Variable Tile Splitting for Adaptative Tile Coding

Mathieu Nassif
School of Computer Science
McGill University
Montréal, Canada
Email: mnassif@cs.mcgill.ca

I. Introduction

Tile coding is used in situations where continuous functions must be approximated. In these scenarios, the developer is responsible for tuning several parameters to make sure the tile coding is efficient. These parameters include the number of tilings, the shape of the tiles (generally uniform rectangles, but nothing prevents other forms), and the size of the tiles.

These parameters will have an impact on the learning rate and the precision of the approach. Coarse tiles will propagate the updates very quickly, but will result in less precision in the limit. In contrast, very small tiles will give more precision, but the learning rate my greatly suffer from the slow propagation.

To address this problem, Whiteson et al. [1] designed an adaptative tile coding approach that takes a limited number of parameters. The approach starts with a small number of large tiles, e.g., a 2x2 grid for an environment with two features such as the mountain car task [2]. The initial tiling thus favors a rapid propagation of the updates, which lead to a quick initial learning rate. Then, as the values of each tile stabilize, the tiles are split. The new tiles can bring more precision to the value of the different states that were initially grouped together. This process is repeated until a time or precision limit is reached.

Whiteson et al. designed their original approach to trigger a tile splitting when the tile values seemed to be stable. The values are defined as stable when the last p consecutive updates changed the value of their tile by a larger delta than the minimum delta observed for each tile. When this situation occurs, only one tile is split, and in two exact halves along one axis. Therefore, when splitting, the approach evaluates which tile would lead to the most beneficial split, and in which axis. Then, the value at which to split is trivially determined as the middle of the tile.

Although their approach effectively reduces the number of decision that must be taken by the developers, there are still at least two decision that impacts the efficiency of the approach: the number p of stable updates to observe before splitting, and the strategy to select the right tile and axis to perform the split.

In their original paper, Whiteson et al. evaluated a value of $p = 50$, and suggested two ways to select the tile to split. In both ways, the approach keeps track of the value that would have each half-tile, in each dimension. For example, at some point in time, a tile may have a value of 14.2, but the upper half of the tile would have a value of 11.7 if it was separate from the lower half, which would have a value of 19.2. Similarly, the left half would have a value 16.1, and the right half, a value of 13.4. Then, the first option is to split the two halves that differ the most in value. The other option was to consider which split would have the highest impact on the associated greedy policy. Thus, the second option makes the split that would incur the most changes in the policy.

While the experiments on two classic two-dimensional problems show that this adaptative tile coding approach can effectively produce high quality approximations, with fast learning curves, comparable to the best fixed approaches, the two design decisions limit the applicability of their approach. Even in their evaluation, the *value criterion* for splitting tiles under-perform when compared to the other approaches.

Furthermore, the best place to split a tile may not be exactly in the middle in most cases. Therefore, to achieve a good precision, the approach must wait for several splits. This may take long, since the splitting rate is controlled by a lower bound.

In this paper, we present a new splitting approach for the adaptative tile coding method to solve the problems mentioned. In this approach, the only remaining parameter left is the value of p, although the heuristic to find the optimal value to split a tile could be modified without changing the general framework.

II. New Approach to Split Tiles Dynamically

Our new splitting approach makes two general changes from Whiteson et al.'s work. The tiles are no longer split one at a time, but rather each tile is responsible to trigger a split on itself, when its value has stabilized. Also, we no longer assume that the best split is in the center of a tile. Therefore, we must find the optimal value where to split a tile using a new heuristic.

Despite these conceptual changes, we keep the following assumptions from the original approach: every split will be made on a single axis, and we assume the value of a tile reached a stable level when the last p updates modified the value by a higher delta then the minimum delta observed for that tile.

The high-level algorithm of our approach is presented in Figure 1. We can see at line 5 that each tile gets its own update counter u_t that is updated only for that tile. Lines 17 to 26 show that the decision to update a tile is made locally, for each tile. This means that multiple tiles can be updated successively.

1: Initialize 1 tiling T
2: Initialize environment
3: **for all** tiles t in T **do**
4: Initialize t
5: $u_t \leftarrow 0$
6: **end for**
7: **repeat**
8: $s \leftarrow$ next state
9: $t \leftarrow$ tile containing s
10: $\delta \leftarrow \max_a(R(s,a) + \gamma V(T(s,a))) - V(s)$
11: $w_t \leftarrow$ value of tile t
12: $w_t \leftarrow w_t + \alpha\delta$
13: **for all** axis x **do**
14: update min, max and threshold for t and x
15: **end for**
16: $m_t \leftarrow$ minimal delta observed on t
17: **if** $m_t \leq \delta$ **then**
18: $u_t \leftarrow u_t + 1$
19: **if** $u_t > p$ **then**
20: split t
21: $u_t \leftarrow 0$
22: **end if**
23: **else**
24: $u_t \leftarrow 0$
25: $m_t \leftarrow \delta$
26: **end if**
27: **until** time expires

Fig. 1. High-Level Algorithm for New Tile Splitting (Adapted from Whiteson et al. [1])

Lines 14 and 20 are the core of this new approach. The decision of the axis to split and the evaluation of the threshold over which to split a tile are described in the remainder of this section.

