Variable Resolution Discretization in the Joint Space

Christopher K. Monson, David Wingate,
and Kevin D. Seppi
{c,wingated,kseppi}@cs.byu.edu
Computer Science, Brigham Young University

Todd S. Peterson
petersto@uvsc.edu
Computer and Networking Sciences,
Utah Valley State College

Abstract

We present JoSTLe, an algorithm that performs value iteration on control problems with continuous actions, allowing this useful reinforcement learning technique to be applied to problems where a priori action discretization is inadequate. The algorithm is an extension of a variable resolution technique that works for problems with continuous states and discrete actions [6]. Results are given that indicate that JoSTLe is a promising step toward reinforcement learning in a fully continuous domain.

1. Introduction

Reinforcement Learning (RL) can be a useful way of representing control problems because of its simplicity. RL solution techniques, such as value iteration, can discover complex solutions that may be difficult to represent or compute in closed form. In the specific case of value iteration, a problem is solved by computing an approximation to the *value function*. The value function may then be used to create a control policy. [4]

One of the more attractive properties of value iteration is that it may be performed iteratively. The value of each point in space may be updated according to an equation like the following:

$$V(\mathbf{s},\mathbf{a}) = \int_0^\tau \gamma^t R(\mathbf{s}(t),\mathbf{a})\,dt + \gamma^\tau \sup_{\mathbf{a}'} V(\mathbf{s}(\tau),\mathbf{a}') \qquad (1)$$

where $V(\mathbf{s},\mathbf{a})$ denotes the value of the initial state $\mathbf{s}$ and action $\mathbf{a}$, $\mathbf{s}(t)$ is the state resulting from the application of $\mathbf{a}$ for t units of time, $R(\mathbf{s}(t),\mathbf{a})$ is the current reinforcement, $\gamma \in [0,1)$ is the discount factor, and τ is the amount of time for which $\mathbf{a}$ is executed. Note that this assumes a deterministic environment, so transition probabilities have been omitted.[1]

[1] Additionally, even though this definition of the value function looks like a *quality function*, we maintain the nomenclature of value iteration so as to avoid confusion with Q-learning.

Implementing (1) directly is impossible. In a discrete setting, (1) becomes something like the following:

$$V(\mathbf{s},\mathbf{a}) = \phi \sum_{t=0}^{T-1} \gamma^{t\phi} R(\mathbf{s}(t\phi),\mathbf{a}) + \gamma^{T\phi} \sup_{\mathbf{a}'} V(\mathbf{s}(T\phi),\mathbf{a}') \qquad (2)$$

where ϕ is a problem-specific time step.

With discrete actions, the $\sup$ operator becomes a $\max$ operator and a simple linear search is sufficient to implement it. In domains with continuous actions, however, the presence of the supremum is problematic, as a perfect implementation would require an exhaustive search of an infinite space. Fortunately this issue does not plague all continuous action RL problems; a well-known result from optimal control theory states that many minimum time control problems may be solved optimally using only bang-bang control [2]. This fact allows many researchers to optimally discretize continuous action spaces a priori.

Though many interesting problems may be solved using a naive discretization, many others may not. Even if a problem may be solved using bang-bang control in simulation, the policy generated can rarely be run on real hardware. A method for solving these problems without a static action discretization is needed. This paper presents the Joint Space Triangulation Learner (JoSTLe), which enables value iteration to solve problems with continuous actions. It is based on Variable Resolution Discretization (VRD) as presented by Muños and Moore [6], and relies on the same fundamental observation that not all portions of the problem space are of equal importance.

JoSTLe uses a homogeneous data structure to dynamically allocate resources across both state and action spaces. In the same way that VRD allows each problem to dictate regions of interest in the state space, JoSTLe allows each problem to dictate those regions in the combined state-action (or "joint") space. This creates the possibility for discretizing actions differently at each state.

