Proceedings of the 2002 IEEE
International Conference on Robotics & Automation
Washington, DC • May 2002

Satisficing Anytime Action Search for Behavior-Based Voting

Thomas J. Palmer
tjpalmer@tjpalmer.com

Michael A. Goodrich
mike@cs.byu.edu

Computer Science Department
Brigham Young University
Provo, UT 84602

Abstract

Within the field of behavior-based robotics, a useful and popular technique for robot control is that of utilitarian voting, where behaviors representing objectives assign utilities to candidate actions. Utilitarian voting allows for distributed, modular control but is not well suited to systems with very high resolution or several, dependent degrees of freedom. Past systems have depended on exhaustive searches that choose the action with the highest combined utility. We present, instead, a framework that casts the action search problem as an anytime algorithm. This allows directed searches to be used with time limit constraints. Aspiration-based satisficing additionally helps to reduce CPU usage. In experiments so far, we have decreased CPU usage by up to 42% while causing result quality to be no more than 6% worse.

1 Introduction

Behavior-based robotics is a field which for years has focused on bottom-up intelligence and addressing real-world environments [1]. A common feature of behavior-based control systems is that of modular behaviors operating concurrently to decide the actions taken by the robot. Among the different techniques used for fusing the decision process of multiple behaviors, *utilitarian voting* is popular and allows especially robust decentralization [2, 3, 4, 5, 6, 7, 8].

The behaviors in a utilitarian voting (or simply, *voting*) system assign utilities to candidate actions. These utilities are then combined across behaviors in some way, usually by a sum or a weighted average. The action with the highest combined utility is then taken. This process repeats at each time step. Because of its simplicity, such voting is used in far more situations than can be easily recognized. However, some research is very salient, such as Rosenblatt's Distributed Architecture for Mobile Navigation [6] and Pirjanian's work on voting from the perspective of multiobjective decision theory [4].

In many cases, utility maximization is an unrealistic expectation [9, 10]. Whenever multiple degrees of freedom (or *action dimensions*) are dependent, the growth in the number of potential actions is often exponential. Very high resolutions of actuator control add to the difficulty of finding optimal actions. Utilitarian voting provides a number of benefits for robot control, but as robot complexity increases, traditional voting will not be able to address the full needs of control systems. It should also be possible to use voting at the core of a system without exhausting CPU resources; the behaviors themselves may have processing-intensive algorithms.

Current voting architectures (including those already cited) do not allow the voting system to have full control of multidimensional robot actions. Rosenblatt, for instance, performs voting on each dimension separately using reduced actuator resolutions. As an example, sharp turns require slower speeds, and he adds such constraints as ad hoc rules outside any voting. He increases effective resolution through interpolation but without any guarantees that interpolated actions are actually better. In general, current systems require that all actions be considered at every time step, and blind search techniques are made possible only through such reductions of the search space.

We propose extending current voting methodologies by abstracting the action search process. We cast the action search as an *anytime algorithm*, a term first introduced by Dean and Boddy [11]. That is, the search for the next action to be taken should be able to be stopped at anytime. Rather than trying to blindly loop through all possible actions, a directed search can be used to find actions with high utility quickly.

Imposing a time limit may allow actions to be chosen sufficiently fast for system requirements, but time limits do not necessarily prevent the action search process from monopolizing the CPU. Processing-intensive behaviors may exist and may not have additional processors on which to operate. We adapt from [12] a *satisficing* mechanism based on an aspiration level that represents utilities of recently taken actions. The aspiration can be used for stopping searches or for knowing when a search is necessary in the first place. Satisficing is a technique essentially focused on finding *good enough* solutions.

Satisficing was first introduced in the 1950s by economist Herbert Simon [13] and has been formalized and analyzed in a number of ways [9, 14, 12, 15, 16, 17]. Of direct pertinence are Pirjanian's analysis of satisficing within voting systems [15] and Zilberstein's discussion of the relationship between satisficing and anytime algorithms [17]. In no voting system we have seen, however, has satisficing been used to avoid exhaustive action space search.

We organize this paper as follows. In section 2, we discuss each part of our solution in more detail. Section 3 shows how we have used our framework in an actual robot control system, and section 4 shows our results.

2 Framework

We apply utilitarian voting to the control of complicated robots with multiple, dependent, high-resolution action dimensions without exhausting CPU resources. We do this by means of (a) directed searches used in an anytime fashion and (b) aspiration-based satisficing.

