signifying the possibility that an action (such as wander) by another agent might leave *self*'s perception unchanged. Thus our notion of a passive "update" does not insist upon a definite change of *self*'s situation. For simplicity of presentation in this paper we omit reflexive x-arcs from our pictures showing multi-agent situation graphs.

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 6 that the chosen policy is

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

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 due to the second agent wandering (from $(5, e)$), or proactively 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, all obtained by counting.

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 *multi-situations*, which generalise situations and are exactly those situations used in the group graph. 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. Examples of a multi-situation for the two-agent version of Example 6 are $((r1, (4, c)), (r2, (4, f)))$ and $((r1, (5, c)), (r2, (5, c)))$. (This extended format is used to make more explicit the association between agents, states and perceptions.) On the other hand, $((r1, (5, e)), (r2, (5, e)))$ is not a multi-situation as it is physically impossible.

As in the single agent case the simulator executes runs. A single run begins with the agents assigned to some chosen multi-situation s in $\mathcal{G}^g$. 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 $((r1,(4,c)),(r2,(4,c)))$ in Example 6 might be: $r1$ is selected, its policy (say) dictates action `li`, and the new multi-situation would be either $((r1,(5,f)),(r2,(5,c)))$ or $((r1,(5,f)),(r2,(5,e)))$, depending on whether the agent $r2$ 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 state of the world 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 recognising 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 co-operate 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) co-operation 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 another agent r_i to have the atomic perception kp (representing "knows" p), whose meaning is that one or more other agents are perceiving, and communicating to r_i, that p is true. We assume that the content of p is instantly transmissible from those other agents to r_i 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 2-plank 2-agent problem of Example 6. 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 that 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 7. This example extends Example 6 in two ways: (i) by allowing communicated perceptions, and (ii) by allowing either two or three agents. The particular states and perceptions for this formulation for two agents are given in Figure 11. The *self*-restricted graph G_c for two agents and the policy

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

	$o \in \mathcal{O}$
	[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`}

Fig. 11. States, perceptions and actions (Example 7)

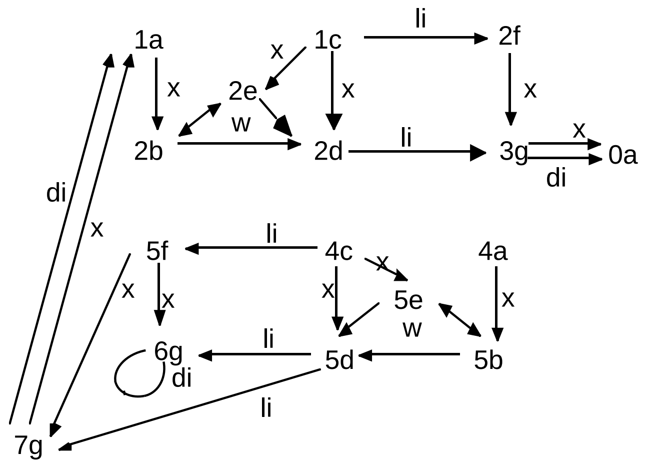

Fig. 12. The policy graph G_c for 2 agents (Example 7)

is shown in Figure 12. 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 9 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 Figure 12 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 for 3 agents. The charts for the 2-agent and 3-agent case are shown in

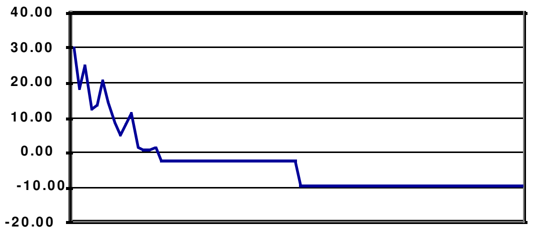

Fig. 13. Policy chart for 2 agents (Example 7)

Figs. 13 and 14, respectively.

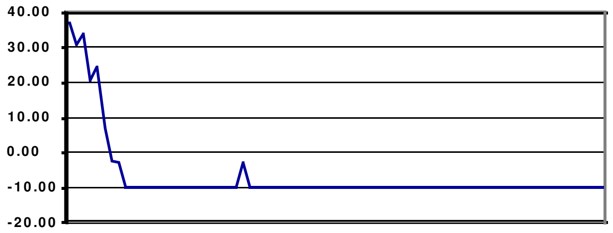

Fig. 14. Policy chart for 3 agents (Example 7)

For non-communicating agents, the closest policy to that shown in Figure 12 is

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

which is shown in Figure 15. We assume there are just two agents. The policy shown in Figure 15 offers few nodes the opportunity to reach the goal and has a poor observed policy value 8.70. By contrast, the use of communication as reflected in Figure 12 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 communication has allowed *self* to distinguish between situations $1a$ and $2b$ 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 $a : (s0, nh, knh)$ and $b : (s0, nh, kh)$ as

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

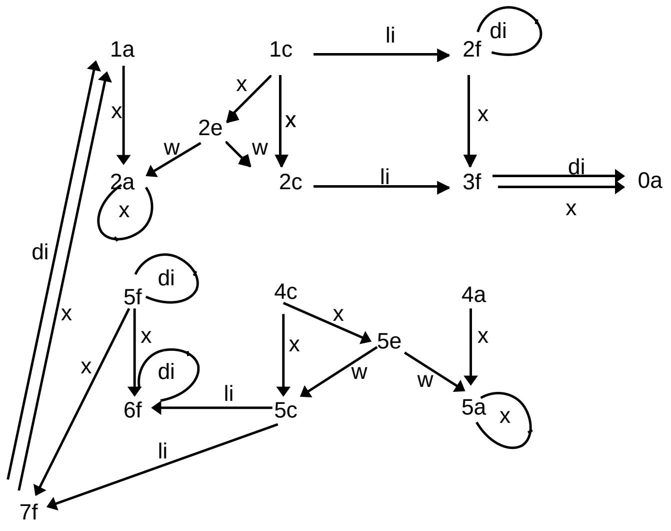

Fig. 15. Policy graph for 2 (non-communicating) agents (Example 6)

where $h1$ and $nh1$ 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.

