Policies for Cloned Teleo-Reactive Robots

Krysia Broda and Christopher John Hogger

Department of Computing, Imperial College London
kb@doc.ic.ac.uk, cjh@doc.ic.ac.uk

Abstract. This paper presents a new method for predicting the values of policies for cloned multiple teleo-reactive robots operating in the context of exogenous events. A teleo-reactive robot behaves autonomously under the control of a policy and is pre-disposed by that policy to achieve some goal. Our approach plans for a set of conjoint robots by focusing upon one representative of them. Simulation results reported here indicate that our method affords a good degree of predictive power and scalability.

1 Introduction

This paper examines the problem of designing optimal or near-optimal policies for a group of *teleo-reactive* (TR) robots operating in the context of *exogenous* events. From the viewpoint of any individual robot an exogenous event is any change in the world not caused through its own actions. The characteristics of a TR robot are, typically, that it behaves autonomously under the control of a stored program or policy, 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. The main features of TR robots were introduced in [11] and further developed in [12]. We make two key assumptions about a robot: that it has (i) little or no access to cognitive resources, such as beliefs or reasoning systems, and (ii) only partial observational capability, in that its perceptions may not fully capture the whole environmental state. Such robots could find uses in those applications, for instance nano-medicine or remote exploration, where physical or economic constraints might preclude such access. Informally, a good policy is one which disposes a robot to perform well in pursuit of a defined goal whatever state it is currently in.

A reactive robot responding only to its current perception of the state can be modelled naturally by a Markov Decision Process (MDP). When that perception captures less than the entirety of the state it is often modelled using the framework of Partially Observable MDPs (POMDPs) [7, 5, 10]. Seeking good policies for a given robot within the POMDP framework typically requires, in order to compensate for the lack of full state information, that the actual state be estimated using a history of previous events. When the history is owed *only to actions of known robots*, it is accurately representable. However, it is not accurately representable if arbitrary exogenous events can occur, which is exactly

the case in which we, by contrast, are interested. Our approach is somewhat similar in motivation to that of [6], in that we plan for a set of conjoint agents by focusing upon some representative subset of them.

Our position must also be distinguished from that taken by many in the multi-agent community who equip their agents with complex theories about the environment and elaborate communication mechanisms, because of our focus on minimally equipped robots. Nevertheless, the problem remains of dealing with a robot having limited perception.

Choosing a good policy for a given goal is generally difficult. Even simple robots and worlds can offer huge numbers of policies to consider. Most ways of designing TR-(teleo-reactive) policies are learning-based, as in [1] which applies inductive logic programming to determine advantageous associations between actions and consequences. Other learning schemes for TR-planning are [9], [14] and [8].

The core concepts in our approach, first proposed in [2], are as follows. The world in which the robot operates has a total set $\mathcal{O}$ of possible *states*. The robot is presumed to possess perceptors through which it may partially perceive these states; the set of all its possible *perceptions* is denoted by $\mathcal{P}$. The robot is also presumed capable of certain *actions* which form a set $\mathcal{A}$. In any state $o \in \mathcal{O}$, the robot's possible perceptions form some subset $P(o) \subseteq \mathcal{P}$, and in response to any perception $p \in P(o)$ the robot's possible actions form some subset $A(p) \subseteq \mathcal{A}$. A *policy* (or program) for the robot is any total function $f : \mathcal{P} \rightarrow \mathcal{A}$ satisfying $\forall p \in \mathcal{P}, f(p) \in A(p)$. The number of possible policies is the product of the cardinalities of the $A(p)$ sets for all $p \in \mathcal{P}$. A *situation* for the robot 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. Associated with the robot is a unique *unrestricted situation graph* G in which each node is a situation. This graph has an arc labelled by an action a directed from node (o, p) to node (o', p') in every case that $a \in A(p)$ and execution of action a could take the world state from o to o' and the robot's perception from p to p'. The possibilities that $o = o'$ and/or $p = p'$ are not inherently excluded. The policies are evaluated using discounted-reward principles [7] applied to the situation graphs.