This is not the first work to address continuous action problems, but it is to our knowledge the first to work with *general* continuous state and action problems. Other ap-

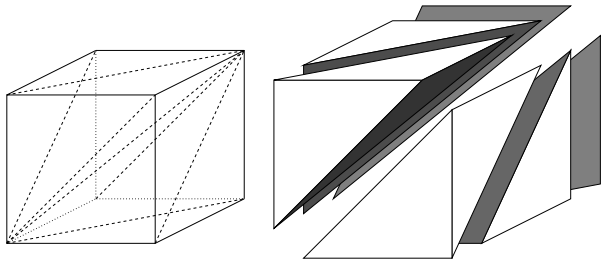


Figure 1. The Kuhn Triangulation of a cube

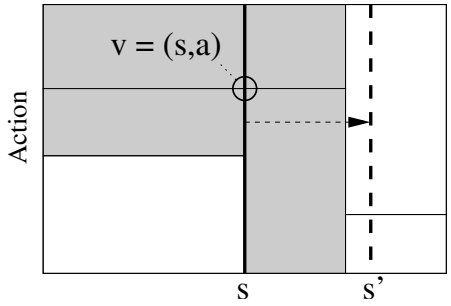


Figure 2. Final state exits adjacent hypercubes (shaded)

proaches typically involve using one discretization technique on the state space and then performing some form of regression in the action space [1] at each state. This approach is useful, but can run into problems when the value function has discontinuous boundaries. It can also be inefficient when one action representation could span multiple states.

2. Basic Variable Resolution Discretization

VRD discretizes the d_s-dimensional state space into hypercubes, arranged hierarchically in a kd-trie. The root node covers the entire space. At every branch, a split is performed in one of the state dimensions, creating two smaller hypercubes. A Kuhn triangulation is implemented at each leaf, effectively splitting it into $d_s!$ simplices (Figure 1). The overall effect is a complete triangulation of the space. The value function is interpolated linearly within this triangulation using barycentric coordinates [6].

VRD proceeds in two phases: *value iteration* and *state space refinement*. It begins with a rough initial discretization of the state space. The discretization is used to perform value iteration until it converges. The information gained from the value iteration process is used to refine the discretization, ideally splitting in areas that require finer representation to generate a good policy. The iteration and refinement steps are repeated until a satisfactory representation of the value function is obtained.

The kd-trie allows for efficient point localization while the Kuhn triangulation allows for efficient interpolation. A full description of the algorithm with comprehensive citations on component elements is contained in [6].

3. Extending to the Joint Space

JoSTLe is an extension of VRD but retains many of its characteristics, including the kd-trie and Kuhn triangulation. The primary difference between the two is that JoSTLe works in the joint state-action space rather than just the state space.

Let d_s be the dimensionality of the state space and d_a be the dimensionality of the action space. We may then define the joint space as the Cartesian product of state and action spaces, yielding a space whose dimensionality is given by $d = d_s + d_a$. This joint space is tessellated by hypercubes in the same manner as VRD's state space. Each vertex of each cube is composed of the concatenation of state and action vectors: $\mathbf{v} = (\mathbf{s}, \mathbf{a})$. Associated with each vertex is a value $V(\mathbf{s}, \mathbf{a})$, sometimes abbreviated as $V(\mathbf{v})$.

This extension into a higher dimensional space represents the primary mathematical difference between JoSTLe and VRD. Equation (2) still applies directly. There is another minor difference in the way that a trajectory's stopping point is determined. In VRD a trajectory stops when it exits the initial *simplex*. JoSTLe stops once the state no longer intersects any *hypercubes* adjacent to the initial state/action point. Figure 2 illustrates this idea. The circled vertex is the starting point, and the trajectory in the state space does not end until it is no longer intersecting any of that vertex's hypercubes. This paper addresses the issues of implementing such a system.

The addition of action dimensions to the discretization poses some unique problems that require attention. The first issue is that of finding actions at each state that produce the maximum value. The second issue is related to the generalization of the search algorithm to arbitrary dimensions. The third is that of deciding when and how to split hypercubes. These issues are addressed in the next sections.