6 Abstraction and Generic Situations

A key concern for a framework of this kind is that it shall adequately scale up to the treatment of realistic scenarios. For a scenario having perceptions $p_1, \ldots, p_n$ the number of distinct policies is the product $|A(p_1)| \times \ldots \times |A(p_n)|$. We have already emphasized that the framework is intended for simple agents in which each action set $A(p_i)$ is small. Given this, the number of policies depends chiefly upon the number n of distinct perceptions and, correspondingly, upon the diversity of the world as modeled for any particular problem.

The key to scalability therefore lies, in our view, in the use of appropriate representation. For instance, in the case of *BlocksWorld*, the tower-building examples fall into a general class in which the basic percepts (other than holding and not holding) need only be seeing a tower of the desired height, or one of lesser height or one of greater height. It is therefore not necessary to represent the seeing and acting-upon of all specific heights (1, 2, 3, *etc.*), unless the agent's physical capabilities happen to be predicated upon specific heights.

We now introduce a method of *abstractions* utilising *generic situations*, which can be described informally as situations which generalise more than one *concrete situation.* Since the purpose of introducing abstraction is to reduce the number of policies, it is the perceptions that should be the primary candidate for generalisation. However, sometimes this imposes a necessary generalisation upon the states as well. We have, in fact, already introduced several different kinds of *abstraction*, including abstract and world model, the self graph and communicated perceptions. For example, in an abstract model of some world model, the concrete situations are those used in the world model involving the various descriptors used by the simulator and the generic situations are situations of the abstract model – *i.e.* in the unrestricted situation graph. Each generic situation is the set of all concrete situations included in it. An example of an abstract percept is given by communicated percepts introduced in Section 5. These correspond to communications to *self* from other agents and a percept such as "some agent is holding" abstracts more concrete percepts such as "Agent$_i$ is holding", which would be used in the world model. (By way of comparison, note that in an exact model the generic situations are the concrete situations.) The concrete situations abstracted in a self graph are the multi-situations of the appropriate group graph, whereas the generic situations are the situations in the self graph and represent the set of possible multi-situations given *self's* situation. Another use of abstract perceptions is to model "don't care" percepts. For example, in *Blocks World* it may be irrelevant whether an agent is holding a block or not if it is not perceiving any tower. In that case the generic perception "not seeing a tower" would include two concrete perceptions, namely "not seeing a tower and holding" and "not seeing a tower and not holding". Such "don't care" perceptions are used implicitly, for example, in [15, 27].

The term "abstraction" is often used to indicate that certain aspects of the "Physics" of a "real-world" environment are ignored by the modelling process. However, this kind of abstraction (e.g. to ignore the shapes and colours of the blocks in *Blocks World*) is not relevant to the task of policy evaluation.

6.1 Generic Situations

Quite generally, generic situations can be given in terms of *generic states* and *generic perceptions.* Let $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{R} \rangle$ be a (possibly multi-agent) TR-application. The sets $\mathcal{O}$ and $\mathcal{P}$ are, respectively, called *concrete* states and perceptions and may be partitioned into classes, each class being called either a generic state or a generic perception as appropriate. The sets of generic states, perceptions and situations are, respectively, denoted O_a, P_a and S_a. Each situation (O, P) in S_a, where $O \in O_a$ and $P \in P_a$, is given by the set of concrete situations:

$$(O, P) = \{(o, p) | o \in O, p \in P \text{ and } (o, p) \text{ is a valid concrete situation}\}$$

It may be useful to restrict the set of actions made available to the policy for generic perception P to be a subset of the intersection of the sets of actions available to each concrete perception $p \in P$. This makes it possible to associate a single action with P that is possible whatever concrete perception actually pertains. Otherwise, actions could routinely fail, requiring more complex estimation of the transition probabilities. In practice, we have found this is not a very restrictive criterion.

The transitions in an unrestricted graph G_a formed from generic situations are derived from transitions in the (virtual) unrestricted graph G for the concrete situations. If there is an arc in G between situations $s = (o, p)$ and $s' = (o', p')$, then there must be an arc in G_a between generic situations $S = (O, P)$ and $S' = (O', P')$, where $s \in S$ and $s' \in S'$.

We can illustrate the notion of generic situation for the three species of abstraction mentioned above, namely the self graph, abstract and world model and communicated perceptions. First, for a self graph and two agents r_1 and r_2 (see Section 4), their concrete states $\mathcal{O}$ and perceptions $\mathcal{P}$ are the usual states and perceptions for the application (for instance states 1-7 of Example 7). Each generic state is a single concrete state. Each concrete perception is given by a pair of perceptions that form possible combinations of perceptions that the two agents could have, which for Example 7 are the tuples

$$(a, a), (a, c), (c, a), (c, c), (b, f), (f, b), (d, f), (f, d), (e, f), (f, e), (g, g)$$

Choosing *self* as r_1, the generic perceptions are formed by partitioning such a set into classes such that two tuples belong to the same class if, and only if, they both have the same perception for *self* and are both valid for some state. For example, the above tuples give rise to 7 classes, including $\{(a, a), (a, c)\}$, $\{(c, a), (c, c)\}$, $\{(e, f)\}$, which we labelled respectively by a, c and e. The generic situations are the situations of the self graph.