A key feature of this approach is that the graph for the robot can represent, besides the effects of its own actions, the effects of exogenous actions, whether enacted by other robots or by other indeterminate agencies. This obviates analysing comprehensively the explicit combinations of all the robots' behaviours. This treatment is one contribution to the control of scalability and is in contrast to that presented in [13], who explicate entirely the joint perceptions of the robots. Scalability can be further controlled by abstraction, by way of replacing sets of concrete situations by abstract generic situations [4]. Situation graphs have also been employed to represent and exploit inter-robot communication [3].

This paper demonstrates the use of situation graphs to predict optimal or near-optimal policies for groups of cloned TR-robots, and assesses their effectiveness on the basis of simulation results. The main contribution is to show

that useful designs for robot clones can be obtained by analysing a single robot acting in the context of events instigated by others. Section 2 illustrates how TR-scenarios are formulated and Sect. 3 discusses the evaluation and testing of policies. In Sect. 4 we describe the treatment of multi-robot contexts and some case-studies are presented in Sect. 5. Section 6 uses the results of those studies to discuss some ramifications of the design method, whilst Sect. 7 summarizes our conclusions.

2 An Illustration

The above ideas are now illustrated using *BlocksWorld* for a single robot case. Of course, *BlocksWorld* is just a generic exemplar of a wide range of state transition systems.

In *BlocksWorld* the world comprises a surface and a number of identical blocks. A state is an arrangement of the blocks such that some are stacked in towers on the surface whilst others are each held by some robot, and is representable by a list of the towers' heights. Suppose there are just 2 blocks and one robot which can, at any instant, see just one thing – the surface, a 1-tower or a 2-tower, denoted by $s0$, $s1$ and $s2$ respectively. Further, it can sense whether it is holding a block or not doing so, denoted by h and nh respectively. Its perception set $\mathcal{P}$ then comprises just 5 legal pairings of "seeing" status and "holding" status. At any instant the robot performs whichever action $a \in \mathcal{A}$ corresponds, according to its policy, to its current perception $p \in \mathcal{P}$; this behaviour is assumed to be durative – the action cannot change while perception p persists. The action set $\mathcal{A}$ in *BlocksWorld* is $\{\mathtt{k}, \mathtt{l}, \mathtt{w}\}$. In a `k`-action ("pick") the robot removes the top block from a tower it is seeing, and afterwards is holding that block and seeing the resulting tower (or the surface, as appropriate). In an `l`-action ("place") the robot places a block it is holding upon what it is seeing (the surface or some tower), and afterwards is not holding a block and is seeing the resulting tower. In a `w`-action ("wander") the robot merely updates its perception without altering the state. Figure 1 shows these aspects of the formulation, labelling the states $1 \ldots 3$ and the perceptions $a \ldots e$.

(a) states and perceptions

o		$P(o)$
1	[1, 1]	$\{a, b\}$
2	[1]	$\{d, e\}$
3	[2]	$\{a, c\}$

(b) perceptions and actions

p		$A(p)$
a	$[s0, nh]$	$\{\mathtt{w}\}$
b	$[s1, nh]$	$\{\mathtt{k}, \mathtt{w}\}$
c	$[s2, nh]$	$\{\mathtt{k}, \mathtt{w}\}$
d	$[s0, h]$	$\{\mathtt{l}, \mathtt{w}\}$
e	$[s1, h]$	$\{\mathtt{l}, \mathtt{w}\}$

Fig. 1. Formulation of the 2-block world

Figure 2a shows the unrestricted graph G, where each node (o, p) is abbreviated to op. There are 16 policies, one being

$$f = \{a \to \mathtt{w},\ \ b \to \mathtt{w},\ \ c \to \mathtt{w},\ \ d \to \mathtt{w},\ \ e \to \mathtt{l}\}$$

To adopt this policy is to eliminate certain arcs from G to leave the *f-restricted graph* G_f shown in Fig. 2b. G_f gives the transitions the robot could make under policy f. For a given goal, choosing f partitions the set $\mathcal{S}$ of nodes in G into