3.1. Searching for Maxima

The search for actions that produce the maximum value at a given state is a fundamental part of value iteration. In traditional value iteration the action space is discrete and homogeneous. In the discretizations produced by JoSTLe the action space is continuous with a heterogeneous discretization. The piecewise linear interpolation imple-

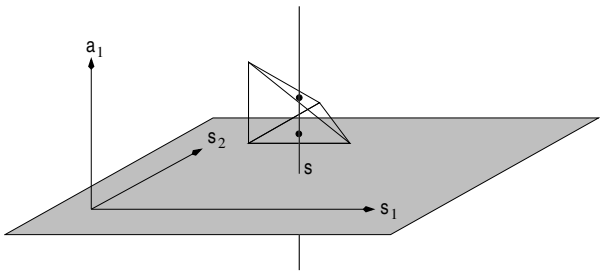


Figure 3. A joint space simplex

mented in JoSTLe addresses this problem effectively.

Figure 3 illustrates a joint space with two state dimensions and one action dimension. The tessellation generated by the kd-trie would actually cover the entire space with hypercubes, each of which would be triangulated, but for purposes of explanation only one triangle is shown.

The action search problem may be viewed as a search along the line in the figure. The line represents a region of constant state and variable action. Regions of this nature must be searched at every step of value iteration in order to calculate a discounted value. While in general this is a nonlinear programming problem, the representation allows for significant simplification. Because the interior values of each simplex follow a linear function, the maximum must occur at a simplex boundary. It cannot occur uniquely at the interior. This is easily proven by showing that the gradient of the interpolation function is constant. The proof that this holds even when constrained along a region such as that in Figure 3 is also fairly straightforward [5].

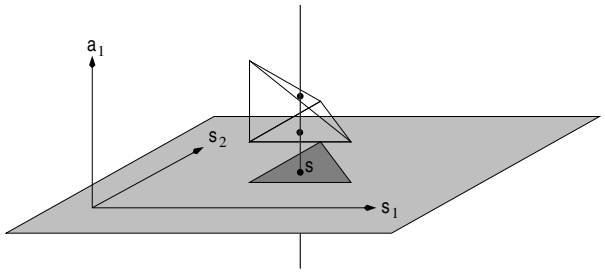
This insight allows us to restrict our search to the points where the line (as depicted in Figure 3) intersects with the simplex boundaries, effectively transforming a continuous problem into a discrete problem. The search is performed by finding the intersection points and picking the maximum.

3.2. Generalization to Arbitrary Dimensions

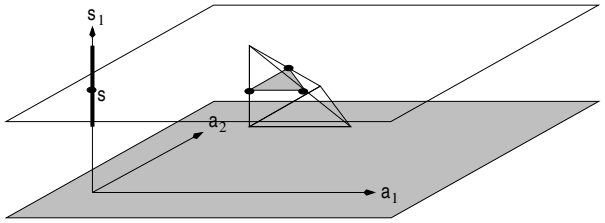
Finding the intersection points can be difficult. Figure 4 illustrates the problem and an insight that serves to generalize the algorithm.

Each possible spatial intersection generates a different kind of search space. In Figure 4(a) the regions of interest are points on a line. In Figure 4(b), however, the interesting portions of the space are vertices of a triangle formed by slicing a tetrahedron with a plane. In general, we are always seeking the vertices of a simplex formed from such a slice, and the shape of that simplex will change based on the dimensionalities of the state and action spaces.

Linear interpolation once again allows us to take a useful shortcut. Figures 4(a) and 4(b) also show the projections of



(a) $d_s = 2, d_a = 1$



(b) $d_s = 1, d_s = 2$

Figure 4. Joint space projections

the simplex into the state space. Note that in both projections, the region of interest is a single point in overlapping simplices. Again, because interpolation is linear, the points of intersection are easily found by projecting the boundaries onto the state space and performing interpolation on these lower dimensional shadows. Though boundaries will overlap in the lower dimensional space, each will have a unique set of vertices and produce different answers. One of these will be the maximum.

