Cache Performance of Priority Metrics for MDP Solvers

David Wingate and **Kevin D. Seppi**
Computer Science Department
Brigham Young University
Provo, UT 84602

Abstract

As algorithms scale to solve larger and larger MDPs, it becomes impossible to store all of the model information of the MDP and the supporting data structures of the algorithm in RAM. This motivates the study of the *disk-based-cache efficiency* of solution algorithms. We contrast the cache efficiency of normal value iteration with that of the P-EVA algorithm, and introduce the concept of "intrinsic cacheability." We concentrate on prioritized solution methods, and demonstrate that the choice of priority metric greatly affects cache behavior. Experimental results indicate that the best priority metric allows problems which are four times larger than available RAM to be solved effectively.

Introduction

Value iteration is a well-known technique for solving MDPs, but can converge slowly when applied naively. The performance of value iteration (VI) can be improved by several orders of magnitude by avoiding redundant or useless backups, and by performing backups in the "correct" order. Both improvements can be accomplished simultaneously through *prioritized computation*: instead of naively sweeping through the state space and backing up every state at every iteration, a prioritized method executes each backup in priority order. This has the effect of dramatically accelerating convergence by focusing computational effort on regions of the problem that are maximally productive.

Wingate and Seppi (Wingate & Seppi 2003) demonstrated that demanding perfect prioritization of backups is prohibitively expensive due to priority queue overhead, but also noted that this overhead can be all but eliminated through *approximate prioritization*. Their algorithm approximated a perfectly prioritized backup ordering through the use of *partitions*, which are simply sets of states. Instead of constructing a priority queue over states, the queue is constructed over partitions. When a partition p is processed, their algorithm value iterates normally over the states within p until they converge. It then reprioritizes p and any other partitions containing states that depend on any state in p. The resulting algorithm is named P-EVA, which is short for "Partitioned Efficient Value Iterator."

Experimentally, the P-EVA algorithm is so effective at quickly solving large MDPs that the factor limiting the size of problem which can be solved is no longer time, but RAM. In order to solve problems that do not fit into available RAM, two options are possible: since not all data is in use at all times, some of it may be cached to disk. Or, if the data was derived from a system model (perhaps through discretization of a continuous problem like Mountain Car or Acrobot), it may be recomputed. Naturally, not all MDPs are derived from system models, so the only generally applicable method is to write unused data to disk.

These observations motivate the study of *cache efficiency* as an additional consideration when designing algorithms to solve very large MDPs. If an algorithm accesses information in a way that is not amenable to good cache behavior, it severely limits the size of problems that can be considered for solution. In addition, if an algorithm is accessing information in a way that is not good for a cache, it is probably inefficient from the standpoint of information propagation.

Since prioritized methods are among the most efficient information propagators, it is reasonable to examine them as candidates for good cache behavior. This paper therefore focuses on the P-EVA algorithm, and examines the two different priority metrics it employs. It also discusses the ways in which its cache behavior compares to that of normal value iteration. An important perspective this paper takes is that of *implicit* cache management: we examine the emergent caching properties of the solution algorithms with respect to a given priority metric, as opposed to developing explicit cache management strategies.

Prioritized, Partitioned Value Iteration

This section briefly reviews the P-EVA algorithm and the definitions of the *H1* and *H2* priority metrics. It also reviews the semantics of each priority metric, which is important when considering their cache behavior.

Normal VI is defined in terms of the value function, which is typically expressed as

$$V_t(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s'} Pr(s'|s,a) V_{t-1}(s') \right\}$$

where $R(s,a)$ is the reward for executing action a in state s, $Pr(s'|s,a)$ is the probability of transitioning to state s' given (s,a), and $\gamma \in [0,1)$ is the discount factor.

