Extended abstract: Learning search strategies

Daishi Harada and Stuart Russell

Background

The underlying motivation for the work presented in this paper is to usefully understand what is called the *value of computation* (Russell & Wefald 1991). By this we intuitively mean the following. Suppose we have some computational process C which, at some point, has a choice between the computations c_0 and c_1. We would like to be able to make claims of the form: c_0 is a better choice for C than c_1 because the *value* of c_0 is greater than that of c_1. Extending this idea by calling the set of choices made by C a *program*, we would similarly like to be able to say that the value of the program $\pi = \{c_i\}$ is greater than that of $\pi' = \{c'_i\}$. This, in turn, would allow us to define the best, or *bounded optimal* (Russell & Subramanian 1995) program for C. Let us consider what would be required of a formalism to make these ideas meaningful.

It is certainly necessary to first provide a context for the computation; i.e., we need to define what we want the computation to do. As an example, the classic framework of complexity has developed by considering the context of *decision problems*. Unfortunately, it is difficult to construct a definition for the value of computation within this framework which is ultimately non-trivial. The underlying cause of this difficulty is that there is a "correct answer" to a decision problem. This answer is independent of time, and is hence also independent of the computational process. At best, we can only discuss the amount of time (or some other resource) taken by the *entire* computational process which computes and outputs this answer. What we would like, instead, is to be able to consider the incremental *process* of computation and the tradeoffs between continuing and stopping the computation at any point in time. This idea is explored in the field of *anytime algorithms* (Dean & Boddy 1988; Zilberstein & Russell 1995). Unfortunately, a limitation of the framework of anytime algorithms is that the focus is still typically on some single, independent problem. It is assumed that this problem is given to the computational process when it is initialized, and the goal becomes that of characterizing the performance of the computation with respect to the time at which it is stopped/interrupted. What we would like, however, is to be able to express the problem itself as a process with which the computational process must interact; for example, we would like to be able to address the question of what to do when decision problems occur *sequentially*. Hence we find that the context of *control problems* is most suited for our discussion.

The particular model of control problems which we consider is based on[1] that of *semi-Markov decision processes* (SMDPs) (Parr 1998). It is well known that SMDPs have optimal policies which are stationary[2]. Hence if we assume that our computational process is capable of perfectly implementing this policy then the problem is equivalent to the one typically studied in the field of *reinforcement learning* (Bertsekas & Tsitsiklis 1996; Kaelbling, Littman, & Moore 1996). This case places no real constraints on the computation, and hence makes considering the value of computation uninteresting. However, this assumption holds only for relatively simple domains. In general, real-world problems are sufficiently large that a brute force attempt to represent/implement the optimal policy is not feasible given the actual computational resources available. Much of the current research in the field of reinforcement learning therefore addresses the use of approximations and heuristics (Bertsekas & Tsitsiklis 1996; Ng, Harada, & Russell submitted).

The goal of this paper is to consider yet another framework in which real-world control problems may be solved. The benefit of the framework, compared to the other approaches, is that the computational process/controller is explicitly modelled. What this means is that a solution to a problem within our framework can *by definition* be implemented, and in addition, will be the bounded optimal solution to the problem. In this paper we consider the case when we select *search* as our model of computation. In the next section we describe this framework in more detail. Then In the section following, we discuss some of the issues

[1]The imprecision here is due to the fact that the actual formalism describing our framework is not yet clear. The reason for this is discussed further below.

[2]At least in the sense of *perfect rationality* (Russell & Subramanian 1995)

which must be addressed to solve problems within the framework. Finally, in the last section we present some preliminary results supporting this approach.

The idea

We assume that the reader is familiar with the basic ideas and terminology from the fields of reinforcement learning and search. In this section, we motivate our idea using MDPs.

Let $\langle X, U, T, R \rangle$ denote a MDP where the tuple elements are $\langle$states, controls, transitions, rewards$\rangle$. Assume that we have a model of the domain, and an estimate of the value function $\hat{V} : X \to \mathbb{R}$. The standard method of obtaining a policy π from this estimate is to define for all $x \in X$:

$$\pi_{\hat{V}}(x) \quad = \quad \operatorname*{argmax}_{u \in U} E\left[R(x,u) + \gamma \hat{V}(Y)\right],$$

where Y is a random variable distributed according to $T(\cdot; x, u)$ representing the successor state and γ is the discount factor. It is clear that this may be interpreted as performing a depth 1 search/lookahead.

Now suppose we instead allow the controller to perform arbitrary search, and to base its control on the backed up information. To do this, we need to make decisions about the following: the order in which search nodes are expanded, and when to stop searching and actually "commit" to a control. The approach that we take is to view these decisions as the *meta-level* control problem. In this context, we call the original problem the *object level* control problem.