The algorithm for searching the action space at a given state $\mathbf{s}$ thus becomes very simple and elegant:

- Find all hypercubes intersected by the hyperplane at $\mathbf{s}$. This is easily done with a kd-trie using an orthogonal range query.
- For each hypercube, enumerate all Kuhn simplices.
- For each simplex, enumerate all d_s-dimensional boundaries.
- Project each boundary into the state space and interpolate at $\mathbf{s}$.

The proof that this method is equivalent to finding intersections in the joint space is given in [5].

3.3. Splitting Criteria

In principle, it would be best to refine the joint space only where it will improve the policy. Unfortunately, this is

not generally computable in advance. An alternative would be to refine regions of the space that improve the value function estimate. This is also not generally knowable, but an approximation may be made.

We refine the model based on maximum interpolation error. The error function is defined over all points $\mathbf{p} = (\mathbf{s}, \mathbf{a})$ within each simplex $\mathcal{S}$ as

$$\mathbf{E}(\mathbf{p}) = \left| V(\mathbf{p}) - \widehat{V}(\mathbf{p}) \right| \tag{3}$$

where $V(\mathbf{p})$ is the multistep *discounted* value at $\mathbf{p}$, and $\widehat{V}(\mathbf{p})$ is the *interpolated* value at $\mathbf{p}$, which is a weighted sum of the $V(\mathbf{p})$ values of the vertices in the enclosing simplex.

This error function is defined everywhere in the interior of each hypercube. The purpose of splitting is to reduce the maximum error. Splitting is therefore done when $\sup_{\mathbf{p}} \mathbf{E}(\mathbf{p})$ surpasses some suitable threshold ϵ.

In practice, $\sup_{\mathbf{p}} \mathbf{E}(\mathbf{p})$ is approximated with a set of random sample points $\mathcal{P}_{\mathbf{H}}$ within a hypercube by $\max_{\mathbf{p} \in \mathcal{P}_{\mathbf{H}}} \mathbf{E}(\mathbf{p})$. Whenever this value is greater than ϵ for a hypercube, the cube is marked for splitting. [2]

Once all cubes have been evaluated, the marked cubes are split. What remains is to determine in which dimension to split them. We split in the dimension that produces a new cube with the smallest maximum error. More precisely, let $\mathbf{C}_x^L(\mathbf{H})$ and $\mathbf{C}_x^R(\mathbf{H})$ be the left and right "children" resulting from a split of hypercube $\mathbf{H}$ in dimension x:

$$x = \underset{x' \in \{0, \cdots, d\}}{\arg\min} \left[\min_{D \in \{L, R\}} \left(\max_{\mathbf{p} \in \mathcal{P}_{\mathbf{C}_{x'}^D}} \mathbf{E}(\mathbf{p}) \right) \right] . \tag{4}$$

This may result in two child cubes with very different errors. The split is performed so as to minimize the maximum error of *one* of the cubes, leaving the possibility that the other will have a very high error. This is tolerable because the high error cube is likely to be split at the next iteration to improve its error characteristics.

No splitting is done if the hypercube has low error or if the hypercube is smaller than the smallest feature of interest, a parameter described in the next section.

3.4. JoSTLe Parameters

JoSTLe adds a small number of tunable parameters to the standard value iteration algorithm, shown in Table 1.

The minimum feature length in dimension i is denoted ω_i. This parameter determines when a cube is too small to be split regardless of error and it serves to keep the algorithm from splitting forever around discontinuous boundaries in the value function. This parameter is typically easy to obtain given that many reinforcement learning problems have known reward boundaries. The smallest reward feature size is often a good starting point for this parameter.

[2] Random sampling is a naive and simple way to approach the problem, but it is not likely to be the best. Other approximations are the subject of future work.