Second, the use of more informative descriptors in the world model used by the simulator than are represented in the abstract model by the unrestricted situation graph generally made use of generic states, but concrete and generic perceptions were identical. For example, in *Blocks World* with one agent, 5 blocks and a 4 x 4 grid, concrete states might be represented by a tuple giving the grid coordinates of the agent and the various towers, as well as the direction the agent is facing (one of the eight compass points). Thus $[(3,1,NW),(1,2,2),(3,1,4)]$ could represent the concrete state of the agent at grid point $(3,1)$ and facing North West while holding a block, and there being a 1-tower and a 3-tower at respective grid points $(2,2)$ and $(1,4)$. The generic states partition the concrete states according to the distribution of tower heights, whereas the generic and concrete perceptions are identical. A generic situation such as $([1,3],(s1,h))$, meaning the agent is seeing the 1-tower and holding a block, would include the concrete situation given above, among many others.

Third, Example 7 introduced communicated percepts, for example for a 3 agent society and *self* again as r_1, kh means that at least one of the agents other than *self* is communicating to *self* it is holding a plank end. This represents the set of six vectors $\{[\_,h,nh],[\_,nh,h],[\_,h,h]\}$ (where _ indicates *self* may be holding or not). In a similar way the perception knh represents the two vectors $\{[\_,nh,nh]\}$. Since this is necessarily a multi-agent society, the abstraction described above using the self graph can then be imposed to form generic situations.

Example 8. This illustrates more fully the World Model abstraction, since this is the conventional notion of abstraction and is taken from *Blocks World*. The goal is to build a 2-tower and a 4-tower from a sufficiently large number of blocks and to perceive the surface. There is one agent only. If there were just 10 blocks, then a comprehensive formulation of this problem dealing with its 72 possible states would not be manageable. Nor might it be realistic to suppose the agent possessed 11 distinct perceptors $s0,\ldots,s10$; even if it did, the unrestricted situation graph G would contain a huge number of situations and would offer 2^{20} policies to consider. Our approach to identifying the best policy is to assume that we need not distinguish many slightly different situations from one another, but instead collect them into generic situations. The generic states are labelled $[e2,\ e4]$, $[e2,\ ne4]$, $[ne2,\ e4]$ and $[ne2,\ ne4]$, where $e2$ denotes "a 2-tower exists" and $ne2$ denotes "no 2-tower exists" (and analogously for $e4$ and $ne4$). (In diagrams these are shortened to $[e2,\ e4]$, $[e2]$, $[e4]$ and $[]$, respectively.) For example, the state $[e2,\ e4]$ consists of all concrete states that have at least one 2-tower and one 4-tower. The generic perceptions are pairs consisting of h or nh paired with $s0$, $s2$, $s4$ or sx, the latter denoting "seeing an x-tower", where x denotes "neither 0, 2 nor 4". In order that the action `pick` be uniformly available to concrete percepts in the sx percept, the percept $s0$ is not grouped into the sx percept. This formulation gives just 24 generic situations and 128 policies.

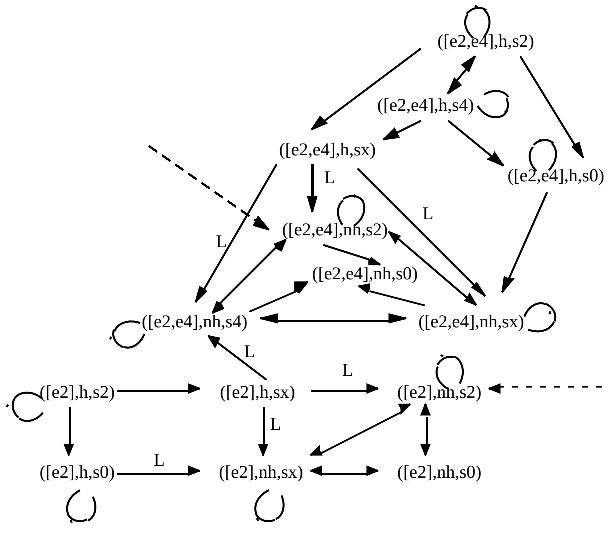

Fig. 16. Part of the graph G_f for Example 8 (10 blocks)

Figure 16 shows part of the restricted graph (for 10 blocks) for the policy whereby the agent, if seeing a tower of height neither 0, 2 nor 4 (sx), or a tower of height 0 ($s0$) can `place` if holding (h), but in all other cases wanders. The intended goal is the situation $([e2, e4], nh, s0)$. It is to be expected that an unrestricted graph G_a formed using generic states will exhibit more non-determinism than the corresponding graph G. For instance, in the restricted graph shown in Figure 16, after the action `place` from the state $([e2], (h, sx))$ the agent may either be looking at a 2-tower ($x = 1$), or a 4-tower ($x = 3$) or some other tower ($x \neq 1, x \neq 3$), whereas `place` is deterministic at the concrete level.