A. Information to Keep for Each Tile

To make a good decision for the tile to split, each tile should ideally know the underlying value function over its domain. Of course, such a function is unknown, but if we could know it, the goal would be to split the tile so that the variance over each new fragment is minimized.

To estimate such a split, we can try to find the point where the value is exactly between the maximum and the minimum rewards observed on this tile, as in Figure 2. By putting the highest values on one fragment, and the lowest on the other, we can hope for an efficient way to group close values together, assuming the value function does not contain too much cliffs. Of course, with multi-modal functions, this approach would take multiple steps to properly split the tile. However, in the long run, we can expect the domain on the different tiles to be close to linear or piecewise linear.

Estimating this *middle* value is also challenging without storing all data points in memory. We must conserve only a constant number of variables for each tile, and the data points are read in sequence.

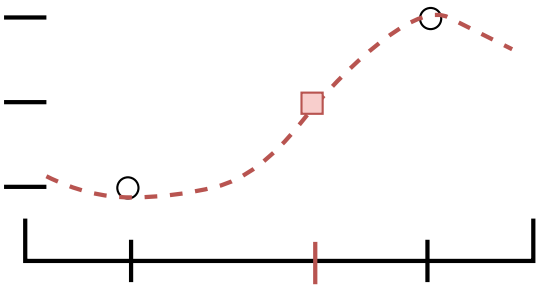


Fig. 2. Hypothetical Tile Viewed on Only One Axis. The circles represent the extremums of the value function (in dashed, unknown), and the square is the middle value. The objective is to find the x-position of the square.

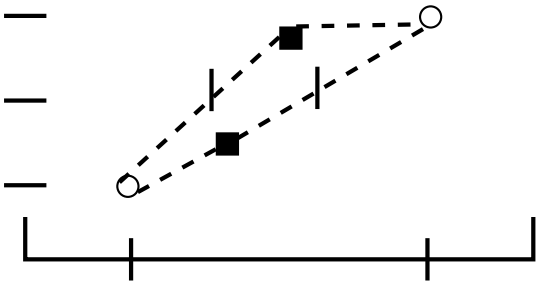
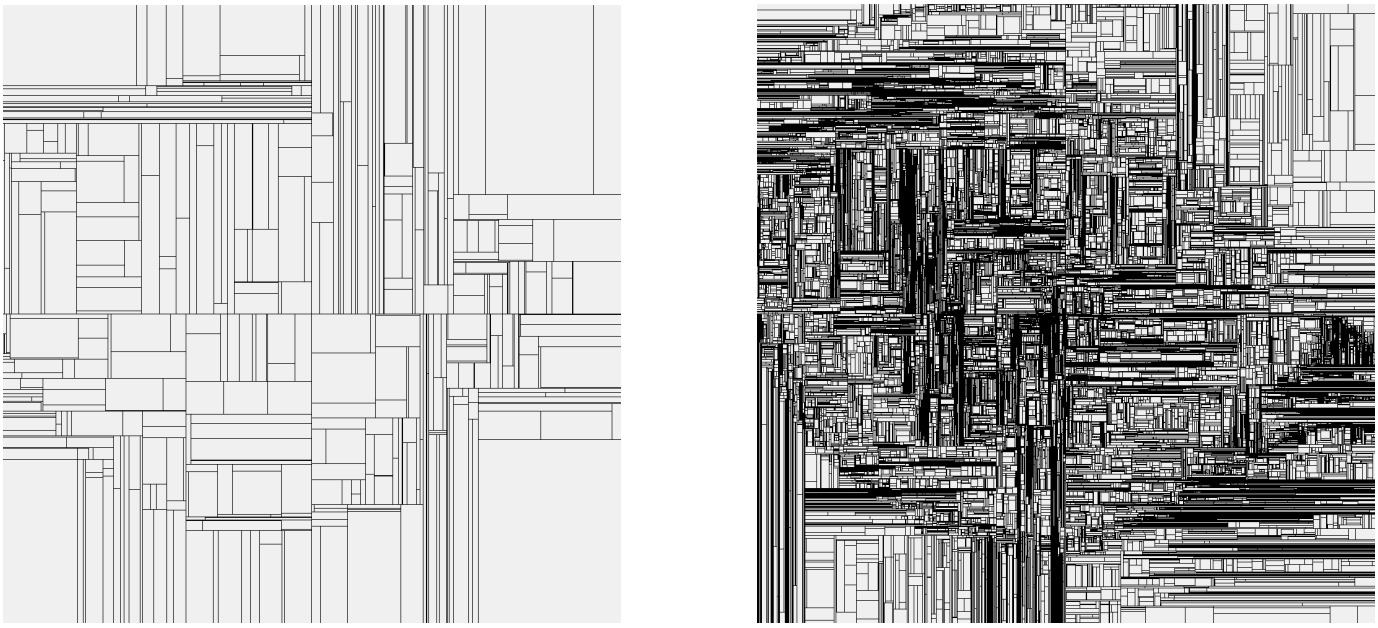


