Reasoning about when to start Acting

Richard Goodwin
rich@cs.cmu.edu
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890
(412) 268-8102

Abstract

Faced with a complicated task, some initial planning can significantly increase the likelihood of success and increase efficiency, but planning for too long before starting to act can reduce efficiency. This paper explores the question of when to begin acting for a resource bounded agent. Limitations of an idealized algorithm suggested in the literature are presented and illustrated in the context of a robot courier. A revised, idealized algorithm is given and justified. The revised idealized algorithm is used as a basis for developing a new "step choice" algorithm for making on-the-fly decisions for a simplified version of the robot courier task. A set of experiments are used to illustrate the relative advantage of the new strategy over always act, always compute and anytime algorithm based strategies for deciding when to begin execution.

Introduction

Faced with a complicated task, some initial planning can significantly increase the likelihood of success and increase efficiency. Conversely, sitting on one's hands, considering all possible consequences of all possible actions does not accomplish anything. At some point, action is called for, but when is this point reached?

The question of when to start executing the current best action is crucial for creating agents that perform tasks efficiently. The more time and resources spent creating and optimizing a plan, the longer the actions of the plan are delayed. The delay is justified if the improvement in the plan more than offsets the costs of delaying execution. This tradeoff is basic to creating resource bounded rational agents. An agent demonstrates resource bounded rationality if it performs optimally given its computational limits.

Deciding when to begin execution is complex. There is no way to know how much a plan will improve with a given amount of planning. The best that can be achieved is to have some type of statistical model as is done in the anytime planning framework (Dean & Boddy 1988). In addition, the decision should take into account the fact that it may be possible to execute one part of the plan while planning another.

Approaches

The classical AI planning approach to deciding when to begin execution is to run the planner until it produces a plan that satisfies its goals and then to execute the plan. This approach recognizes that finding the optimal plan may take too long and that using the first plan that satisfied all the constraints may produce the best results (Simon & Kadane 1974). It also relies on the heuristic that a planner will tend to produce simple plans first and simple plans tend to be more efficient and robust than unnecessarily complex plans.

In contrast to the classical AI approach, the reactive approach is to pre-compile all plans and to always perform the action suggested by the rules applicable to the current situation. The answer to the question of when to begin acting is always "now". This approach is applicable when it is possible to pre-compile plans and in dynamic environments where quick responses are required.

A third approach is to explicitly reason about when to start execution while planning is taking place. Using information about the task, the current state of the plan and expected performance of the planner, the decision to execute the current best action is made on-the-fly. This allows execution to begin before a complete plan is created and facilitates overlapping of planning and execution. This approach also allows execution to be delayed in favour of optimizing the plan, if the expected improvement so warrants.

Idealized Algorithm

Making on-the-fly decisions about when to begin execution can be modeled as a control problem where a meta-level controller allocates resources and time to the planner and decides when to send actions to the execution controller for execution. The control problem is to select actions and computations that maximize the expected utility. An idealized algorithm that has been suggested by a number of researchers, selects an item from the set $\{\alpha, C_1, \ldots, C_k\}$ which has highest expected utility(Russell & Wefald 1991). In this list, α is the current best action and the $C_i s$ represent possible computations. The utility of each computation must take into account the duration of the computation by which any act it recommended would be delayed. Of course,

$$E(dist(n, planTime_n)) = dist_{initial} - nA(1 - e^{-BplanTime_n}) \tag{1}$$

$$E(travelTime(dist_{a,b}, planTime_{a,b})) = \tag{2}$$
$$\begin{cases} (3.17dist_{a,b} - 1.50planTime_{a,b}/\tau_{pp})/v & planTime_{a,b} < 1.33\tau_{pp}dist_{a,b} \\ 1.17dist_{a,b}/v & planTime_{a,b} \geq 1.33\tau_{pp}dist_{a,b} \end{cases}$$

this ideal algorithm is intractable and any real implementation can only approximate it.

Revised Idealized Algorithm

The idealized algorithms ignores the fact that most agents can act and compute concurrently. A better control question to ask is whether the agent should compute or compute and act. The corresponding control problem is to select from the list $\{(\alpha, C_1), \ldots, (\alpha, C_k), (\phi, C_1), \ldots, (\phi, C_k)\}$ where each ordered pair represents an action to perform and a computation to do. ϕ represents the *null* action corresponding to computing only.