In our satisficing anytime search framework, each behavior b consists of two operations:

$$\mu_b : S_w \times S_b \times A \rightarrow [0, 1] \tag{1}$$

$$\psi_b : S_w \times S_b \rightarrow A^n \tag{2}$$

μ_b is a function that assigns utilities to candidate actions and ψ_b is a function that provides suggestions to the search mechanism. S_w is the set of world states, S_b is the set of internal behavior states, and A is the set of all action vectors available to the voting system. n represents an arbitrary integer meaning that ψ_b can return any number of actions. An evaluation function like μ_b is found in most voting systems, while ψ_b is more unique in that it is only useful for directed search algorithms. Note that we constrain utilities to be between 0 and 1.

A positive weight is associated with each behavior, and the group utility of each action is determined by a weighted average of behavior votes:

$$\mu_B(s_w, s_B, a) = \frac{\sum_{b \in B} w_b \mu_b(s_w, s_b, a)}{\sum_{b \in B} w_b} \tag{3}$$

B is the set of all behaviors, w_b represents each behavior's weight, and a is any potential action vector. The resulting, combined utility remains between 0 and 1. Other types of utility combination, such as the multiplicative T-norm used by Roland Stenzel [8], are compatible with our search approach. Having some bounds on possible utilities is the most important aspect, since adapting an aspiration level would be harder without bounds.

At each time step, an action is chosen and the associated utility is used for updating the aspiration level. The initial aspiration level (before any actions have been chosen) is arbitrary but should be chosen such that it represents common utilities for the problem. When an action is taken, we update the aspiration level using a modified form of the equation used by [12]:

$$\alpha_{i+1} = (1 - r)\alpha_i + r\mu_B(s_w, s_B, a) \tag{4}$$

α_i is the aspiration level (always between 0 and 1) at the current time step, α_{i+1} is the aspiration at the next time step, and $r \in [0, 1]$ is the rate at which the aspiration level changes. Higher r results in more quickly adapting aspirations and therefore less memory of older time steps. Setting r to 0 results in a fixed aspiration, and setting r to 1 results in an aspiration always equal to the utility achieved at the last time step.

At every time step, a procedure is followed consisting of the following steps:

1. Read from the sensors the new state of the world.

2. If $\mu_B(s_w, s_B, a_{i-1}) \geq \alpha_i$, proceed to step 4 without search and set a_i to a_{i-1}, since the previously chosen action is still satisficing. If the previously chosen action is not satisficing, proceed to the next step.

3. Search the action space, using suggestions given by ψ_b for each behavior if the search algorithm supports the use of suggestions. End the search when the time limit expires or when some other search ending criterion is met. Set a_i to be the best action found by the search.

4. If the time limit has not expired (due either to ending the search early or skipping the search entirely), pause the decision process for the duration of the time limit.

5. Update α_i using equation 4 and a_i. Perform the selected action.

We allow searches to be ended early based on satisficing criteria. For instance, the search process could be ended as soon as any satisficing action is found. Figure 1 shows how anytime algorithms and an aspiration level work together. While ending a search early may seem similar to skipping the search entirely when the previous action is satisficing, there are some important differences which we discuss in section 4.

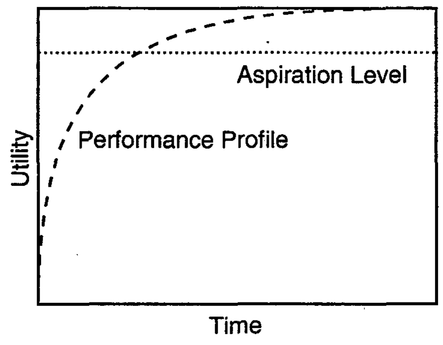

Figure 1: Aspiration-based anytime search during one decision step. As time increases, the utility of the best solution found increases. The aspiration level can be used as a cutoff resulting in a shorter search which still meets current expectations.

By pausing when a search ends before the time limit, CPU usage is reduced, and behaviors with processing in separate threads receive more execution time. This additional time is especially beneficial to behaviors that use anytime algorithms for their own computation because the amount of available time is not constant.

3 Implementation

We have developed an architecture based on the framework just described [10]. This architecture, called Bruvo 1 (Boundedly Rational Utilitarian Voting with Overrides version 1), fleshes out more details than what we have described here. We have implemented this architecture and developed a robot application using Nomad Super Scout mobile robots. In our application, the robot is used for autonomous wandering and semiautonomous goal-seeking tasks. Figure 2 shows a simulated world similar to the real hallway environment in which the robot operates. In section 4, we use this simulated world for comparing different search techniques.

The voting system is responsible for (a) controlling the robot's translational and turning velocities, (b) the acceleration (or deceleration) at which the velocity changes, and (c) the interval between firing sonars.