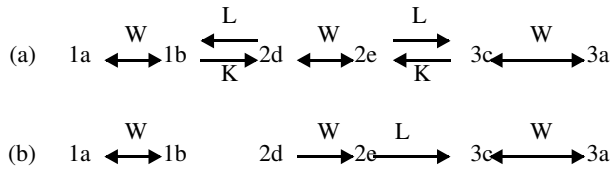

Fig. 2. (a) Unrestricted graph G; (b) f-restricted graph G_f

two disjoint subsets T_f and N_f called the *trough* and *non-trough* respectively. T_f comprises exactly those nodes from which there is no path in G_f to the goal, and N_f comprises all others. Choosing, say, the goal $3c$ in the example yields $T_f = \{1a, 1b\}$ and $N_f = \{2d, 2c, 3c, 3a\}$. Here, no arc in G_f is directed from N_f to T_f. When such an arc does exist, G_f is described as NT-bridged. In that case a robot able in principle to reach the goal from some initial node may traverse the bridge and thereafter be unable to reach the goal, since no arc directed from T_f to N_f can exist. Define $\lambda = 100|N_f|/|\mathcal{S}|$. Then for a series of experiments choosing initial situations randomly, the robot's predicted success rate $SR_{\mathrm{pre}}(f)$ (as a percentage) is $< \lambda$ if G_f is NT-bridged but is otherwise exactly λ. For the example above $SR_{\mathrm{pre}}(f) = 66.67\%$.

3 Predicting and Testing Policies

This section explains how policies are evaluated in a single-robot context. The extension to the multi-robot context then follows in the next section.

The predicted value $V_{\mathrm{pre}}(f)$ of a policy f is the mean of the values of all situations in G_f, assuming they are equally probable as initial ones. The value of a situation S should reflect the reward accumulated by the robot in proceeding from S, favouring goal-reaching paths over others. If S has immediate successor-set SS then its value $V(S)$ can be measured by the discounted-reward formula

$$V(S) = \Sigma_{s \in SS}(p_s.(rwd(s) + \gamma.V(S_i)))$$

where p_s is the probability that from S the robot proceeds next to s, $rwd(s)$ is the reward it earns by doing so and γ is a discount factor such that $0 \leq \gamma < 1$. In a single-robot context equal probabilities are assigned to those arcs emergent from

S. We employ two fixed numbers R and r such that $rwd(s) = R$ if s is a goal and $rwd(s) = r$ otherwise, where $R \gg r$. The situations' values are therefore related by a set of linear equations which, since $\gamma < 1$, have unique finite solutions and so determine a finite value for $V_{\text{pre}}(f)$. In general, choosing $R \gg r$ ranks more highly those policies well-disposed to the reaching of the goal, whilst γ controls the separation (but not, in general, the ranks) of the policies' values. A *Policy Predictor* program is used to compute policy values by this method, and also to compute (as above) the upper bounds (λ) on their success-rates.

We tested the quality of the *Policy Predictor* by using a *Policy Simulator* program. Each *run* of the simulator takes an initial situation S and subsequently drives the robot according to the given policy f. The simulated robot then implicitly traverses some path in G_f from S. The run terminates when it either reaches the goal or exceeds a prescribed bound B on the number of transitions performed. As the path is traversed, the value $V(S)$ is computed incrementally on the same basis as used by the predictor. Equal numbers of runs are executed for each initial situation S and the mean of all observed $V(S)$ values gives the observed policy value $V_{\text{obs}}(f)$. The simulator also reports the observed success rate $SR_{\text{obs}}(f)$.

The simulator supports both *positionless* and *positional* simulation modes. The former associates no positional data to the robot or the towers, and so uses the same information about the problem as the predictor. Thus, if the robot picks a block from the 2-tower in the state [1, 2], it is indeterminate as to which tower it will see afterwards in the new state [1, 1]. By contrast, the positional mode assigns discrete grid coordinates to the robot and the towers, and exploits these to effect transitions in a manner somewhat closer to physical reality through knowing precisely where the robot is located and what it is seeing. This paper gives results only for positionless simulation, using parameter values $R = 100$, $r = -1$ and $\gamma = 0.9$.

