Planning Under Time Pressure

Ethan Burns
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
eaburns at *cs.unh.edu*

Introduction

Heuristic search is a technique used pervasively in the fields of artificial intelligence, automated planning and operations research to solve a wide range of problems from planning military deployments to planning tasks for a robot that must clean a messy kitchen. An automated agent can use heuristic search to construct a plan that, when executed, will achieve a desired task. The search algorithm explores different sequences of actions that the agent can execute, looking for a sequence that will lead it to a desired goal state. In many situations, an agent is given a task that it would like to solve as quickly as possible. The agent must allocate its time between searching for the actions that will achieve the task and actually executing them. We call this problem *planning under time pressure*.

Classic heuristic search algorithms do not address planning under time pressure. A* (Hart, Nilsson, and Raphael 1968), one of the most well-known heuristic search algorithms, finds optimal plans that have the minimum execution time. (In general A* optimizes any cost metric, however, we are concerned with time so we assume that cost = time.) Unfortunately, it is often impractical or intractable to find an optimal plan. Other algorithms, such as greedy best-first search (Doran and Michie 1966), will quickly find solutions of unbounded sub-optimality. Unfortunately, these greedy solutions often lead to plans that are too time consuming to execute. Finally techniques, such as weighted A* (Pohl 1970) or more recently Explicit Estimation Search (Thayer and Ruml 2011), try to find a balance between optimality and unbounded sup-optimality by returning solutions that are guaranteed to be within a user-specified factor of optimal. It is not clear, however, how to choose a sub-optimality factor to properly trade-off planning time and execution time. The thesis of my dissertation is: when under time pressure, an automated agent should explicitly attempt to minimize the sum of planning and execution times, not just one or just the other.

My plan for tackling the problem of planning under time pressure has three stages. The first stage focuses on *parallel heuristic search*, where the parallel capabilities of modern computing hardware are used in order to decrease search time without increasing execution time. As of the writing of this abstract, the first stage is complete with respect to my dissertation. The second stage will focus on the setting of off-line heuristic search where the entire plan is synthesized before any actions are executed. I will be near the completion of my work on second stage by the time of the AAAI doctoral consortium. The third and final stage will focus on planning under time pressure when planning and execution of actions may happen concurrently. The remainder of this abstract discusses these three topics in more detail.

Parallel Search

One technique for decreasing the sum of planning and execution time is based on the simple idea that decreasing search time while holding execution time constant will decrease the sum of the two. To achieve this, we can exploit the parallel nature of modern computer hardware to decrease the search time by exploring portions of the search space concurrently. This is no easy task, in fact, we have shown that many naïve techniques for parallelizing search actually perform worse than serial search (Burns et al. 2010).

Along with a fellow graduate student and my advisor, I have created a parallel best-first search algorithm called PBNF (Burns et al. 2010). PBNF is able to achieve state-of-the-art performance by using information found in an abstract representation of the search space. At a high level, PBNF uses abstraction to find disjoint portions of the state space that can be searched in parallel with perfect duplicate detection without requiring any communication during expansion. The idea of using abstraction for breadth-first search was developed by Zhou and Hansen (2007), however, PBNF adapts this technique for best-first search. We prove that the basic PBNF algorithm is incomplete in infinite state spaces and present a simple modification to it that we prove makes it complete.

We have showed that PBNF can greatly decrease the search time required to find optimal solutions to a variety of planning problems. Additionally, PBNF has been adapted to act as a bounded sub-optimal search algorithm like weighted A* or as an anytime algorithm that returns a stream of solutions of decreasing cost. PBNF is an important contribution as it enables heuristic search to take advantage of modern processors that rely on parallelism instead of better clock-rates for increasing performance.

While parallelism helps to decrease the sum of planning and execution time, it is possible to attack the problem of planning under time pressure from a different angle. The next sections describe work that involves meta-reasoning–reasoning about search time in order to focus search effort toward inexpensive solutions that can be found more quickly.