Table 1. JoSTLe parameters

ω_i	Minimum feature length in dimension i
Ω	Minimum Lebesgue measure of a hypercube
ϵ	Error threshold
σ	Number of sample points per hypercube

In practice, the performance of the algorithm degrades smoothly as this parameter is increased (as splits are limited). Smaller values always yield better accuracy, but often at the expense of convergence speed.

The ω_i parameter may be used in a number of ways. It can limit a cube's ability to split in a particular dimension if its length in that dimension is less than ω_i. Alternatively, it can be used to compute a minimum allowed Lebesgue measure $\Omega = \prod_{i=1}^{d} \omega_i$. If this latter method is used, then a cube is not split in any dimension if its Lebesgue measure is smaller than Ω. The experiments outlined in this paper use the former, though the latter was tested with similar results.

The error threshold ϵ is also used during the splitting process. If a cube's maximum error is less than ϵ, the cube need not be split. Determining an appropriate value for ϵ can be challenging, but a practical approximation may be made based on the maximum range of reward values $R_{\max} - R_{\min}$, the time step ϕ, and the discount factor γ.

If the problem has only terminal reinforcements, an upper bound on the error is given by $\phi(R_{\max} - R_{\min})$ since the integral over trajectory rewards will be zero until the end of the trajectory. Although this is a fairly conservative bound, it works well in practice.

Problems with non-terminal reinforcements may use an alternate upper bound, determined by accounting for an infinite string of discounted rewards: $\frac{\phi(R_{\max} - R_{\min})}{1-\gamma}$.

These upper bounds can be used to define a more intuitive error threshold. For example, the threshold may be set to some fraction of the upper bound, making it easy to generate reasonable values. 1% of the maximum error is often a reasonable error threshold.

The number of points σ scattered in a given hypercube may be computed from the minimum Lebesgue measure:

$$\sigma = \left\lceil \frac{\Omega_{\mathbf{H}}}{\Omega} \right\rceil \tag{5}$$

where $\Omega_{\mathbf{H}}$ is the Lebesgue measure of hypercube $\mathbf{H}$. The smallest allowable feature is allocated exactly one point. If σ becomes 1, then the hypercube can no longer be effectively tested for splitting.

Initially, the number of sample points can be very large, resulting in a substantial increase in time spent sampling

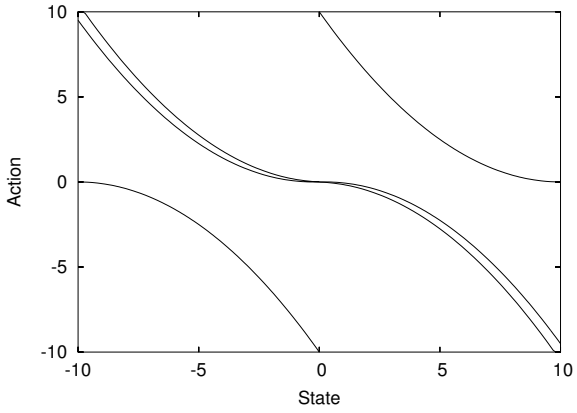
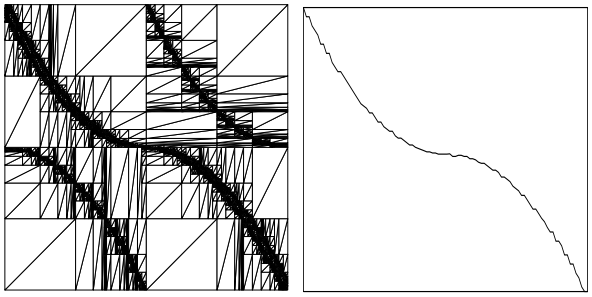


Figure 5. 1D Golf reward boundaries



(a) Discretization (b) Policy

Figure 6. JoSTLe discretization and policy for 1D Golf

and testing for splits. This number may be limited to a reasonable maximum, e.g. 1000.

3.5. Convergence

Gordon addressed the issue of convergence at length, and showed that averaging function approximators will allow the value iteration process to converge [3]. Among these are barycentric interpolators, of which the linear interpolation method described here is one. Because value iteration is done separately from the refinement process, and we operate over a finite set of actions, Gordon's convergence result still applies.