For this example the simulations were made using 10 blocks and a 6x6 cellular grid making 1008 runs for each policy with bound $B = 200$. Each run began with a randomly-chosen initial configuration of towers each assigned randomly to some cell and with the agent also assigned randomly to some cell. The wander action was implemented as a move to any vacant cell in the immediate neighbourhood of the agent's cell. After the move, if the new neighbourhood contained any towers then the agent would next see one of those towers, chosen randomly, but would otherwise next see the surface. The predictor took account of this in its determination of the transition probabilities on the policy-restricted graphs. Figure 17 shows the ranking chart which, though having many fluctuations, is reasonably monotonic overall and has a surprisingly high overall quality measure $Q_{\mathcal{F}} = 89.69\%$.

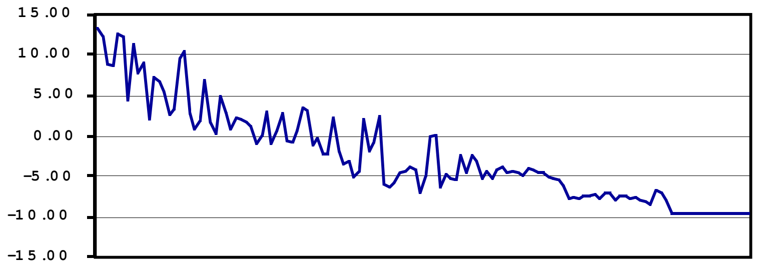

Fig. 17. Observed policy values in Example 8

The observed optimal policy is that which places a block only upon the surface or upon an x-tower and never picks a block and is the one illustrated in Figure 16 and is also the one predicted as best. It may seem counter-intuitive that this policy should never need to pick a block, but this is just a consequence of the relatively high probability that any configuration in which the agent finds itself will either already satisfy the goal condition or else require only wander and/or place actions to bring it about.

The fluctuations in the chart partly arise from the prediction method being insufficiently sensitive to the underlying transitions at the concrete level leading to the phenomenon we call *piecewise incoherence*, which we illustrate and discuss in Section 6.2. Informally, it means that a restricted situation graph G_a for generic situations can include paths that are not actually possible. This can cause both under- and over-estimation of policy values, since there may be additional paths leading to the goal, or additional paths leading away from the goal.

The volatility is also partly due to the choice of the generic states and perceptions in the abstraction. For example, a different choice than the one used in Example 8 might split the generic perception sx into two, one where $x < 4$ and the other when $x > 4$. We have always chosen an abstraction that distinguishes the goal. That is, the generic goal situation contains only concrete goal situations.

6.2 Predictive Quality for Abstractions

The predictive quality of the framework is affected by various factors, which are almost all to do with the fact that the situation graphs we use in the predictor are only abstractions of the world model. In multi-agent communities, for instance, the self graph $\mathcal{G}^s$ does not reflect all deadlock or trough situations and may include paths that are not possible in $\mathcal{G}^g$, which can lead to overestimation of a policy's value if the extra paths lead to the goal, or to underestimation if the extra paths lead away from the goal. These latter effects are also present if generic situations are used, though for slightly different reasons. We say that a graph that includes impossible paths lacks *piecewise coherence* (see [7] for an analysis of these effects). For graphs with non-determinism a further effect is due to the difficulty of making good estimates of the probability distribution on such arcs, especially of `x`-arcs. Despite these problems, we believe that our framework still allows good predictive quality in general. We now consider these effects in more detail.

Piecewise incoherence We illustrate piecewise incoherence using Examples 7 and 8. If there is a path through situations $s_0, s_1, \ldots, s_m$, the normal assumption is that an agent starting at s_0 and taking the action labelling the transition to s_1 would then be in a position to take the action labelling the transition to s_2 and so on, through to s_m. However, this may not be the case for generic situations: although the transitions at each step are due to a particular transition between concrete situations the destination of the transition starting at s_i (say) may not be the source of the next transition in the path.

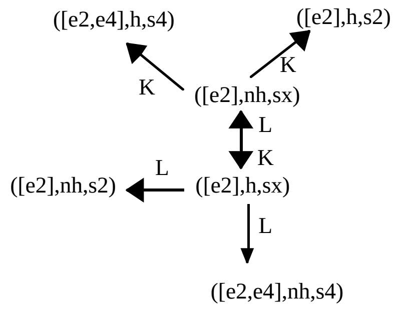

Fig. 18. Part of G_f for Policy f (Example 8)

In Example 8, for instance, consider the policy f which makes the agent `pick` from an x-tower ($x \notin \{0, 2, 4\}$), if not holding, or, if it is holding, makes it `place` on an x-tower ($x \notin \{0, 2, 4\}$) and in all other situations makes it `wander`. A fragment of the graph G_f for this policy is shown in Figure 18 and shows a path through nodes $([e2], h, sx)$, $([e2], nh, sx)$ and $([e2, e4], h, s4)$, using the actions `place` and then `pick`. However, if an agent starts with perception sx and places a block on an x-tower, in order that its perception shall remain sx, it must have started looking at a tower of height ≥ 5 and afterwards be looking at a tower of height ≥ 6. In that case, the subsequent `pick` action cannot result in perception $s4$. Each of the two transitions involved is valid, but the destination of the first `place` action is not the source required for the second `pick` action, which is seeing a 5-tower. This is what we mean by a piecewise incoherent path. For this particular policy, there are quite a number of apparent paths going towards the goal that are, in fact, not possible, so making the predicted value for this policy too high.