Off-line Planning

In many settings, planning problems are solved off-line, i.e., an entire plan is synthesized before any execution begins. The goal of this stage is to develop an algorithm that attempts to minimize the sum of both planning and execution time. Such an algorithm will need to reason about the amount of planning time that it will require to find a certain solution. A good starting point for this work is the BUGSY algorithm, created by Ruml and Do (2007). BUGSY attempts to solve the planning under time pressure problem by finding a solution that minimizes a user-specified utility function, given as a linear combination of solution quality and planning time.

While BUGSY has a lot of promise, it is technically flawed because its method of estimating the search time required to find a solution is incorrect. When estimating this value, BUGSY assumes that it will only search nodes along the single path to that solution from then on. In practice, however, BUGSY will *vacillate* between many different solution, spending time on many different paths and not just the one path beneath a single node. BUGSY can grossly underestimate the amount of search time required to find a solution, so a solution may appear very desirable when it is in fact very costly.

Dionne, Thayer, and Ruml (2011) introduce a method called *expansion delay* for estimating search vacillation. Expansion delay estimates the number of expansions that an algorithm performs between the expansion of a node and the expansion of its successor, i.e. it estimates the number of expansions that will be performed along other paths between each single step along one path to the goal. In current work, I am adapting the BUGSY algorithm to use expansion delay in order to achieve more accurate estimates of the search time remaining to a goal, improving its ability to optimize the sum of search time and execution time.

Combined Planning and Execution

It is often possible and, in fact, desirable to allow for concurrent or interleaved planning and execution. Allowing execution before an entire plan is found is especially desirable in robotics or video games, in which an agent is expected to begin acting right when it is given a task, and likely before it has finished planning for achieving the task. The parallelism inherent in this setting may be exploitable by an algorithm that is attempting to reduce the sum of planning and execution times. For example, it may be beneficial to begin an action before thoroughly exploring plans that extend from it; while executing an action there can be an abundance of time for further planning before the next action must be returned. As a more concrete example, consider cooking: you may start boiling a pot of water before you have even decided what meal to make because most meals require boiling water and water requires a lot of time to boil.

The starting point for my research in planning under time pressure with concurrent planning and execution is the Decision Theoretic A* (DTA*, Russell and Wefald 1991) algorithm. DTA* handles concurrent planning and execution by deciding when to stop planning and to execute each action using a decision theoretic analysis the current planning state. DTA* suffers from a few problems. First, DTA* does not directly optimize the sum of planning and execution time. Second, it does not explicitly take advantage of concurrent execution (c.f., the boiling water example from above). Third, DTA* makes myopic decisions and, finally, it doesn't take into account uncertainty in its estimations. These issues all point to possible places where better techniques can be developed for planning while under time pressure.

Conclusions

Current heuristic search algorithms are ill-suited for planning under time pressure. Most classic heuristic search algorithms attempt to find short plans that will execute quickly. Finding such short plans is problematic as it often requires a prohibitive amount of time. Other search techniques disregard execution time and simply attempt to find any legal plan as quickly as possible. The resulting plans are usually of very low quality and take a long time to execute. When time is of the essence, it is undesirable to spend more time looking for a shorter plan than the amount of time saved by the decreased plan length. Likewise, it is undesirable to spend more time executing a longer plan than the amount of time saved by the decreased planning time. As such, new techniques are required to explicitly solve the problem of planning while under time pressure.

References

Burns, E.; Lemons, S.; Ruml, W.; and Zhou, R. 2010. Best-first heuristic search for multicore machines. *Journal of Artificial Intelligence Research* 39:689–743.

Dionne, A. J.; Thayer, J. T.; and Ruml, W. 2011. Deadline-aware search using on-line measures of behavior. In *Proceedings of the Fourth Symposium on Combinatorial Search (SoCS-11)*.

Doran, J. E., and Michie, D. 1966. Experiments with the graph traverser program. In *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 235–259.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2):100–107.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1:193–204.

Ruml, W., and Do, M. B. 2007. Best-first utility-guided search. In *Proceedings of IJCAI-07*, 2378–2384.

Russell, S., and Wefald, E. 1991. *Do the Right Thing: Studies in Limited Rationality*. MIT Press.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, 674–679.

Zhou, R., and Hansen, E. A. 2007. Parallel structured duplicate detection. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*.