As an example, take the MDP fragment illustrated in Figure 1, and let the current state of the domain be x_0. Then a simple depth 1 search would indicate that the best control in x_0 is u. However, expanding the node x_2 would reveal that perhaps v is the better choice. For the sake of argument, let us assume that additional search would not further change this result. Then a "good" meta-level policy would control the search process as illustrated in Figure 2. Although this simply illustrates the step-by-step expansion of a search tree, the important point is that this process may be viewed as a controllable system. In general, the set of possible controls in a search state M would consist of expanding one of the (state, action) pairs in the frontier of M, or the special control `commit`. By integrating this process as the controller of the object level problem, we obtain our new, meta-level control problem.

The approach that we propose to take towards solving this control problem is to apply the standard algorithms from reinforcement learning. For this approach to work, however, there are several issues which must be addressed.

Issues

We first discuss an important issue which was finessed in the previous section, that of *time* (certainly an important part of any dynamical system). Ideally, we would like to be able to argue the viability of our approach in something like the following way. Note that SMDPs may be viewed as Markov at *any* point in time if we augment the state space with the time interval which has elapsed since the last state transition. Hence, assuming that the controller keeps track of this elapsed time, the joint dynamical system consisting of both the object level SMDP and the controller is semi-Markov *with respect to* the decision points of the controller. This justifies the application of the standard algorithms.

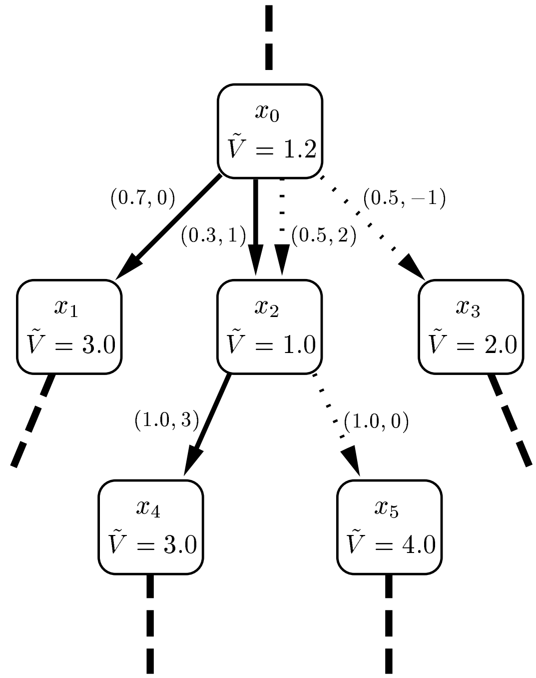

Figure 1: A problem fragment. The solid and dotted lines represent controls u and v, respectively. The labels on the lines are (transition probability, reward) pairs.

Unfortunately, this argument ignores the problem of synchronization. It *does hold* if we assume in addition that: "communication" between the object level SMDP and the controller occurs asynchronously through a passive medium; and the elapsed time is measured and available "magically" by some entity external to both the SMDP and the controller. In other words, if both the SMDP and the controller are required to "poll" the other for their inputs (i.e., the control signal and the (state, reward) pair), then the dynamics of the two systems decouple cleanly and the argument follows. This, however, is a rather weak model for interaction. In particular, it does not allow us to express the idea of *interruption*. For example, it does not allow the controller to react to important changes in the domain which may occur during its deliberation, and in addition, even were the controller lucky enough to poll the domain state at the right time to recognize its im-