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. Overall predictive quality is reflected by the extent to which the chart exhibits a monotonically decreasing profile. A precise measure of this can be computed as the Kendall rank-correlation coefficient $\tau_{\mathcal{F}}$ for $\mathcal{F}$, as follows. Let (f, g) be any pair of distinct policies in $\mathcal{F}$ satisfying $V_{\text{pre}}(f) \leq V_{\text{pre}}(g)$. This pair is said to be concordant if $V_{\text{obs}}(f) \leq V_{\text{obs}}(g)$, but is otherwise discordant. Then $\tau_{\mathcal{F}} = 2(C - D)/n(n-1)$ where C and D are the numbers of concordant pairs and discordant pairs, respectively, in $\mathcal{F} \times \mathcal{F}$. Its least possible value is -1 and its greatest is +1. It is convenient to map it to a percentage scale by defining $Q_{\mathcal{F}} = 0\%$ when they disagree maximally.

Figure 3 shows the chart for the 2-block example above, choosing the goal as $(3, c)$, i.e. "build and see a 2-tower". Observed policy values are measured along the vertical axis, and predicted ranks (here, from 1 to 16) along the horizontal one. Arcs emergent from $(3, c)$ were suppressed from G to mirror the fact that the simulator terminates any run that reaches a goal. For each policy the simulator executed 1002 runs with $B = 100$. The chart is perfectly monotonic and $Q_{\mathcal{F}} =$

100%. The optimal policy is $\{a \rightarrow$ `w`, $b \rightarrow$ `k`, $c \rightarrow$ `w` (*or* `k`), $d \rightarrow$ `w`, $e \rightarrow$ `l`$\}$, which picks a block only from a 1-tower and places a block only upon a 1-tower. Its success rate is 100%.

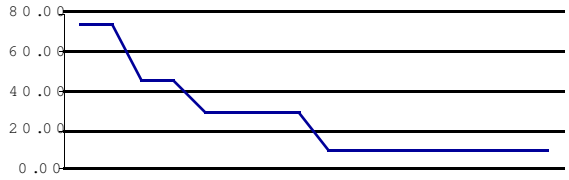

Fig. 3. Policy ranking chart for building a 2-tower

4 Policies for Multiple Robots

Whether or not it is useful to employ multiple robots in the pursuit of a goal depends upon the nature of the world and of the robots' interactions with the world and with each other. With limited perceptions, incognizance of the goal and lack of communication, simple TR-robots of the kind we have considered may cooperate advantageously only by serendipity. Predicting accurately the behaviour of multiple robots 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 robots there are potentially more than 13,000 policies to consider.

This section explains how a situation graph focusing upon one robot, termed "*self*", can express the effects of other robots acting in concert with it. The key question is whether such a graph enables prediction of good policies for the combined robots without requiring explicit analysis of all the combinations of the situations they occupy.

For this purpose we introduce a special domain-independent action named `x` that all robots possess in their $\mathcal{A}$ sets. So for *BlocksWorld* $\mathcal{A}$ is now {`k`, `l`, `w`, `x`}. In any situation graph an `x`-arc directed from (o, p) to (o', p') signifies that, from the viewpoint of *self*, the state transition from o to o' is effected *exogenously* by some other agent(s), here restricted to be other robot(s). So, when *self*'s policy prescribes an `x`-action for perception p, this is interpreted operationally as requiring *self* to "wait" (that is, to become inactive) until its situation is updated exogenously to (o', p'). On the other hand, when *self*'s policy prescribes some action other than `x` *self* may alternatively undergo an exogenous transition caused by the action of another robot. We call this *passive updating* of *self*.

Whether or not the robots all have the same policy, it is desirable to reduce the possibility of deadlocks in which they would all be waiting. One way to do this is to arrange that whenever a robot's policy requires it to wait, the

transition that it awaits should be achievable in one step by some other robot. In the case that all robots have the same policy f (i.e. are clones) this is called the *clone-consistency principle*, defined as follows. Suppose f contains $p \rightarrow \mathtt{x}$. If G_f has an $\mathtt{x}$-arc from (o, p) to (o', p') then (i) G_f must also have an arc from (o, q) to (o', q') labeled by an action $\mathtt{a}$ other than $\mathtt{x}$ and (ii) f must contain $q \rightarrow \mathtt{a}$. Besides reducing deadlock this principle also reduces very significantly the number of policies requiring to be examined. A weaker one could require that the progression from o to o' be achieved within $k > 1$ steps for some specified k, allowing a more liberal dependence upon exogenous behaviour but at the expense of having more policies to consider.