Fig. 3. Hypothetical Tile Viewed on Only One Axis. The circles represent the extremums fo the value function, and the squares are two observations. For each of them, we intrapolate the value function using lines, and estimate the middle value.

To address this issue, we propose the following procedure. Each tile has the following attributes: the maximum and minimum values M and m observed on the tile, the corresponding position vectors x_M and x_m where those extremums where observed, and two weight vectors w_M and w_m for these position vectors.

At every update, with a vector of features x and a value v, we first check if v is a new maximum or minimum value. If so, we update the correct extremum. If v is not an extremum, we check for each axis i if the corresponding element x_i of x is between the corresponding elements of the x_M and x_m. If it is not the case, the value of x_i is ignored for this axis. Otherwise, we update w_M and w_m in the following fashion: we use linear intrapolation with x_m, x and x_M to find the position p where the exact average of m and M should be, as shown in Figure 3. We than find two positive real numbers c_m and c_M such that (1) $c_m + c_M = 1$ and $c_m * x_{m,i} + c_M * x_{M,i} = p$. We finally add c_m and c_M to their respective weight vectors.

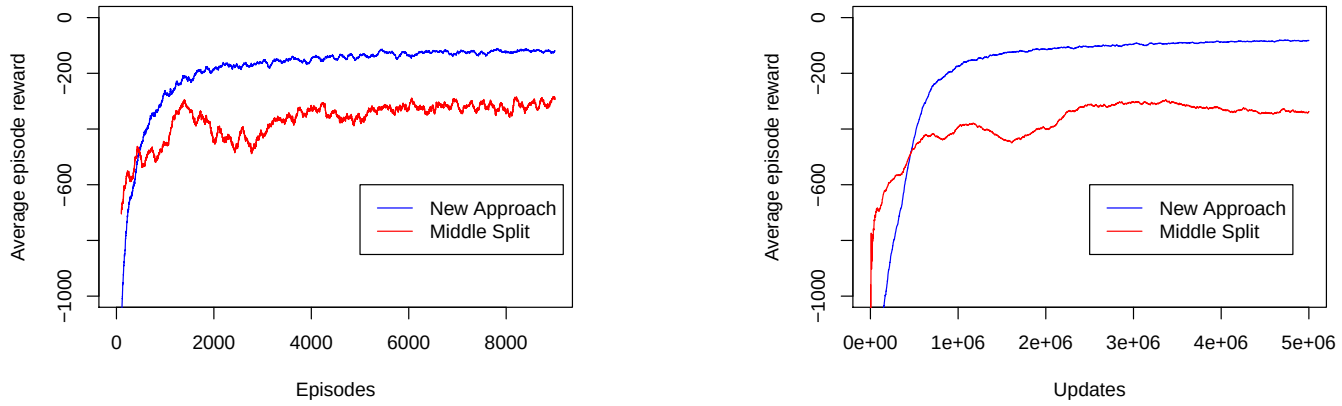
If v became one of the extremum, we follow a similar procedure, using the old extremum as the "new point" rather than v. If the old extremum is not between the two new extremums, we give them a weight of 0.5 each. Otherwise, we again use linear intrapolation, but with two middle points: the old extremum and the old estimated threshold (see Section II-C for how to compute the threshold from the information on each tile). We reset the weights on the extremums to the values of c_m and c_M, multiplied by the sum of the weights of the old extremums (so that the sum of the weights did not change due



(a) Tiling after 50 000 updates

(b) Tiling after 5 000 000

Fig. 4. Tilings Obtained For One of the Experiments With the New Approach



(a) Learning Rate Per Episode

(b) Learning Rate Per Update

Fig. 5. Comparison of the Learning Rates of the Original and the New Splitting Approaches

to the transformation).