Since piecewise incoherence arises from two steps that do not "connect" with each other, we hypothesise that using a two-step lookahead in evaluating situation values will lead to an improvement in the predicted policy values. A transition between the generic situations $S1$ and $S2$ guarantees only that for *some* concrete situation in $S1$ there is some concrete successor situation in $S2$. However, there may be some concrete situations in $S1$ for which there are no concrete successor situations in $S2$. In case the transition to $S1$ was to one of these situations, then the value of $S1$, which normally depends on the value(s) of its successors. should not be dependent on the value of $S2$. We claim that the value of a situation j, (for some policy f) given that j has been reached from i, denoted $V(j|i)$ be given by the formula

$$V(j|i) = \Sigma_{k \in SS(j)} p_{ijk}(r_{jk} + \gamma V(k|j))$$

where p_{ijk} is the probability of the transition from j to k given that j was reached from i. The value of a situation i is then given by the formula

$$V(i) = \Sigma_{j \in SS(i)} p_{ij}(r_{ij} + \gamma V(j|i))$$

where p_{ij} is the probability of the transition from i to j. In a piecewise incoherent case some of the values p_{ijk} would be 0. For the same case of a 10-block world as in Example 8, the policy chart using the new formulas is shown in Figure 19. It is a slightly more monotonic chart. Its quality measure is 91.41%, compared to the earlier one-step-prediction figure of 89.69%. The observed best policy continues to be predicted as best. In this example the amount of piecewise incoherence is somewhat low. In other studies we have examined, where this property features more prominently, greater improvements in the predictive quality have been observed. For example, for the goal of building a 10-tower we used an abstraction that classified the seeing perceptions into "seeing 0", "seeing < 5", "seeing 5" or "seeing > 5". The overall Kendall value was similar for both the one-step and the two-step predictions, but whereas the two-step predictor ranked correctly the best six policies, the one-step predictor accorded them no discernible significance among the others.

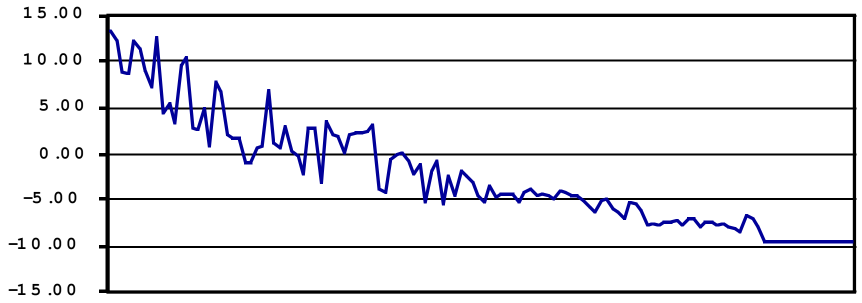

Fig. 19. Result of using conditional values in Example 8

The particular abstraction used to form the generic situations may also be a factor in obtaining reliable prediction and this still needs to be further investigated. So far, in constructing the abstractions (by hand), we used two guiding principles: (i) make the goal a specific generic situation, and (ii) perceptions that appear similar to an agent in small environments may be combined into a generic perception.

In the case of the self graph the incoherence effect is due to the presence of `x`-arcs [7]. 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. For instance, in Example 7 and for 2 agents, consider the policy

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

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 the agent $r1$ and the other agent is $r2$. In situation $(2, e)$ $r1$ is looking at a held end that it is not itself holding. It would enter $(1, c)$ if $r2$ could drop the end whilst being in $(2, f)$. $r2$ would then enter situation $(1, c)$ as well, both agents seeing the same end of the plank. $r1$ would then lift the end and move to situation $(2, f)$, while $r2$ would be passively updated to $(2, e)$. It is clear there is no way for $r1$ to get to $(3, g)$, for this would require $r2$ to lift the other end of the plank to the one $r1$ is holding. This would require $r2$ to be in situation $(2, d)$ rather than $(2, e)$. We call this phenomenon *group incoherence*.

There does not seem to be any easy way to avoid the effects on policy values due to either kind of 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 states, as was illustrated for Examples 6 and 7 in Section 5. An analysis of the relation between a group graph and a corresponding self graph is given in [7]. Piecewise incoherence can be avoided by restricting generic situations to satisfy a *uniformity property*: if there is a transition between $s1 \in S1$ and $s2 \in S2$, then there is a transition from each $s \in S1$ to some $s' \in S2$. This property was used in [28]. However, the approach using conditional situation values appears to be very promising, though it has yet to be tested for the problem of group incoherence.

7 Discussion

In this section we apply our framework to an example from [9], which uses belief states, and compare our result with the one obtained there. We also consider some extensions.

7.1 The "Star-finding" Problem

A different approach to finding policies for agents with partial observability is given in [9, 18] and uses the method of *belief states*, in which the beliefs of an agent are represented by real-valued n-dimensional vectors, $n \geq 1$. For instance, for a problem which can be modelled using four states a 4-dimensional vector would be used. The belief state $[0.25, 0, 0.5, 0.25]$ indicates that the agent believes it is more likely to be in state 3 than in states 1 or 4 and definitely is not in state 2. Each node is now a belief state and the arcs are labelled by actions. In order to obtain an optimal policy for traversing this infinite graph, an iterative algorithm is used to find an approximation by computing $V(b) = max_{a \in \mathcal{A}}(r + \gamma \times e)$, where r is the expected reward for action a from state b and e is the estimated expected value of successor states, using the current estimates. In [31] it was shown that the optimal value function can be represented using some finite set of vectors $\mathcal{V}$, such that the value of a node is computed as $V(b) = max_{v \in \mathcal{V}} b.v$. In [9, 18] it is shown how a policy can be found using this result.