Considering pairs of actions and computations allows the controller to accept the current action and focus computation on planning future actions. This can be advantageous even if more computation is expected to improve the current action. For instance, I may be able to improve the path I'm taking to get to my next meeting that saves me more time than the path planning takes. However, I might be better off if I go with the current plan and think about what I'm going to say when I get to the meeting instead.

It is not the case that an agent should always consider both acting and computing. The original idealized algorithm is applicable for agents that play chess. In such a context it is not a good idea to sacrifice expected improvements in the current move in order to compute during moving. The utility of computing after a move and before the opponent has responded is relatively low. Considering both acting and computing only complicates the control mechanism and can reduce the overall performance of the agent. One property of chess that contributes to the low utility of computing while acting is that each action is irrevocable. There is in effect an infinite cost for undoing a move.

Using the revised idealized algorithm, I have been investigating the question of when an agent should begin to act. In the rest of this paper, I'll discuss the results in the context of a simple robot courier domain. I will first present a situation where there are improvements to be had by judiciously overlapping planning and execution. I then present a more formal analysis of a particular problem and use the analysis as a basis for generating some preliminary empirical results.

Robot Courier

In his thesis, Boddy introduces a robot courier task where a robot is required to visit a set of locations (Boddy 1991). The robot's environment is represented as a grid where each grid cell is either free or blocked. The robot is located in one cell and can attempt to move to one of the four adjacent cells. If the neighbouring cell is free, the move succeeds. Otherwise it fails and the robot remains in the same cell. Both successful and unsuccessful moves require time. The robot is provided with a map that can be used for doing route planning. The robot's objective is to visit the given locations, in any order, in the least amount of time possible.

The planning problem consists of two parts; ordering the visits and planning paths between locations. The purpose of planning is to reduce the amount of time needed to complete the set of visits. The initial ordering of locations could be used as the tour plan. Tour planning make the tour more efficient. Similarly, the agent has an execution controller that can dead-reckon between locations and will eventually make its way to any reachable location. Path planning can reduce the amount of time needed to travel between locations. The controller can also make use of partial plans to improve its efficiency.

For Boddy's robot, planning is done using a pair of anytime planning algorithms. Path planning is done by a heuristic search that yields increasingly complete paths between locations. Tour improvement is done using two-opt (Lin & Kernighan 1973). The expected improvement for each algorithm is a function of the amount of time spent planning. The performance of the two-opt algorithm can be characterized by a curve that exponentially approaches an asymptote (Equation 1). Path planning initially improves at a fixed rate until a cutoff point after which there is no further expected improvement.

Let's examine the first example presented in the thesis where the robot must go from its current location, l_0, and visit two locations, (l_1, l_2), in sequence. The distances from l_0 to l_1 and from l_1 to l_2 are 100 units. The problem is to create a deliberation schedule for allocating path planning time to each of the two paths.

In creating deliberation schedules, the assumption is made that the planner can only plan a path between locations before the robot begins to move between them. This assumption avoids the problem of how to do communication and co-ordination between the planner and the execution controller when both are working on getting the robot to the next location. It also reduces the complexity of the deliberation scheduler.

The deliberation plan given in the thesis entails the robot initially doing path planning for the first leg of the trip and then overlapping path planning for the second part of the trip with the execution of the first leg of the trip. The plan is parameterized by the time spent path planning each leg of the trip: $planTime_{1,2}$ and $planTime_{2,3}$. The plan is optimal when the time to follow the first path equals the expected time to plan the best path for the second part of the trip. The expected path length as a function of the distance between points and the amount of time spent planning are given in equation 2. Figure 1 gives the set of parameter values used in the thesis. The total expected time for visiting both locations, given an optimal amount of initial planning, is

τ_{pp}	$= 1$	Time to perform one planning step.
$dist_{1,2}$	$= 100$	Distance between first two locations.
$dist_{2,3}$	$= 100$	Distance between second two locations.
v	$= 1$	Speed of the robot.
$2/v$	$= 2$	The time cost for taking a step that fails.
$P(occupied)$	$= 0.25$	The probability that a given location is occupied.
$planTime_{a,b}$	$=$	Time spent planning the trip from a to b.
$planTime^*_{a,b}$	$= 1.17 dist_{a,b}/v$	Expected time to plan the optimal trip from a to b.