This information depends on the order in which the data points are observed in each tile. However, this limitation should be mitigated by the fact that the value of a tile must be stabilized before the tile is split. Therefore, the best approximation of the optimal split will be given at the end, with, hopefully, more cumulative weight than the first, incorrect approximations.

In addition to this information, each tile also computes the variance in the values observed relative to the position in each axis. This variance is used to find the best axis to split.

B. Where to Split: Axis

Once a split action is triggered, our approach first selects the best axis on which to perform the split. The selected axis is the one where the observed variance is the lowest. The rationale behind this decision is that when the variance is high, this could mean either that the value have not fully stabilized *according to this axis*, or that slope of the value function is mostly oriented toward other directions. Therefore, splitting a tile on an axis with a higher variance would not be a good decision.

C. Where to Split: Value

Given the information available with each tile, and once the axis where to perform the split is selected, we select the weighted average of the extremums, with their respective weights, for the selected axis, as the threshold to make the split.

This way of splitting result in a biased estimator of the position where the value function is exactly between the two extremums. This estimator is biased toward the center of the positions of the two extremums. We perceive this bias as a positive consequence of our heuristic. It is an advantage, as it will prefer splits with two small fragments rather than one tiny fragment and one large fragment that will have to be split again and again. Our method also gives us the flexibility to change to a new extremum without necessarily losing all previous information.

III. Results

We evaluated our approach using the mountain car task. We compared our results with the original approach where tiles were split in two exact halves. We slightly modified the original by keeping the update count for each tile individual (i.e., tile splitting became a local decision rather than a global one, just like in our approach), to have a better comparison.

For our evaluation, we used the following parameters: $\alpha = 0.1$, $\gamma = 0.999$, and $p = 50$, in accordance with the original approach. We also limited the number of steps for each episode to 2000. If an episode would go over 2000 steps, we would keep all the updates of the episode, but reset the environment for the next iteration.

We first look at the tilings produced by our approach. Figure 4 shows the tiling obtained for only one experiment, in the early stages of the experiment (after 50 000 updates) and at the end (after 5 000 000). We can see that the tiling quickly focus on the center of the feature space. At the end of the experiment, the tiles in the center are very small, but the edges are less important.

Next, we performed 10 experiments for the original approach, and 10 more for our new approach. In each one of them, we performed as many episodes as necessary until we obtained 5 000 000 updates (each step leads to an update, so the sum of the length of all episodes in one experiment is 5 million). Figure 5 shows the resulting learning rates. In both graphs, the y-axis represent the average reward of the last N episodes. For the first graph, N was set to 100, and for the second, 500. The x-axis represent the number of episodes/updates performed previously. All curves show the average of value of the ten experiments.

We can see from these figures that our approach learns slightly slower than the fixed splits, but this delay is quickly regained, before either approach seem to converge. Then, our approach converges more quickly to a better solution than the fixed splits, which may show that it produces a better tiling, with a competitive learning rate.

In addition, our results show that our splitting approach lead to a lower variance in the results. Therefore, it may be better suited for environments where the optimal parameters to give to the traditional tile coding or the original adaptative tile coding are unknown. However, because we only experimented with the mountain car task, more work should be done before claiming that our approach is generalizable to other domains.

IV. Discussion and Conclusion

We presented a new approach, based on Whiteson et al.'s adaptative tile coding [1], to reduce even further the amount of design decisions a developer must make before using tile coding on an environment.

Using independent update counters for each tile, we removed a portion of the decision of when to update (now, the only parameter is the length of the stable period p), and there is no need to select only one tile to split from the whole tiling.

Using an estimation of the middle between the maximum and minimum value inside a given tile, we also make a better decision of where to split the tile, freed from the assumption that it should be split into two even halves. We select the axis based on the estimation of the variance of the value function over the tile. We estimate the position of the division with a constant number of values to store in memory, thus keeping a reasonable performance both in time and space.

Our results show promising improvement over the original adaptative tile coding, both in term of convergence and variance.

Currently, our method can generate a high number of tiles quickly, which could limit its application, especially in higher dimensions. Furthermore, environments where the value function has many cliffs in dense regions could be problematic for our approach, as each tile would have a multi-modal value function, which is hard to split. Future work is required to evaluate to which extent can our approach be generalized, and how to mitigate the rapid growth in the number of tiles. One possible direction could be using hashing on top of our tiling.

References

[1] S. Whiteson, *Adaptive Tile Coding*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 65–76. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13932-1_7

[2] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," *Advances in neural information processing systems*, pp. 369–376, 1995.