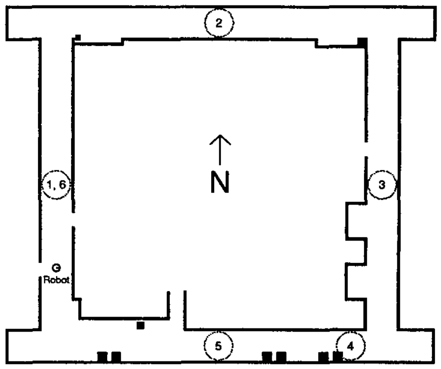

Figure 2: The simulated hallway world used for comparative tests. It is similar to the real environment in which the robot operates. Holes in the walls represent open doors to rooms which the robot does not enter. Block objects represent chairs and other obstacles. The circled numbers represent ordered goal locations. The robot is placed at the starting location and orientation.

In all, there are 501 different translational velocities, 301 turning velocities, 390 accelerations, and 255 sonar firing intervals, resulting in a total of about 15 billion potential action vectors. On the real robot, 0.05 seconds are given to find an action at each time step, and, in the simulator, 0.025 seconds are given. The simulator uses a faster computer; thus the shorter time limit helped the results be more comparable. Either way, a comprehensive search of 15 billion actions is obviously impossible in so short a time on modern personal computers. Therefore, the problem has been chosen to be adequately complex so that our boundedly rational approach can be represented properly.

Our robot control system is made of six utilitarian behaviors of the type two discussed here and two override behaviors that help prevent unacceptable actions from being taken [10]. The six utilitarian behaviors are:

1. *center-in-hall*

   This behavior uses a proportional-derivative controller based on the difference in distance to left and right sides of the robot to try to keep it centered in the hallway.

2. *move-forward*

   This behavior always wants to move forward at full speed and ignores other action dimensions. It has a small weight so that it can be easily outvoted.

3. *regulate-speed*

   This behavior tries to keep a minimum headway (time required to reach an unmoving point) of 2.5 seconds and assumes forward, straight line motion.

4. *silence*

   This behavior tries to maintain a short sonar firing interval when necessary but slows down the sonars when the robot is not moving. Short intervals equate to more rapid firing which is necessary for keeping updated information about the environment.

5. *seek-goal*

   This behavior uses a simple goal-seeking heuristic. It tries to orient towards the goal when nearby obstacles are not in the way. It has a heavy weight so that when there are disagreements about direction, the goal can still be reached.

6. *turn*

   This behavior tries to turn when the robot is confronted with an obstacle. It also tracks recent tendencies of the robot to turn right or left and favors the side with a strong trend if there is one.

We implemented three search methods:

1. Genetic Algorithm

   This simple genetic algorithm combines and modifies action vectors at the level of individual action dimensions. Behavior suggestions make up the initial search seeds.

2. Low Resolution

   This search reduces the resolution of the action dimensions so that the total number of actions is 5304, rather than 15 billion. Potential action vectors are iterated through in an undirected fashion. Resolution is most reduced in the acceleration and sonar firing interval dimensions; only a few settings in those dimensions are useful to any of the behaviors. Once the highest utility action vector is found, it and its neighbors are filtered through a quadratic interpolation. If the interpolated action is better than the original, it is used instead. This follows the interpolation pattern used by Rosenblatt [6].

3. Split Space

   This search loops through the different action dimensions one at a time, using the values from best action found so far for the other dimensions. All dimensions are looped through multiple times until no further changes are made. Using this technique, 1447 action vectors are considered per iteration through all dimensions. For our application, one loop through each dimension is almost always sufficient, but at least a partial second loop is needed to verify this.

The low resolution and split space searches provide the same kind of search characteristics found in traditional voting systems (e.g., [4, 6]), and both searches finish within the set time limits.

We also tested three heuristic criteria for ending searches:

1. End whenever an action is found whose utility meets or exceeds the aspiration ($\mu_B(a) \geq \alpha_i$).
2. End whenever an action is found whose utility meets what the aspiration would be at the next time step if 1.0 utility were achieved at this time step ($\mu_B(a) \geq (1-r)\alpha_i + r$).
3. End only if an action of 1.0 utility is found ($\mu_B(a) = 1$).

Results comparing the different search algorithms and search-ending criteria are discussed in the next section.

4 Results

In addition to the three search-ending criteria, we tested the search methods with a constant aspiration of 1.0 ($\alpha = 1$ and $r = 0$), which always causes the search to be performed until the time limit expires or until an action of utility 1.0 is found. We are most interested in comparison between CPU usage and result quality. Figures 3, 4, and 5 show the CPU time used per decision step, utility of taken actions, and time to complete a lap around the halls, respectively, for each of the configurations.