In the multiple-robot context the predictor assigns probabilities to the arcs of a graph G_f as follows. Each node $S = (o, p)$ has emergent arcs for the action $\mathtt{a}$ that *self* performs according to the rule $p \rightarrow \mathtt{a}$ in its policy f. If $\mathtt{a} = \mathtt{x}$ then these $\mathtt{x}$-arcs are the only arcs emergent from S. The predictor counts, for each one, the number of distinct ways its transition could arise, taking account of the number of other robots that could effect it. Their probabilities are then required to be proportional to these counts and to have sum $\Sigma_{\mathtt{x}} = 1$. If $\mathtt{a} \neq \mathtt{x}$ then S additionally has arcs labeled $\mathtt{a}$ and these are assigned equal probabilities having sum $\Sigma_{\mathtt{a}}$. In this case $\Sigma_{\mathtt{a}}$ and $\Sigma_{\mathtt{x}}$ are made to satisfy $\Sigma_{\mathtt{a}} + \Sigma_{\mathtt{x}} = 1$ and $\Sigma_{\mathtt{x}} = (n-1)\Sigma_{\mathtt{a}}$ where n is the total number of robots, reflecting the relative likelihood in situation S of *self* being the robot selected to act or one of the others being selected to act.

In the multiple-robot context the simulator effects transitions between *multi-situations*, which generalise situations. A multi-situation is a *physically possible* assignment of the robots to situations that share a common state and differ (if at all) only in perceptions. An example of a multi-situation for the 2 robot example shown in Fig. 4 is $\{(r1, (3, d)), (r2, (3, d))\}$. On the other hand, $\{(r1, (3, d)), (r2, (3, a))\}$ is not a multi-situation as it is physically impossible.

At any given moment in a simulation, some robots may be flagged as *waiting*, because they previously performed $\mathtt{x}$-actions for which they still await the required transitions. Robots not currently so flagged are called *active*. A run begins with an initial multi-situation chosen randomly from the set of all multi-situations distinguishable up to robot identity. Each subsequent transition from a state o is made by randomly choosing some active robot and performing the action $\mathtt{a}$ prescribed by its policy, causing the state to become some o'. If $\mathtt{a} \in \{\mathtt{w}, \mathtt{x}\}$ then $o = o'$ and the other active robots' perceptions remain unchanged. If $\mathtt{a} \in \{\mathtt{k}, \mathtt{l}\}$ then $o \neq o'$ and the other active robots are passively updated in a manner chosen randomly from all possible ways of updating them to state o', whilst any waiting robots whose required transitions have now been effected become active and acquire the perceptions in state o' that they had awaited.

A transition from the multi-situation $\{(r1, (2, d)), (r2, (2, d))\}$ (again for the example shown in Fig. 4) might be: $r1$ is selected, its policy (say) dictates action $\mathtt{k}$ and the new multi-situation would be either $\{(r1, (6, h)), (r2, (6, i))\}$ or $\{(r1, (6, h)), (r2, (6, d))\}$. The situation change for $r2$ in this transition is an ex-

ample of a passive update corresponding to an x-arc from either $(2, d)$ to $(6, i)$ or $(2, d)$ to $(6, d)$. The simulator chooses randomly from these two possibilities.

A run is terminated when any active robot in the group reaches a goal situation or when the simulation depth bound B is reached. The simulator's successive random choosing of which robot 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.

An x-arc in the graph thus represents two notions at once. On the one hand it represents the action of deliberate waiting in accordance with *self*'s own policy. On the other hand it indicates how *self* can be impacted by the actions of others.

5 Multiple Clone Examples

o		$P(o)$
1	[2,2]	$\{e, i\}$
2	[1,1,1,1]	$\{d, i\}$
3	[1,1,2]	$\{d, e, i\}$
4	[1, 3]	$\{d, f, i\}$
5	[4]	$\{g, i\}$
6	[1,1,1]	$\{a, d, h, i\}$
7	[1, 2]	$\{a, b, d, e, h, i\}$
8	[3]	$\{c, f, h, i\}$
9	[1,1]	$\{a, h\}$
10	[2]	$\{b, h\}$

p		$A(p)$
a	$[s1, h]$	{l, w, x}
b	$[s2, h]$	{l, w, x}
c	$[s3, h]$	{l, w, x}
d	$[s1, nh]$	{k, w, x}
e	$[s2, nh]$	{k, w, x}
f	$[s3, nh]$	{k, w, x}
g	$[s4, nh]$	{k, w, x}
h	$[s0, h]$	{l, w, x}
i	$[s0, nh]$	{w, x}