Prioritizing backups necessitates additional information,

the most important of which is the *Bellman error* of a state:

$$B_t(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s'} Pr(s'|s,a) V_t(s') \right\} - V_t(s)$$

P-EVA builds two different prioritization metrics upon the Bellman error function. The first metric, *H1*, is equal to the Bellman error itself:

$$H1_t(s) = B_t(s)$$

This is the same priority metric used by Moore and Atkeson in their work on Prioritized Sweeping (Moore & Atkeson 1993), although their work was done in a very different context. The second metric is:

$$H2_t(s) = \begin{cases} B_t(s) + V_t(s) & B_t(s) > \epsilon \\ 0 & \text{otherwise} \end{cases}$$

P-EVA defines the priority of a state s is defined as either $H1_t(s)$ or $H2_t(s)$, depending on which priority metric is being used. The priority of a partition is defined as the maximum priority of any state within the partition. We say that a state s is *dependent* on state s' if $\exists a \in A$ s.t. $Pr(s'|s,a) \neq 0$. We say that a partition p depends on another partition p' if $\exists s, s'$ s.t. $s \in p$, $s' \in p'$ and s is dependent on s'.

The P-EVA algorithm begins by constructing a priority queue over partitions using either *H1* or *H2*. Partitions are then processed in priority order by selecting the highest priority partition p. States within p are value iterated until convergence. The priorities of any partitions depending on p are then recomputed, and the priority queue is updated. The process repeats until termination, which typically occurs when an ϵ-optimality test is passed.

The *H1* and *H2* priority metrics generate very different backup orderings. Using *H1*, P-EVA can be thought of as a greedy reduction in the error of the value function estimate. P-EVA-*H1* therefore tends to propagate value function information quickly throughout the state space, but typically must process regions multiple times. P-EVA-*H2* tends to process loops repeatedly, which ensures that regions are fully converged before moving on to other regions of the problem. This minimizes the amount of reprocessing necessary, and helps explain why P-EVA-*H2* exhibits much better cache behavior than P-EVA-*H1*.

Later in the paper, we will talk about an "information frontier." By this, we are referring to the places in the value function estimate which are between regions of the problem which have not yet been processed, and regions which are fully converged. Formally, these are the states with the highest Bellman error.

Normal VI and Predictive Caches

As a baseline, it is important to consider the cache behavior of normal VI, using a disk-based cache. To compare standard VI to P-EVA, we consider a hypothetical VI variant which iterates over partitions, backing up states within each partition. This algorithm exhibits pessimal behavior for non-predictive caches, because as it iterates over a problem, it touches each partition once, and then moves on to the next partition. It never revisits a partition until it has visited every other partition in the problem. Assuming a cache strategy that kicks out the least-recently-used element, any cache smaller than the entire problem will always yield a hit ratio of zero.

It is possible to improve this performance through the use of a predictive cache, since the order in which partitions will be accessed is known a priori. However, such a predictive cache will not be explored further, either for normal VI or for P-EVA. Such a study would detract from the goal of the paper, which is to evaluate the intrinsic cache behavior of the algorithms. The many issues surrounding predictive caches represents an excellent research space that we leave for future work.

Information-Frontier-Only Statistics

The final observation we make relative to cache behavior involves the *intrinsic cacheability* of a problem. In the P-EVA algorithm, there are two different times that partitions are needed. The first happens when a partition p is extracted from the priority queue as having the highest priority. p becomes the working partition, and if it is not in cache, it must be retrieved from disk. We term these the "information-frontier-only" partition accesses, for reasons explained below. However, there is a second time that partitions are needed: when we have *finished* processing p, we must recompute the priority of any partition d that depends upon p, meaning that the transition information for certain states in d will be needed. This in turn means that if d is not in cache, it must be read from disk. We term these "auxiliary partition accesses."

Currently, the P-EVA algorithm pulls the entire contents of d out of the cache for each auxiliary access, but this is not strictly necessary. It is possible (and even probable) that not *all* states in d depend upon some state in p, so it may be possible to pull out only the necessary states. Although disk latency may still be a factor, this may be reduce the time needed by the disk read. Or, it may be feasible to recompute the transition information for just the states in d that depend on some state in p, instead of recomputing the entire partition.