Utility results have been included because it is the most specific measure of how well a search performs. At the same time, the meaning of utilities is not intuitively obvious. For this reason, we have included the consequential measurement of lap time (i.e., the time to reach all six goals). Note that small decreases in utility can result in significant increases in lap time, although the exact correspondence is not always predictable.

Utility can also be translated into short term consequences. The exact effects depend on the voting pattern and the weight of each behavior. For instance,

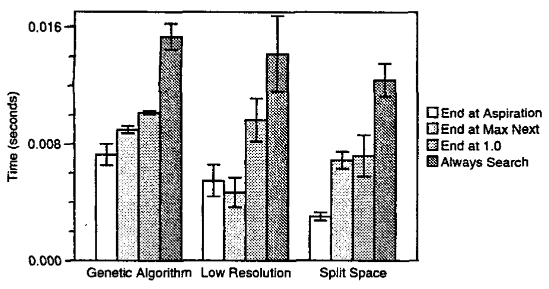

Figure 3: Mean CPU time spent per decision step. Error bars represent 95% confidence intervals.

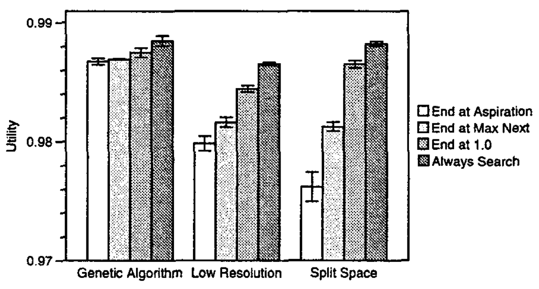

Figure 4: Mean utility of taken actions. Invisible error bars indicate intervals below plot resolution.

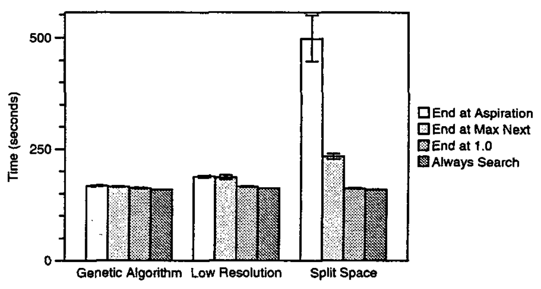

Figure 5: Mean time to complete one lap.

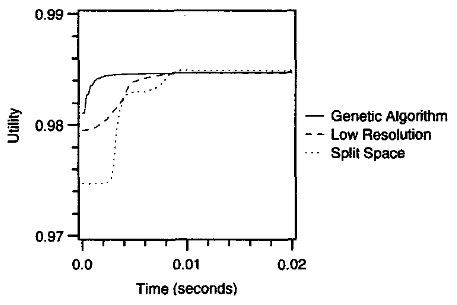

Figure 6: Mean performance profiles.

if 0.01 utility were lost to just the *move-forward* behavior, the robot would move at 9.5 in/s rather than at 25 in/s. If 0.01 utility were lost to *center-in-hall* and it wanted to go straight forward, the robot would instead have an offset velocity of 0.5 in/s to the left or the right. Such short term consequences help to provide an intuitive feel for the differences in utility seen in figure 4.

Figures 3 and 4 show that ending searches early significantly reduces the CPU time used by the genetic search without causing marked decrease in utility. On the other hand, the two undirected searches drop significantly in utility as the search time decreases. The split space search especially degrades when searches are ended early. The reason for these results can be seen in the performance profiles of the search algorithms (in figure 6), which show how utility increases with search time. Also noteworthy is that the low resolution search never reaches the highest utility of the other two because even with interpolation it cannot always find the actions with highest utility.

Another significant point also helps to explain the low utilities achieved by the undirected searches when the searches were ended early. Under some circumstances, the utility available to the voting system changes either up or down for almost all actions. For instance, when the *seek-goal* behavior is inactive because of impeding obstacles, it votes 1.0 for *all* actions. When *seek-goal* transitions suddenly from actively voting to being inactive, many poor actions have their combined utility increased. The aspiration level cannot adjust in one step, so many poor actions become suddenly "satisficing". Directed searches, like the genetic algorithm, do not often consider obscure, poor actions, but the undirected searches may begin their search in undesirable parts of the action space. Many low quality actions may still be considered satisficing, and the robot may behave in an undesirable fashion. For instance, the split space search has been observed to cause the robot to back up slowly for extended periods of time.