4. Experiments

4.1. 1D Golf

One-dimensional Golf is a test problem with low dimensionality and a continuous state and action space. A golf ball is sitting on a one-dimensional line and must be hit into a hole in the center of the space. The state space is described by $\mathbf{s} \in [-10, 10]$. The action space is also continuous: $\mathbf{a} \in [-10, 10]$. The hole is centered at $(0, 0)$ and is 0.5 units wide. The environment is deterministic and accessible. The system characteristics are

$$\mathbf{s}_{t+1} = \mathbf{s}_t + \frac{\mathbf{a}}{|\mathbf{a}|}\sqrt{10|\mathbf{a}|} \ . \tag{6}$$

If the ball hits a wall, it stops and a reinforcement of -1 is received. If it lands in the hole (i.e., $\mathbf{s}_{t+1} \in [-0.25, -.25]$) a reinforcement of 1 is received. In all other cases, a reinforcement of 0 is received. A graphical representation of the joint space with the positive (in the center) and negative (in the corners) reward boundaries is shown in Figure 5.

This problem is more interesting than it first appears. The region of high reward is very small and nonlinear. Additionally, reinforcements are not located strictly at the boundary of the problem space, making them difficult for VRD to find.

4.2. Results

Both JoSTLe and VRD were applied to the golf problem. Though there are many possible splitting criteria for VRD, in a 1-dimensional problem *average corner value difference* works as well as any of them (more complex criteria are only helpful in higher dimensions [6]). In VRD, splitting occurred if the value difference was above 0.001, and in the case of JoSTLe, ϵ was set to 5% of the upper bound on the error. The timestep $\phi = 1$. Both used a γ of 0 since it was known a priori that only one step is ever needed.

JoSTLe began with a single joint space hypercube and learned the appropriate discretization over time. VRD began with a single line segment in the state space and was applied using several different uniform action discretizations. For each algorithm, policy accuracy was calculated after every round of splitting and iteration. Since the optimal policy is known for this problem and always consists of a single step, the accuracy was calculated by scanning the state space and querying the models for correct policy values. The accuracy is the ratio of correct actions to total states queried.

The policy obtained by the joint learner is shown in Figure 6. In all available states, a correct action is chosen (the accuracy is 100%). The discretization hypercubes are also shown in Figure 6; it is clear that the learner concentrated its resources on areas of sharp reward transition. This behavior is expected since $\gamma = 0$.