The distinction between "information-frontier-only" and "auxiliary" partition accesses is significant for another reason. Although we do not pursue this idea in this paper, it may be possible to *approximate* the priorities between partitions, instead of recomputing them exactly. Such an approximation may not require all of the transition information in the d partition, which means that an auxiliary partition access may not be necessary at all.

Therefore, there are two different measures of cache performance: first, there is the cache performance of the algorithms *as they stand*, and second, there is the *intrinsic cacheability* of the problem itself. Of course, both measures must be taken with respect to a given priority metric. Counting auxiliary partition accesses dramatically changes the cache performance characteristics of our algorithms. For that reason, the results section reports two sets of cache efficiencies.

Experimental Results

Two traditional reinforcement learning problems were selected to quantify P-EVA's cache efficiency: Mountain Car (MCAR), and Double-Arm Pendulum (DAP). These were discretized using a procedure similar to that described by Munos and Moore (Muños & Moore 2002) to generate several large, discrete MDPs. For all experiments, the optimal value function was computed to within $\epsilon = 0.0001$ with $\gamma = 0.6$ and $V_0 = 0$. Both states and partitions were generated with regularly spaced grids in the problem space. We note here that all of the experiments are cache simulations, which is why actual wallclock times are not discussed.

The experiments yielded striking results. On the positive side, every experiment indicated that the *H2* metric greatly outperforms the *H1* metric, sometimes by as much as a factor of two. Additionally, reasonable cache efficiencies (above 80%) can be achieved using a cache capacity of only 20%. On the negative side, we note that auxiliary partition accesses are expensive vis-a-vis partition caching, and that the information-frontier-only cache performances could be improved.

All of the experiments yielded very consistent results. Figures 1, 2, 3, and 4 all show that the *H2* metric *dramatically* outperformed the *H1* metric in terms of cache performance. These efficiencies directly affect the number of misses per partition. For example, at a capacity of 22%, and counting both information-frontier and auxiliary accesses, the *H2* metric generates an average of 3.08 misses per partition. The *H1* metric, on the other hand, generates 22.02 misses. For information-frontier accesses only, *H2* generates 2.13 misses, while *H1* generates 10.37.

This general pattern of results holds for all problems, for cache sizes from 0% capacity to about 80% capacity. After about 80% capacity, the *H1* metric sometimes yielded better cache efficiency, but not by much. At 100% capacity, both algorithms yielded a 100% cache hit ratio, as expected.

We also note that the marginal cache efficiency equals one at a cache capacity of about 22%. This is the point of diminishing returns, where one must add more than one unit of cache capacity to increase the cache hit rate by one unit. However, at 22% capacity, the *H2* metric yields a 90.92% hit rate on Mountain Car and 86.03% on Double-Arm-Pendulum. This is quite good, and implies that problems which are four times larger than available RAM can be efficiently solved.

There are two negative items of note on the "information-frontier-only" series of graphs (Figures 3 and 4). First, we note that the scale of the "Average cache misses per partition" axis is consistently an order of magnitude less for the information-frontier-plus-auxiliary-accesses experiments! This implies that auxiliary partition accesses are expensive vis-a-vis cache efficiency. Although we can still achieve a good *percentage* cache hit rate, the *absolute value* of misses is very high. Additionally, the cache hit rate in the information-frontier-only series of experiments is much lower than is desirable. More sophisticated caching strategies may alleviate this.

Conclusions and Future Research

Based on our results, several conclusions are possible. The first conclusion treats the cache performance of the *H2* priority metric compared to the *H1* metric. Other research (which is in submission) has thoroughly examined the relative wallclock performances of the *H1* and *H2* metrics. Generally, the *H2* metric outperforms the *H1* metric. In terms of cache efficiency, however, *H2* always bested *H1*. For a general problem, therefore, whose characteristics are not known beforehand, it makes sense to select the safest option, which appears to be *H2*. The performance benefits of the *H2* metric, combined with its cache efficiency, solidify *H2* as the priority metric of choice for solving very large MDPs. Of course, our experimental domain is fairly limited; experimentation on more problems is necessary to truly justify this claim.