Fig. 4. Formulation of 4-block world with 2 robots

All these examples are for a world having 4 blocks, but with various goals and various numbers of cloned robots. The total number of policies is 13,122 but only 480 are clone-consistent. Each of these 480 was given about 1000 simulation runs with $B = 100$, and the reward parameters used were $R = 100$, $r = -1$ and $\gamma = 0.9$.

Example 5.1: [*2 robots building a 4-tower*] The formulation is shown in Fig. 4. The unrestricted graph G has 30 nodes, among which is the goal $(5, g)$. Figure 5 shows a fragment of G ($4d$ appears twice only for drawing convenience). Even this fragment has some incident and emergent arcs omitted from the figure. Figure 6 shows the policy ranking chart for the set $\mathcal{F}$ of the best 240 policies. For these,

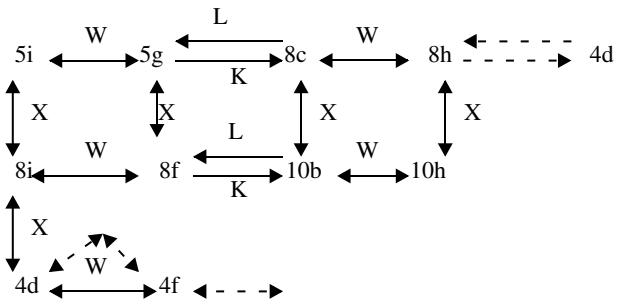

Fig. 5. Graph G for 4-blocks and 2 robots

$Q_{\mathcal{F}}$ is 87.90%. The best ones are ranked almost perfectly. The observed optimal one, having predicted rank 2, is

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

This picks only from a tower of height < 3 and places upon any tower of height < 4 (but not upon the surface).

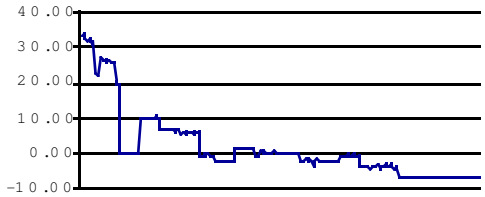

Fig. 6. Policy ranking chart for Example 5.1

Example 5.2: [*3 robots building a 4-tower*] The graph G is now a little larger, as there is an extra state. It has 36 nodes. Figure 7 charts the best 240 policies, for which $Q_{\mathcal{F}}$ is 89.94%. For the best 20 it is 85.79%. The observed optimal policy is the same as for 2 robots, and is the predicted optimal one.

Example 5.3: [*4 robots building a 4-tower*] The graph G now has yet one more state (in which every block is being held). It has 39 nodes. Figure 8 charts the best 240 policies, for which $Q_{\mathcal{F}}$ is 90.02%. For the best 20 it is again 85.79%. The observed optimal policy is the predicted optimal one, being

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

which picks only from a tower of height 1 and places only upon a tower of height < 4.

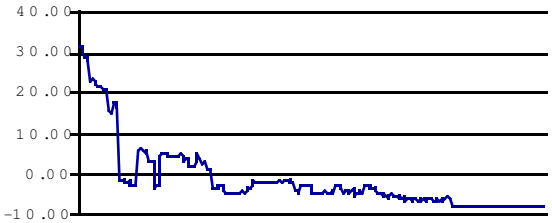

Fig. 7. Policy ranking chart for Example 5.2

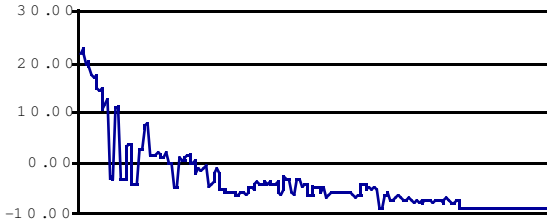

Fig. 8. Policy ranking chart for Example 5.3

Example 5.4: [*2 robots building a 2-tower and two 1-towers*] Here, G is the same graph as in Example 5.1. The goal is now $(3, i)$, i.e. "build one 2-tower, two 1-towers and see the surface". This example is different from the previous three, in that a robot cannot, using a single perception, recognise that the goal has been reached. Figure 9 charts the best 240 policies, for which $Q_{\mathcal{F}}$ is only 78.61%. For the best 20, however, it is 90.53%. The three observed best policies