The above approach and others similar to it are useful in problems where the environment changes only under the actions of the agent. Otherwise, there is no reason why the agent should assume its previous belief state is any guide to the recent past and hence to the current state. When several agents are acting, an agent's environment undergoes exogenous changes due to other agents' actions so this assumption is not valid. In effect, the use of belief states is an encoding of the recent past, or the history of the agent's earlier actions and observations. The next example, taken from [9], illustrates how our framework encodes history using additional percepts.

Example 9 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). We assume the agent stops when it sees the star and stays put if attempting to move left(right) from cell 1(4). The predicted optimal policy is

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

which corresponds to the policy in [9] and is shown in Figure 20. (Note that since every situation uses e, except those for state 3, which use s, in Figure 20 the perceptions are simplified by omitting the e or s.) Notice that, since the previous actions are not relevant when seeing a star, the perception $[s, -, -]$ is, in fact, an abstract perception. Some situations shown in Figure 20 are

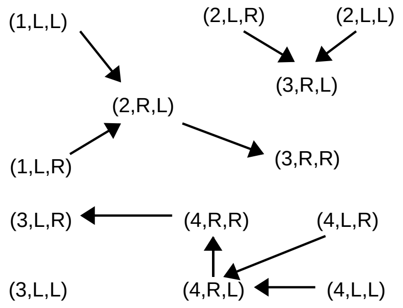

Fig. 20. G_f for Optimal Policy of Example 9

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 reach 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. This example illustrates that, for small-scale single-agent contexts, at least, our method gives the same results as the POMDP method but uses much simpler machinery to obtain them. For multi-agent contexts our use of the *self*-oriented x-arc framework appears simpler and more scaleable than the corresponding "decentralized" POMDP approach of [25], which has to analyse conjointly how all the agents are acting upon the environment.

Policies which do not exploit the agent's recent past are called *memoryless policies* in [22, 30]. An examination of the limitations of such policies is made in [30] and in [23] where the perceptions of an agent are treated as the states of an MDP. This corresponds in our framework to abstracting all states into a single state. It then follows that, since each perception occurs in exactly one state, once the key task of attributing transition probabilities is solved the "optimal policy" for the resulting graph can be obtained using standard discounted reward methods. In Section 7.3 we describe a suitable program for doing this.

The task of assigning well-estimated probabilities to transitions is important in order that predictions made using this abstraction shall not exhibit an excessive degree of piecewise incoherence. For many cases, for example when the perception of the goal situation is not unique to the goal, it is inevitable that a restricted graph will show many non-existent paths, resulting in poor prediction of policy values. In the examples in [23] this is not a problem, since the probabilities are fixed *a priori*. For instance, the example of Parr and Russell's 4×3 maze, when put into our framework has just 72 policies, which we can then easily evaluate to give the same " optimal" policy as shown in [23].

7.2 Extensions to the Framework and Further Investigations

Different Types of Agents In this paper we restrict 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 make the following two explorations: Firstly, deploy the framework for groups of non-homogeneous agents operating with differing capabilities, including those with different perceptions. We would like to establish principles for evaluating the efficacy of co-implemented non-cloned agents possessing distinct, individually-designed policies. Although this appears a hard task it would offer major benefits, since it appears from our own studies that inter-agent cooperation is most useful when the agents have distinct but complementary capabilities. Secondly, compare the behaviour of a single agent operating under a relational policy with that of a group of several agents, such that each adopts a functional policy from those contributing to the relational policy. For example, a single agent with two stochastic actions might be considered to be a combination of four different functional agents, in some proportions corresponding to the distribution of actions of the relational agent; 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 a neural net and back-propagation). An investigation of such adaptive agents is an exciting area for future work.

Memory and other internal perceptions Various examples in this paper have illustrated the potential for modelling memory and communication in our framework. We have already mentioned that 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 *Blocks World* 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 or force. If the memory reflected a true perception of the world it is likely it would be reinforced by the agent either coming across 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. In fact, 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.

If a global memory were maintained amongst a group of agents this would give quite a powerful facility. Such a global memory could be exploited and accessed through perceptions of the form $m = v$ where m names a memory store and v names its value. Such global memory is different from the communicated perceptions we have so far modelled, in that it would be convenient to be able to model that an agent "did not know something" about another agent (as opposed to knowing some negative fact). We envisage employing the concept of negation-as-failure from Logic Programming [11] in order to deal with such cases in future work using our framework.

The use of memory will introduce into our framework the possibility of implementing hierarchical policies. However, a second way to do this is by enhancing the language to definite-clause logic, enabling also the formulation of recursive policies. This extension would make our agents rather more like those of Nilsson, allowing a clearer comparison to be made.

Our focus on simple agents containing only a memoryless 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.

7.3 Branch and Bound Policy Evaluation

Although we have sought to achieve scalability through the use of abstraction, it is not always necessary to evaluate every policy in order to find the good ones. In this section we describe a branch and bound method, which in many cases avoids searching through every policy. The method is adapted to our framework from an algorithm of Littman in [22]. In order to describe it we define some new terms.