As mentioned previously, the idea of approximate inter-partition priorities could have profound benefits. Experimentally, the average number of cache misses per partition increased by an order of magnitude when auxiliary partition accesses were considered, which is quite expensive. Examining approximate inter-partition priorities could therefore be very fruitful. Another direction for future research is to benchmark different types of caches. As noted previously, normal VI can greatly benefit from an intelligent predictive cache, due to the regularity of partition accesses. It seems clear that partition accesses are also quite regular for P-EVA, although they are regular in a more complicated way. A more sophisticated predictive cache that leveraged this regularlity would further improve performance.

The most significant result of the paper is of a holistic nature. Not only do prioritization and partitioning accelerate convergence, but we discover that the combination of the two ideas makes efficient non-predictive caching viable. In addition, as explored in other research, partitioning and prioritization enable efficient parallel implementations of efficient MDP solvers. Thus, the ideas of prioritization and partitioning are complementary, and even synergistic. Additionally, the *H2* metric not only yields superior times to convergence, but it also exhibits outstanding cache efficiency. The combination of these results solidify the choice of partitioned, prioritized value iteration, using the *H2* priority metric, as the algorithm of choice when solving very large MDPs.

References

[Moore & Atkeson 1993] Moore, A. W., and Atkeson, C. G. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13:103–130.

[Muños & Moore 2002] Muños, R., and Moore, A. W. 2002. Variable resolution discretization in optimal control. *Machine Learning* 49 (2-3):291–323.

[Wingate & Seppi 2003] Wingate, D., and Seppi, K. D. 2003. Efficient value iteration using partitioned models. In *Proceedings of the International Conference on Machine Learning and Applications*, 53–59.

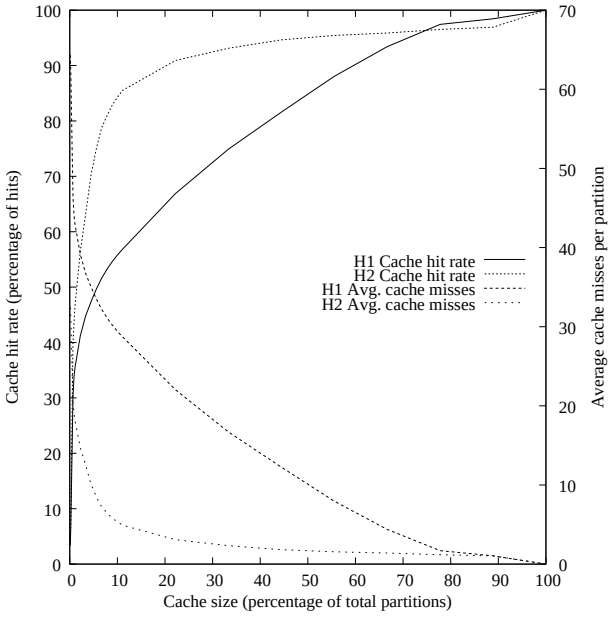

Figure 1: On the left axis, cache performance for the *H1* and *H2* metrics on the MCAR problem. On the right axis, the average number of cache misses per partition. A higher cache hit ratio is better, but a lower number of misses is better. For this problem, a 300x300 state discretization was used, and a 30x30 partition discretization was used.