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

(or $c \to \mathtt{x}$ or $c \to \mathtt{l}$) are the three predicted best ones, sharing the property that they pick only from a tower of height > 2 and can place upon the surface. They differ only in the case that a robot is holding a block and seeing a 3-tower (perception c), and their values are virtually identical. In the case $c \to \mathtt{x}$ the robot waits for another robot to pick from the 3-tower. The case $c \to \mathtt{l}$ is a retrograde choice as there is no merit in building a 4-tower, and it may seem surprising that it occurs in a high-value policy. However, the probability of a 3-tower actually arising is made very low by the other rules in these policies, so that choosing $c \to \mathtt{l}$ has no significant impact upon the policy value. The chart is more volatile in this example. This may be due in part to the looser coupling of state and perception in the goal situation. In the 4-tower examples the perception "seeing a 4-tower" implies that the state contains a 4-tower. In the present example there is no single perception that implies that the goal state has been achieved, so the robots have poorer goal-recognition. However, the volatility in all these examples has other causes as well, as discussed in the next section.

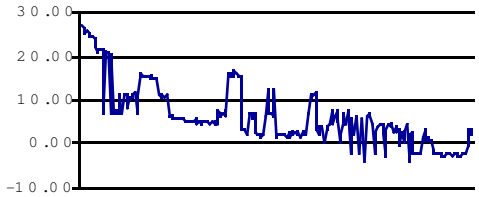

Fig. 9. Policy ranking chart for Example 5.4

Though limited in scope, these case studies suggest that, in order to select a good policy, the robot designer is unlikely to go far wrong in choosing from the best 10% according to the predictor. The predictor is clearly imperfect, and is inevitably so due to the approximations it makes, but unless the design imperatives are very demanding it is sufficient to seek policies yielding reasonable rather than optimal behaviour.

The option remains open to filter ostensibly high value policies using a more sophisticated predictor, in order to eliminate those that are actually worse than the basic one suggests. Such refinements can employ, for a small selection of policies, analyses that would be impractical in scale if applied to all policies.

6 Factors Affecting Predictive Quality

We dealt with a multi-clone context by optimizing a single clone using only the `x`-action to represent the effects of other clones. This is our alternative to analysing comprehensively how the clones behave conjointly. The latter analysis would need an unrestricted graph in which each node were a complete multi-situation having emergent arcs for all the real (non-`x`) actions its members could perform. From this *group-graph* one could then seek the best policy for any one clone. The combinatorial nature of such an analysis makes it generally impractical. Figure 10 repeats the chart for Example 5.4 but highlights two of its many

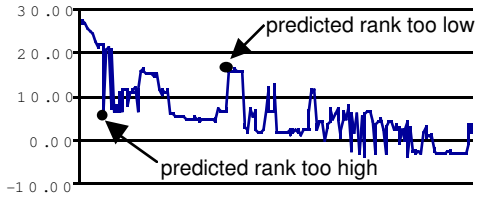

Fig. 10. Prediction anomalies in Example 5.4

anomalies. The policy indicated with too high a predicted rank is

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

If we examine how its predicted value is calculated, we find that a node $S = (4, f)$ in G_f is assigned a positive value $V(S)$ even though, in reality, the goal is unachievable from S. As noted earlier, the calculation of policy values must take into account passive transitions of *self* (x-updating). Our method sees paths from S to the goal that contain steps in which *self* is assumed to be x-updatable by the other robot, although, in fact, the goal is unachievable from S. In situation S no block is held and *self* is seeing a 3-tower and must pick. From S there is the following ostensible path in G_f that reaches the goal:

(a) *self* is in situation $(4, f)$ and – because of what happens next to it in this path – the other robot must also be in $(4, f)$;
(b) the other robot takes action k and moves to $(7, b)$, whilst *self* is passively updated to $(7, e)$;
(c) *self* performs two wander actions taking it to $(7, i)$, whilst the other robot remains at $(7, b)$;
(d) the graph shows an x-action from $(7, i)$ to the goal $(3, d)$, which is correct since another robot could be in situation $(7, h)$. However, if this particular path is followed, the other robot would not be in the required $(7, h)$ but in $(7, b)$.