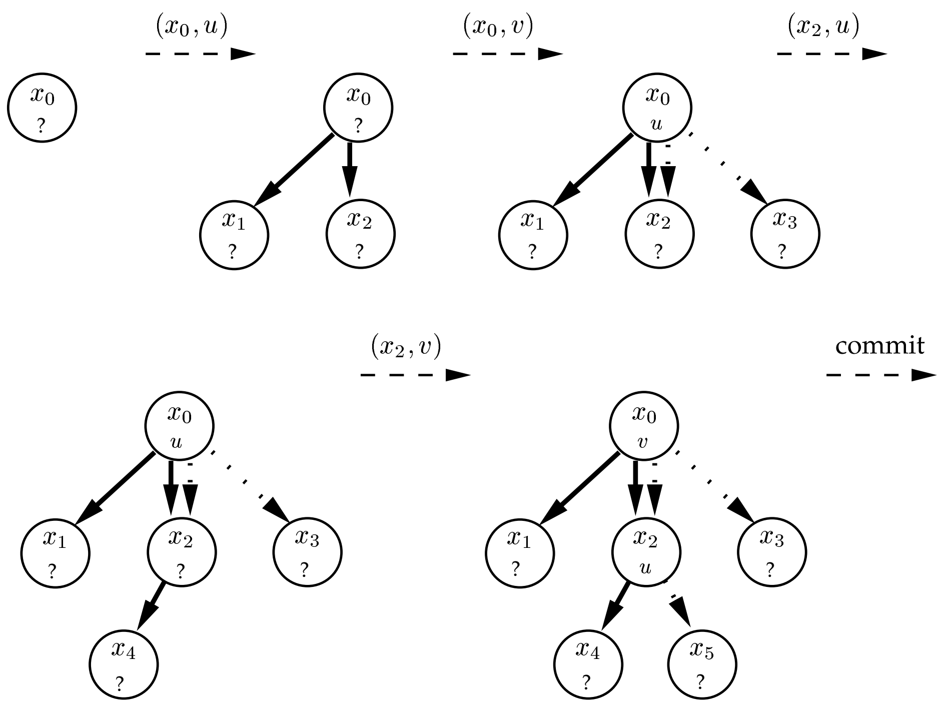

Figure 2: The controller (search) states visited for the problem fragment in Figure 1 under the meta-control sequence $[(x_0, u), (x_0, v), (x_2, u), (x_2, v)]$. Each control (x, u) means "search control u in state x". Underneath each node label is the best control for that node given the current information. The estimated and backed up values are not shown to reduce clutter. The important points are that the best control changes as we expand the tree, and that this process of expansion may be viewed as a controllable process.

portance, there is *nothing it can do* to affect the outcome of the current domain transition. We would prefer, of course, a model in which the interaction between the two processes is *active*. One approach might be to also keep track of the time taken by the controller, to allow the argument above to apply symmetrically between the domain and the controller. We are currently investigating these issues and potential formalisms.

Let us now ignore that fact that we do not have an theoretical foundation through which to describe our system. The hope is that empirically investigating the higher-level issues will shed more light onto the properties and structure we would desire from a theoretical foundation. The principal issue which we must now address before attempting to "naively" apply the standard reinforcement learning algorithms is now how to deal with the fact that we have taken an SMDP which, by assumption, is too large to solve/control directly, and created a meta-level control problem which is even larger. The basic idea is to approximate the meta-level state space with the search state plus just a few statistical features extracted from the object level model. As an example of why we might believe this to be a reasonable approximation, recollect the example illustrated in Figure 1, and note that we were able to mostly justify the "correct" policy even though the object level domain was not described (aside from the value estimate). In general, what we hope to do is to create a *general purpose* controller which is independent of the target domain except through a finite number of parameters. We are currently investigating the case when these parameters represent the *volatility* in the value estimate at a search node as we expand its decendants.

The set of search states over a domain, however, is still much larger than the domain. Hence we are also considering possible function approximators to use with search states. There is, however, a unique issue which arises when considering meta-level problems. Although in the standard model of reinforcement learning representations of a policy based on the value function and the Q-function are considered "essentially equivalent", this is not the case for the meta-level problem. This is because the standard case does not take into account the computational resources required to extract an actual policy out of either the value function or the Q-function, and in particular, the computing of values such as $R + \sum_u T_{xy}(u)V(y)$ are considered "free". However, when doing search, it defeats the purpose of solving the meta-level problem if in order to compute where to search next, it is necessary to search ahead. Hence it follows that when applying reinforcement learning techniques to a meta-level problem, the representation used *must* be a Q-function, as this allows the meta-controller to choose the next computation *without* examining what the consequences of each of the computations might be. We expect that our function approximator will be a recursive/message passing function which starts at the root of the search state and leaves the estimated Q-values at each of the nodes in the frontier.

Preliminary results

Some preliminary empirical results have been obtained using the simple but non-trivial domain of Tetris. These experiments used a drastically simplified controller architecture, and should be taken only as indication of the potential that the framework described in this paper offers.

We assume that the reader is familiar with the basic mechanics of Tetris. We work with a 6×10 (width $\times$ height) Tetris board, and suppose that the search process considers atomic the control sequence $\mathrm{shift}(x) + \mathrm{rotate}(\theta) + \mathrm{drop}$. (see Figure 3). The reward function gives 1 point for each row eliminated, and the performance measure is the total reward gathered before the game ends (when the board fills up).

Standard RL techniques can generate an approximate linear value function for board configurations (Bertsekas & Tsitsiklis 1996). The following features were used in this approximation:

1. The height h_n of column n.
2. The depth d_n of the "valley" at column n. This depth is defined to be $\min(\max(h_{n-1} - h_n, 0), \max(h_{n+1} - h_n, 0))$ for $1 < n < 6$. For d_1 and d_6 we simply take $\max(h_2 - h_0, 0)$ and $\max(h_5 - h_6, 0)$, respectively.
3. The maximum height $\max_k h_k$.
4. The number of holes, where a hole is defined to be an open space with a filled space above (not necessarily directly above.)