Definition 8. *Let $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ be a TR-application. A* full-policy *is one which has a rule for every perception. A* part-policy *is one which has rules for a subset of the perceptions (including no rules).*

The set of policies can be structured using a tree, called a *policy tree*, denoted by $\mathcal{T}$, in which each node is labelled by a perception p and each arc by an action in $A(p)$. If the size of $\mathcal{P}$ is 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. The value of a full-policy f is the sum of the values $V(f, s)$ for each situation s as computed by the formula given in Definition 4.) The sum $\Sigma_{s \in S} V(f, s)$, where S is the set of situations, is approximated by iteration using the recursive Equation 1

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

where, as usual, $SS(s)$ is the successor state of s, p_{su} is the probability of the transition of, and r_{su} is the reward for taking, the arc from s to u, 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.

$$V_{i+1}(f, s) = \Sigma_{u \in SS_1(s)} p_{su}(r_{su} + V_i(f, u)) + \max_a \Sigma_{u \in SS_2(a,s)} (p_{su}(r_{su} + V_i(f, u)) \tag{2}$$

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 given by Equation 2 will be an upper bound for the value of any full-policy that extends part-policy f. Similarly, a lower bound for the value can be found by replacing the *max* operator in Equation 2 by the *min* operator. 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 and lower 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. 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. A global optimum policy value B is maintained. B is initialised to a value of some known good policy. Thereafter, the branch and bound algorithm outlined in Figure 21 is used. Note that in the algorithm a node is associated with a part-policy or a full-policy.

```
procedure BB(c:node, B:int):(G:int, g:node)
//returns G the best policy value and g the best policy
   {if c is a leaf-node //c represents a full-policy
     then {if value(c)>B   then return (value(c),c)}  //c could be the optimum policy
     elseif upperbound(c)>B then
          {for each cc:=next-child-of-c
              {if lowerbound(c)>B then return BB(cc,lowerbound(c)) else return BB(cc,B)}
           }
     else return (B,b)}     // do not search below cc if cc cannot be better than B
    }
```

Fig. 21. Branch and Bound Algorithm for searching a policy tree

Algorithm BB can be made more efficient by some simple steps:

1. The upper and lower bounds are computed as needed.
2. The initial part-policy associated with c need not be empty. If a perception p has exactly one action a then the part-policy c includes the rule $p \rightarrow a$. If, through some means or other, actions for any other perceptions can be fixed, the appropriate rules are added to the initial c.

3. The remaining perceptions are divided into two kinds: *constrained* and *free*. A free perception p is one which is associated uniquely with a state, and therefore uniquely with a situation. All other perceptions are constrained. The perceptions labelling nodes in a policy tree $\mathcal{T}$ are ordered such that constrained perceptions come before free perceptions and the constrained perceptions are ordered according to the number of possible actions. The value of a part-policy c which has rules for all constrained perceptions is equal to the upperbound(c), since there is no constraint on which action to select for the free perceptions, allowing the action which maximises the node value to be selected and assuming the optimum actions are taken thereafter for all free perceptions.
4. When computing the upperbound for a part-policy c if the interim value is bigger than the current optimum B the iterations can be terminated – since the iterated values increase monotonically the upperbound cannot be less than B.

If the returned policy is not a full-policy, the remaining perceptions are free perceptions and the optimal policy can be obtained by running the algorithm again and treating the free perceptions as constrained perceptions. As an illustration the annotated tree for Example 2, including the efficiency improvements, is shown in Figure 22.

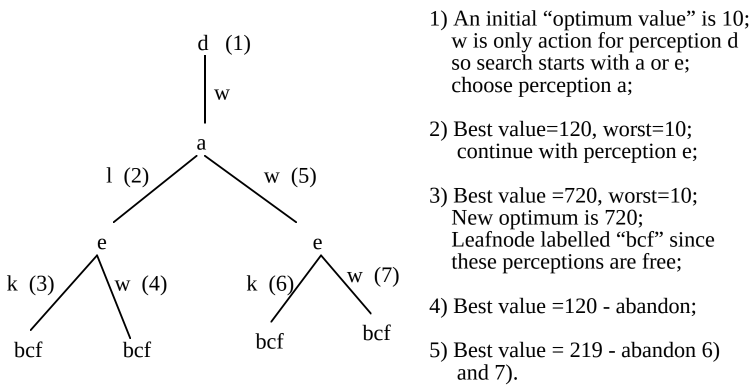
d (1)
w
a
l (2)
w (5)
e
e
k (3)
w (4)
k (6)
w (7)
bcf
bcf
bcf
bcf
1) An initial "optimum value" is 10; w is only action for perception d so search starts with a or e; choose perception a;
2) Best value=120, worst=10; continue with perception e;
3) Best value =720, worst=10; New optimum is 720; Leafnode labelled "bcf" since these perceptions are free;
4) Best value =120 - abandon;
5) Best value = 219 - abandon 6) and 7).

Fig. 22. Policy Tree in Algorithm BB for Example 2

8 Summary

We conclude by re-stating the main contributions of our framework to the design of TR-agents: First, our method of accommodating exogenous behaviour, including multi-agent interaction, solely through the special x-action and its associated clone-consistency principle. Second, our method of supporting inter-agent communication solely through suitable enrichment of perceptions. Both 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. Third, an analysis of the factors involved in design through abstraction and the demonstration that the effects of these factors can be satisfactorily accommodated in the two-step evaluation of the discounted reward procedure. In all of these we have sought to maintain maximum simplicity in both concept and formulation.

References

1. S. Benson, Inductive Learning of Reactive Action Models, *Proceedings of the 12th International Conference on Machine Learning*, pp. 47 -54, 1995.