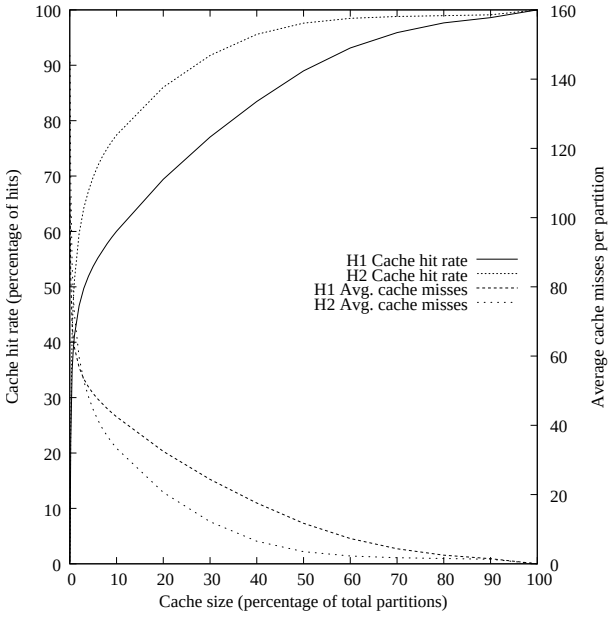

Figure 2: Cache performance results for the DAP problem, using a 30x30x30x30 state discretization and a 10x10x10x10 partition discretization.

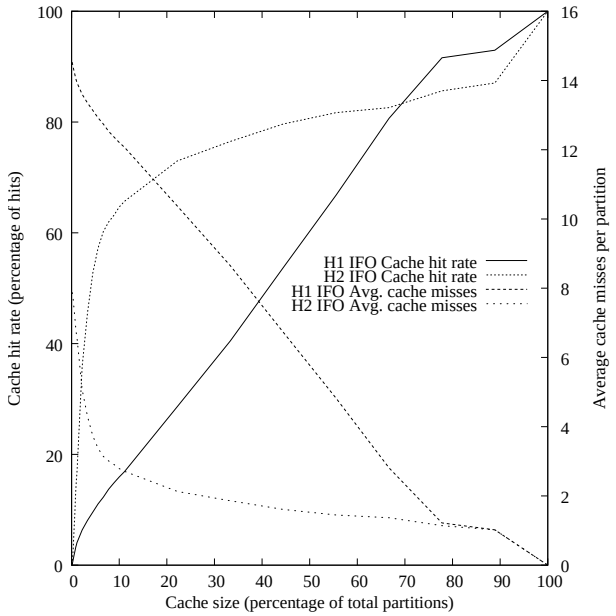

Figure 3: Information-frontier-only cache performance for the MCAR problem, using a 300x300 state discretization and a 30x30 partition discretization.

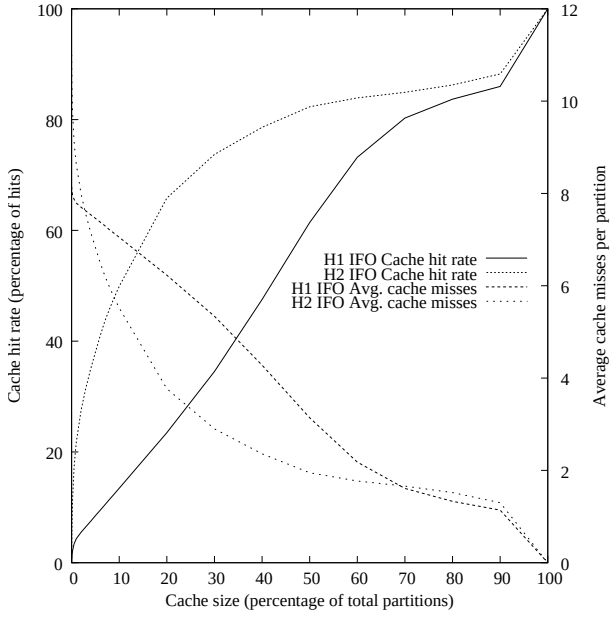

Figure 4: Information-frontier-only cache performance for the DAP problem, using a 30x30x30x30 state discretization and a 10x10x10x10 partition discretization.