In summary, our satisficing approach is compared to currently used voting techniques in table 1. Without satisficing, the split space search performed the best and in the least time. The other three configurations shown had approximately similar lap times and were all within 6% of the best time achieved. At the same time, both the genetic algorithm and the split space search algorithms used about 42% less CPU than when no satisficing was used.

Searcher	Configuration	CPU Time	Lap Time
Genetic Algorithm	End at Aspiration	7.26 ± 0.74 ms	167.6 ± 1.2 s
Low Resolution	End at 1.0	9.64 ± 0.15 ms	166.0 ± 0.9 s
Split Space	End at 1.0	7.18 ± 0.14 ms	161.5 ± 1.3 s
Split Space	Always Search	12.42 ± 1.14 ms	159.3 ± 0.4 s

Table 1: CPU time compared to lap time for different configurations. The bottom line, split space always searching, represents the best results for current techniques normally used in utilitarian voting.

5 Conclusion

We have described how traditional approaches to action search with behavior-based voting either ignore dimensional dependencies or reduce the resolution of the actions for the voting process. This limits the benefits of voting when being used for full control of more complicated robots. We instead view voting as an anytime search process. Such a view allows the use of aspiration-based satisficing to reduce CPU usage significantly while minimally reducing the utility of taken actions. Abstracting the search process also allows directed searches that reach higher utilities in less time than low resolution and split space techniques.

One important direction for future work would be to extend and improve the types of heuristics used for ending searches so that when little additional improvement is likely to be made, the search can end. Work along these lines could show a stronger improvement in CPU usage over traditional techniques by taking better advantage of directed search performance.

Another important area of research would be in more complicated control systems where some behaviors could benefit from additional CPU resources. Such systems should be able to increase result quality by using our techniques rather than simply decrease CPU usage.

References

[1] Ronald C. Arkin, *Behavior-Based Robotics*, MIT Press, 1998.

[2] Barry Brumitt and Martial Hebert, "Experiments in autonomous driving with concurrent goals and multiple vehicles," in *IEEE International Conference on Robotics and Automation*, 1998.

[3] Jonas Karlsson, *Learning to Solve Multiple Goals*, Ph.D. thesis, University of Rochester, 1997.

[4] Paolo Pirjanian, *Multiple Objective Action Selection & Behavior Fusion using Voting*, PhD dissertation, Department of Medical Informatics and Image Analysis, Aalborg University, 1998.

[5] Jukka Riekki, *Reactive Task Execution of a Mobile Robot*, PhD dissertation, University of Oulu, 1999.

[6] Julio K. Rosenblatt, *DAMN: A Distributed Architecture for Mobile Navigation*, PhD dissertation, Robotics Institute, Carnegie Mellon University, 1997.

[7] Sanjiv Singh, Reid Simmons, Trey Smith, Anthony Stentz, Vandi Verma, Alex Yahja, and Kurt Schwer, "Recent progress in local and global traversability for planetary rovers," in *IEEE International Conference on Robotics and Automation*, 2000.

[8] Roland Stenzel, "A behavior-based control architecture," in *IEEE International Conference on Systems, Man, and Cybernetics*, 2000, pp. 3235–3240.

[9] Gerd Gigerenzer and Daniel G. Goldstein, "Reasoning the fast and frugal way: Models of bounded rationality," *Psychological Review*, vol. 103, no. 4, 1996.

[10] Thomas J. Palmer, "Boundedly rational utilitarian voting with overrides: An architecture for autonomous mobile robot control," M.S. thesis, Brigham Young University, 2001.

[11] Thomas Dean and Mark Boddy, "An analysis of time-dependent planning," in *National Conference on Artificial Intelligence (AAAI)*, 1988, pp. 49–54.

[12] Rajeeva Karandikar, Dilip Mookherjee, Debraj Ray, and Fernando Vega-Redondo, "Evolving aspirations and cooperation," *Journal of Economic Theory*, vol. 80, pp. 292–331, 1998.

[13] Herbert A. Simon, "A behavioral model of rational choice," *Quarterly Journal of Economics*, vol. 69, pp. 99–118, 1955.

[14] Michael A. Goodrich, Wynn C. Stirling, and Erwin R. Boer, "Satisficing revisited," *Minds and Machines*, vol. 10, pp. 79–110, 2000.

[15] Paolo Pirjanian, "Satisficing action selection," in *SPIE Conference on Intelligent Systems and Advanced Manufacturing*, 1998, pp. 153–164.

[16] Shinzo Takatsu, "Decomposition of satisficing decision problems," *Information Sciences*, vol. 22, pp. 139–148, 1980.

[17] Shlomo Zilberstein, "Satisficing and bounded optimality," in *AAAI Spring Symposium on Satisficing Models*, 1998, pp. 91–94.