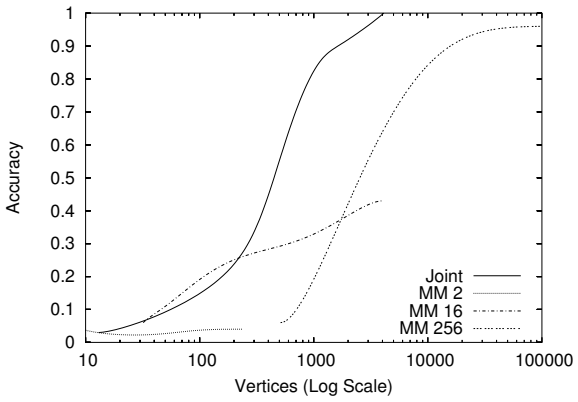


Figure 7. JoSTLe and VRD accuracy vs. number of vertices

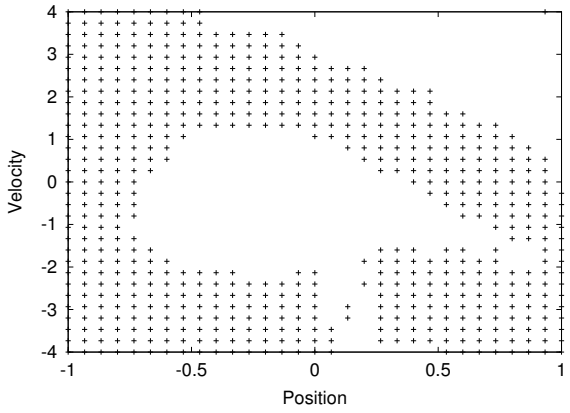


Figure 8. JoSTLe policy for Mountain Car

The accuracy of JoSTLe vs. the number of vertices used is shown in Figure 7. The behavior of VRD is shown on the same graph. The numbered labels indicate the number of discrete actions available to the algorithm throughout its trials. The graph shows that JoSTLe's policy accuracy went up quickly with every refinement, while the accuracy obtained with VRD rose slowly. It took 256 available actions to compare to the accuracy of JoSTLe, and far more state/action pairs. Given fewer actions, VRD peaks at a particular policy accuracy and then levels off, since finer action discretization is required but not available.

4.3. Additional Experiments

The first additional experiment was also done using the 1D Golf problem, altered so that the reward boundaries did not cover the entire state space. This required JoSTLe to do real value iteration (with a nonzero discount factor). The results were just as good as with the original problem.

Additionally, JoSTLe found a nearly optimal policy for the Mountain Car problem [6] without any prior knowledge either of what actions would be useful or of which parts of the state space were interesting. It learned that the two "bang-bang" actions are more useful than the others (full forward and full reverse). These turn out to be exactly the same two actions used in [6].

The policy learned by JoSTLe is shown in Figure 8. The policy is not perfect, but is close to optimal. Some artifacts exist and the reason for their existence is still being explored.

Alternate views of the policy are shown in Figure 9. Figure 9(a) shows the full policy and highlights the fact that JoSTLe focused most of its attention on two actions: full forward and full reverse. These are the bang-bang actions used by VRD. Figures 9(b) and 9(c) show the policy projected onto the Position/Acceleration and Velocity/Acceleration planes. The presence of points at the interior of these figures, rather than exclusively at the top and bottom, indicates that other suboptimal actions crept into the policy in some areas of the state space, a matter which needs to be studied further.

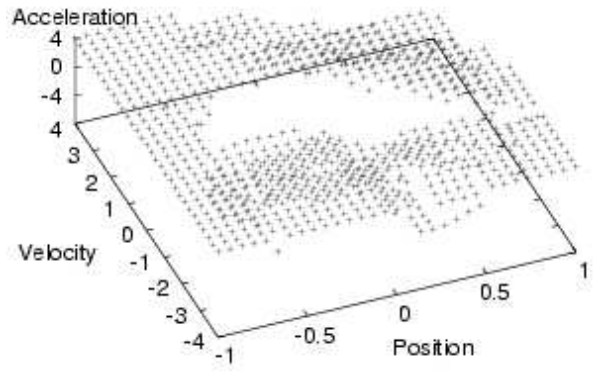
5. Liabilities

One unfortunate characteristic of JoSTLe when compared to VRD is its higher complexity. Because the dimensionality is increased, all of the worst space and time characteristics of VRD are exacerbated in JoSTLe (e.g. d is higher for JoSTLe, and each split still produces 2^d new vertices). Additionally, while VRD never enumerates the simplices of a hypercube, JoSTLe must. Each hypercube has exactly $d!$ Kuhn simplices, each of which JoSTLe must decompose into all $\binom{d}{d_s}$ boundaries. This has a negative impact on dimensional scalability.