This path is infeasible because to achieve it would require *self* to perform actions and undergo x-updates that the other robot could never fulfil. In the group-graph this joint scenario would not be feasible and any group situations containing $(4, f)$ would be assigned negative node values thereby reducing the policy's rank.

Figure 10 also indicates a policy with too low a predicted rank. Anomalies of this kind arise primarily from inaccuracy in the probabilities on the x-arcs in G_f, again owing to insufficient information about the situations actually occupied by *self*'s partners.

7 Conclusions

We have presented a method using situation graphs for predicting policies for TR-robots in multi-robot contexts. The use of situation graphs enables policies to be evaluated taking account of objective states, yielding greater discrimination than approaches (e.g. [7]) that focus primarily upon perceptions. This discrimination is necessary due to our assumption that exogenous events are not only possible but an inherent fact in a multi-robot context, so that we cannot rely on using histories to estimate current situations. Furthermore, since we wish to find policies to achieve goals not detectable by a single perception, we cannot use the method of back-chaining and the so-called regression property used in [11], since a given perception may not be unique to the goal.

However, inclusion of objective state information poses a greater need for scalable methods, especially in multi-robot contexts. We have demonstrated here that in such contexts good policies are obtainable by analysing how any single robot responds to exogenous effects caused by other robots. This applies whether or not the robots are cloned. It has further been shown in [4] that the approach

continues to yield good results in positionally sensitive formulations, even when abstraction is also employed.

Our future work will seek to understand better the relationship between our "self-based prediction graph" and the group graph and to investigate how the use of the former might be refined to give even better predictive power.

References

1. Benson, S. Learning Action Models for Reactive Autonomous Agents, PhD, Dept. of Computer Science, Stanford University, 1996.
2. Broda, K., Hogger, C.J. and Watson, S. Constructing Teleo-Reactive Robot Programs, *Proceedings of the 14th European Conference on Artificial Intelligence*, Berlin, pp 653-657, 2000.
3. Broda, K. and Hogger, C.J. Designing Teleo-Reactive Programs, Technical Report 2003/8, Dept. of Computing, Imperial College London, UK, 2003.
4. Broda, K. and Hogger, C.J. Designing and Simulating Individual Teleo-Reactive Agents, *Poster Proceedings, 27th German Conference on Artificial Intelligence*, Ulm, 2004.
5. 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.
6. Chades, I., Scherrer, B. and Charpillet, F. 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.
7. Kaelbling, L.P., Littman, M.L. and Cassandra, A.R. Planning and Acting in Partially Observable Stochastic Domains, *Artificial Intelligence* 101, pp 99-134, 1998.
8. Kochenderfer, M. Evolving Hierarchical and Recursive Teleo-reactive Programs through Genetic Programming, *EuroGP 2003*, LNCS 2610, pp 83-92, 2003.
9. Mitchell, T. Reinforcement Learning, Machine Learning, pp 367-390, McGraw Hill, 1997.
10. Nair R., Tambe, M., Yokoo, M., Pynadath, D. and Marsella, M. 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.
11. Nilsson, N.J. Teleo-Reactive Programs for Agent Control, *Artificial Intelligence Research* 1 pp 139-158, 1994.
12. Nilsson, N.J.Teleo-Reactive Programs and the Triple-Tower Architecture, *Electronic Transactions on Artificial Intelligence* 5 pp 99-110, 2001.
13. Rathnasabapathy, B. and Gmytrasiewicz, P., Formalizing Multi-Agent POMDPs in the Context of Network Routing, *Proceedings of 36th Hawaii International Conference on System Sciences (HICSS'03)*, 2003.
14. Ryan, M.R.K. and Pendrith, M.D. An Architecture for Modularity and Re-Use in Reinforcement Learning, *Proceedings of the 15th International Conference on Machine Learning*, Madison, Wisconsin, 1998.