2. S. Benson and N. Nilsson, Reacting, planning and learning in an autonomous agent, *Machine Intelligence 14*, Eds. Furukawa, K., Michie, D. and Muggleton, S, Clarendon Press, Oxford, 1995.
3. K. Broda, C.J. Hogger and S. Watson, Constructing Teleo-reactive Robot Programs, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, Berlin, pp. 653-657, 2000.
4. K. Broda and C.J. Hogger, Policies for Cloned Teleo-Reactive Agents, 2nd Conference on Multi-Agent System Technologies, Ehrfurt, LNAI, 3187, Springer Verlag, pp. 328 - 340, 2004.
5. K. Broda and C.J. Hogger, Designing and Simulating Individual Teleo-Reactive Agents, *Poster Proceedings, 27th German Conf. on Artificial Intelligence, Ulm*, pp. 1-15, 2004.
6. K. Broda and C.J. Hogger, Determining and verifying good policies for cloned teleo-reactive agents, *Computer Systems, Science and Engineering, CSSE*, 20 (4), pp. 249-258, 2005.
7. K. Broda and C.J. Hogger, Abstract Policy Evaluation for Reactive Agents, *SARA-05, 3rd Symposioum of Abstract-related Approaches*, Springer, LNAI 3607, pp. 44-59, 2005.
8. R. A. Brooks, Intelligence without Reason, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, pp. 569-595, 1991.
9. Cassandra, A.R., Kaelbling, L.P. and Littman, M. Acting Optimally in Partially Observable Stochastic Domains, *Proceedings 12th National Conference on AI (AAAI-94)*, Seattle, pp. 183-188, 1994.
10. I. Chades, B. Scherrer and F. Charpillet, Planning Cooperative Homogeneous Multiagent Systems using Markov Decision Processes, *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS 2003)*, pp. 426-429, 2003.
11. K. L. Clark, Negation as failure, in *Logic and Data Bases*, Ed H. Gallaire and J. Minker, Plenum Press, pp. 293-322, 1978.
12. Keith L. Clark and Francis G. McCabe, Go! - A Multi-Paradigm Programming Language for Implementing Multi-Threaded Agents, *Annals of Mathematics and Artificial Intelligence*, 41(2-4) pp. 171-206, 2004.
13. K. Kersting, M. Van Otterlo and L. de Raedt, Bellman goes Relational, *Proceedings 21st International Conference on Machine Learning*, Banff, Canada, 2004.
14. L. Dickens, *Learning through Exploration*, MSc Dissertation, Department of Computing, Imperial College, 2004.
15. J. Ferber, Multi-Agent Systems, Addison-Wesley, 1999.
16. C. Guestrin, M. Lagoudakis and R. Parr, Coordinated Reinforcement Learning, *Proceedings 19th International Conference on Machine Learning, ICML-02*, Sydney, Australia, pp. 227-234, 2002.
17. L.P. Kaelbling, M.L. Littman and A.R. Cassandra, Planning and acting in partially observable stochastic domains, *Artificial Intelligence*, 101, pp. 99-134, 1998.
18. L. P. Kaelbling, M. L. Littman and A. R. Cassandra, Planning and Acting in Partially Observable Stochastic Domains, *Artificial Intelligence*101, pp. 99-134, 1998.
19. M. Kochenderfer, Evolving Hierarchical and Recursive Teleo-reactive Programs through Genetic Programming, *EuroGP 2003*, LNCS 2610, pp. 83-92, 2003.
20. R.A. Kowalski, F. Sadri and F.Toni, An Agent Architecture that Combines Backward and Forward Reasoning, CADE-15 Workshop on Strategies in Automated Deduction, pp. 49-56, 1998.
21. R.A. Kowalski and F. Sadri, From logic programming to multi-agent systems, *Annals of Mathematics and Artificial Intelligence*, 25, pp. 391-419, 1999.
22. M. L. Littman, Memoryless policies: theoretical limitations and practical results, *Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour* , MIT Press, pp. 297-305, 1994.
23. J. Loch and S. Singh, Using Eligibility Traces to find the Best Memoryless Policy in Partially Observable Markov Decision Processes, em Proceedings of the 15th International Conference on Machine Learning, pp. 323-331, 1998.
24. T. Mitchell, *Machine Learning*, McGraw Hill, 1997.
25. R. Nair, M. Tambe, M. Yokoo, D. Pynadath and M. Marsella, Taming Decentralised POMDPs: Towards Efficient Policy Computation for Multiagent Settings, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pp. 705-711, 2003.
26. N.J. Nilsson, Teleo-Reactive Programs for Agent Control, *Journal of Artificial Intelligence Research*, 1, pp. 139-158, 1994.
27. N.J. Nilsson, Teleo-Reactive Programs and the Triple-Tower Architecture, *Electronic Transactions on Artificial Intelligence*, 5, pp. 99-110, 2001.
28. N. J. Nilsson, Learning Strategies for Mid-Level Robot Control: Some Preliminary Considerations and Results, *http://ai.stanford.edu/users/nilsson/trweb/tr.html*, 2000.
29. M. R. K. Ryan and M. D. Pendrith An Architecture for Modularity and Re-Use in Reinforcement Learning, *Proceedings of the 15th International Conference on Machine Learning*, Madison, Wisconsin, 1998.

30. S. Singh, T. Jaakkola and M. I. Jordan, Learning Without State-Estimation in Partially Observable Markovian Decision Processes, *Proceedings of the 11th International Conference on Machine Learning*, pp. 284-293, 1994.
31. E. J. Sondik, The Optimal Control of partially observable Markov processes over the infinite horizon: Discounted costs, *Operations Research*, 26, pp. 282-304, 1978.
32. G. W. Snedecor and W. G. Cochran, *Statistical Methods*, Iowa State Univ. Press, 1972.