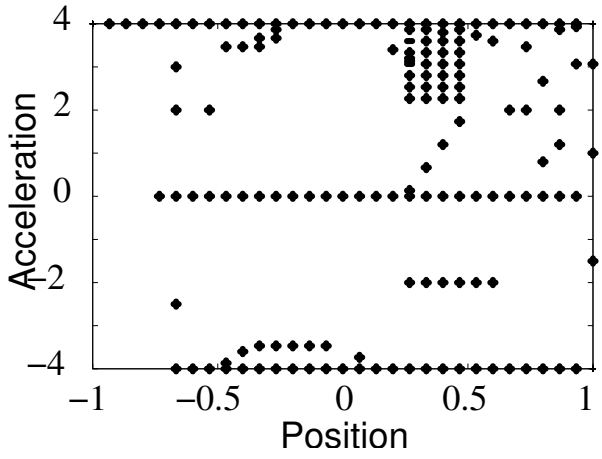
Some optimizations can alleviate the complexity, placing it on a more equal footing with VRD. The culling of degenerate and redundant simplices, as well as the fact that JoSTLe often needs fewer nodes overall can help to significantly reduce the complexity in practice. More work must be done to determine where else the complexity may be reduced.

Another problem is exposed by the fact that a perfectly optimal policy for Mountain Car was never achieved. Research revealed several potential areas of improvement. First, it is not clear that the splitting criterion presented here is exactly what is needed to generate a good policy.

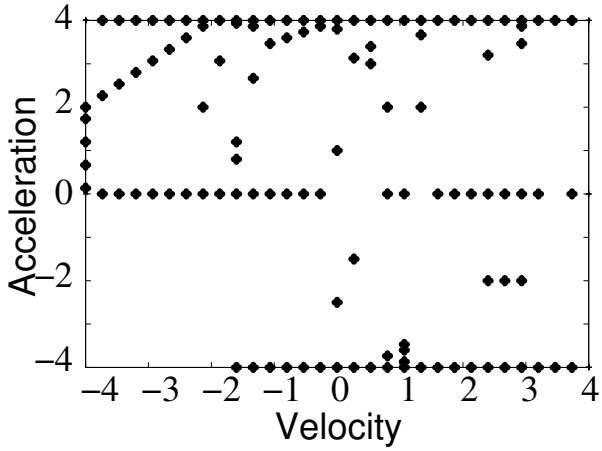
Second, though the integrity of the final value update equation was maintained throughout the development of the algorithm, it appears that one of the basic assumptions of value iteration may have been violated: the Markov property. VRD muscled itself into retaining this property



(a) 3D Policy



(b) Pos/Accel projection



(c) Vel/Accel projection

Figure 9. JoSTLe policy for Mountain Car

in a continuous space by treating interpolation weights as transition probabilities. JoSTLe has no similar interpretation of interpolated points, as it must first choose an action to determine the most likely current states. In other words, we don't know where we are until we go somewhere else, a clear violation of the requirement that our decision be based only on the current state. That JoSTLe works as well as it does indicates that the violation may not be serious, but it does merit further exploration and should be addressed in a future work.

6. Conclusions and Future Research

JoSTLe represents a productive step toward the ability to perform value iteration on problems with continuous actions. It provides a homogeneous framework for refinement of both states and actions and has an elegant appeal. Even so, there is much room for improvement.

The Markov property needs to be studied in greater detail in this context to determine whether there is a useful interpretation of JoSTLe that does not violate this property. Research in that area is ongoing.

Additionally, VRD proposed *influence* and *variance* to discretize only those portions of the space that affect the overall policy; more research could be devoted to an analysis of joint-space analogues.

Non-uniform splitting is another possible research direction. Both JoSTLe and VRD split regions of space in half, which is not always optimal in terms of efficiency. The ability to perform oblique splits may also allow for a more efficient representation. Some preliminary work in this area indicates promise, but more research is needed.

References

[1] L. Baird and A. Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical Report WL-TR-93-1147, Wright-Patterson Air Force Base, Ohio, 1993.

[2] B. Friedland. In *Advanced Control System Design*, New Jersey, 1996. Prentice Hall.

[3] G. J. Gordon. Stable function approximation in dynamic programming. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 261–268, San Francisco, CA, 1995. Morgan Kaufmann.

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

[5] C. K. Monson. Reinforcement learning in the joint space: Value iteration in worlds with continuous states and actions. Master's thesis, Brigham Young University, Computer Science Department, Apr. 2003.

[6] R. Muños and A. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49, Numbers 2/3:291–323, November/December 2002.