In addition, we approximate the real dynamics of Tetris (where the piece falls while the player/controller thinks) by assuming that the time available to the controller for computation is the number of empty rows at the top of the board times some constant ($\tau(10 - \max_k h_k)$.) If the time taken by the controller exceeds that available to it, we let the Tetris piece simply drop from its initial position/orientation.

In the Tetris domain, lookahead search clearly improves the quality of decisions. Figure 4 shows the performance of two controllers—a depth-1 search examining about 100 states and a depth-2 search examining about 10,000 states—as a function of τ. When $\tau = \infty$, i.e., when the game waits for the controller to make its decision, depth-2 search is roughly twice as good as depth-1 search. As τ is reduced, however, depth-2 search quickly becomes impossible in most situations, whereas depth-1 search remains feasible. For any given τ, one would like to learn which depth works best; less trivially, one would like to learn to recognize states in which depth-2 is both useful and feasible, since a flexible policy may dominate either of the fixed policies. This is indeed the case, and the performance of the flexible policy is shown by the line labelled "Metalevel". Note, however, that in this simple simulation experiment we have not illustrated the full

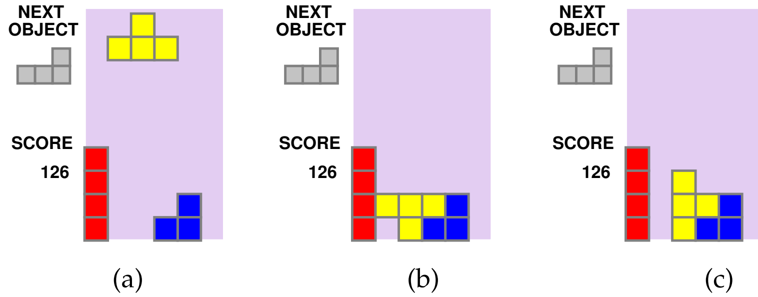

Figure 3: (a) A Tetris state: the T-shaped piece must be placed. (b) and (c) show two possible choices. (b) is better because the next object can complete and remove the second row and thereby eliminate the "hole" created.

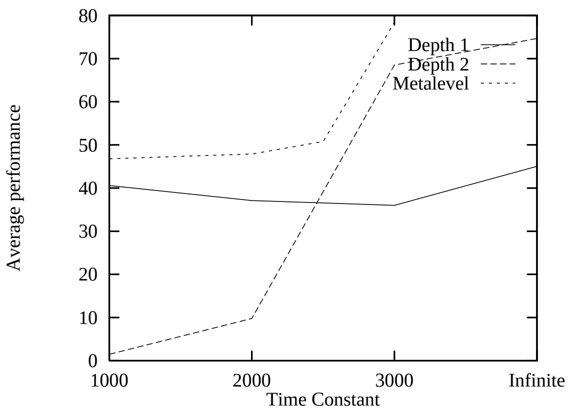

Figure 4: The two lower curves show a comparison of the performance of depth-1 and depth-2 search-based controllers, for varying time constants τ. The upper curve shows the performance of the learned metalevel policy. The lines are the average scores of 10 games played by each controller.

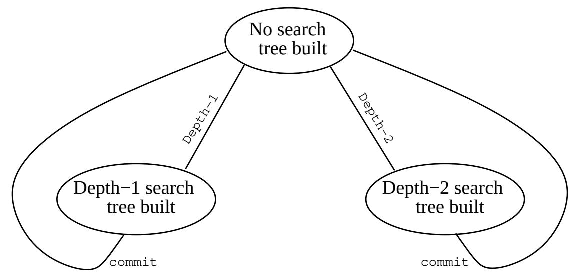

Figure 5: The depth-1 search vs. depth-2 search controller architecture.

model of "learning to search". Rather, we have simply provided the meta-level controller with two choices: search either to depth 1 or to depth 2. This simplified meta-level controller is illustrated in Figure 5.

References

Bertsekas, D., and Tsitsiklis, J. 1996. *Neuro-Dynamic Programming*. Belmont, Massachusetts: Athena Scientific.

Dean, T. L., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 49–54.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285.

Ng, A.; Harada, D.; and Russell, S. submitted. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*.

Parr, R. 1998. *Hierarchical Control and Learning for Markov Decision Processes*. Ph.D. Dissertation, University of California, Berkeley.

Russell, S. J., and Subramanian, D. 1995. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research* 2.

Russell, S. J., and Wefald, E. 1991. *Do the Right Thing*. Cambridge, Massachusetts: MIT Press.

Zilberstein, S., and Russell, S. J. 1995. Approximate reasoning using anytime algorithms. In Natarajan, S., ed., *Imprecise and Approximate Computation*. Kluwer Academic Publishers.