Figure 1: **Robot Courier Parameters**

$$\begin{aligned} \textit{Let} \quad E(TravelTime(dist_{1,2}, planTime_{1,2})) &= planTime^*_{2,3} = 133.0 && (3)\\ \textit{Then} \quad E(Time(plan1)) &= planTime_{1,2} + planTime^*_{2,3} && (4)\\ &\quad +E(TravelTime(dist_{2,3}, planTime_{2,3}) \\ &= 122.\dot{6} + 133.0 + 117.5 \\ &= 373.1\dot{6} \end{aligned}$$

$$\begin{aligned} E(Time(plan2)) &= P(occupied)(E(Time(plan1)) + 2/v) + && (5)\\ &\quad (1 - P(occupied))(planTime'_{1,2} + planTime^*_{2,3} + E(TravelTime(dist_{2,3}, planTime_{2,3}))) \\ &= 0.25 * (373.1\dot{6} + 2) + 0.75 * (120.553 + 133 + 117.5) \\ &= 372.08 \end{aligned}$$

given in equation 4[1].

Suppose we maintain the assumption that the path planner cannot be working on the path that the execution controller is currently executing. However, we remove the implicit assumption that the path planner can only be used to plan paths between the end-points of a path. Making use of the fact that an anytime algorithm can be interrupted at anytime and the results either used or thrown away, we can construct other possible deliberations schedules. Consider a plan where the robot attempts to take one step towards the first location and uses a deliberation schedule that assumes the robot started that one step closer. If the step succeeds, then the robot ends up one step closer to the location and the size of the path planning problem has been reduced. Of course, the step may fail. In which case, the planner could be interrupted and restarted using the original deliberation schedule. The cost of failure is the time it takes to discover that the step failed.

Equation 5 gives the expected time for the modified plan. For the parameters used, the expected time for this modified plan to visit both sites is about one time unit less than the original plan.

Why does the modified plan do better? One contributing factor is that the environment is relatively benign. Even if the robot tries a faulty step, the penalty for recovery is relatively low, 2 unit step times. Another important factor is that the rate of execution is the same order of magnitude as the rate at which the planner can improve the plan. If the robot was infinitely fast, then always acting would be the right thing to do. On the other hand, if the rate of computation was infinitely fast, doing full planning would always be the best thing to do. In between these two extremes, overlapping planning and execution offers potential efficiency improvements. However, blind overlapping of computation and execution, even in a benign environment, is not always the best thing to do. Suppose the processor on the robot was slightly faster. There is a critical point beyond which it makes sense to wait for the planner to finish rather than attempt the first step. In the example above, if the time for one planning step, Tpp, falls below 0.315 time units, then the original plan has the lower expected time to completion. Similarly, if the penalty for a bad step rises to $6.34/v$, the original plan is better.

Finally, note that for this example, the modified deliberation plan is not optimal. In most cases, there are two directions on the grid that reduce the distance to the goal. We could modify the the plan again so that if it fails on the first step, it tries the other direction that leads towards the goal. This would improve the expected times slightly. In the limit, a system where the planner and the execution controller were more tightly coupled would likely produce the best result for this example. See (Dean *et al.* 1991) for an example.

Tour Planning

How should an agent decide when to begin acting? For a more formal analysis, let's consider only the problem of ordering locations to visit for the robot courier. To simplify the analysis, consider the case where the distance between locations is proportional to the actual time to travel between them. Let's also assume that utility is linear with respect to the time needed to complete all the visits. In this situation,

[1]These numbers are slightly different from those in the thesis which were truncated.

$$
\begin{aligned}
E(\mathit{Improvement}(t,n)) = I(t,n) &= nA(1 - e^{-Bt}) &(6)\\
\mathit{Compute\ only\ rate} &= \dot{I}(t,n) = nABe^{-Bt} &(7)\\
\mathit{Compute\ and\ Act\ rate} &= \mathit{Rexe} + \frac{1}{\triangle t}\int_{t}^{t+\triangle t} \dot{I}(t,n-1)dt &(8)\\
&\approx \mathit{Rexe} + \frac{1}{2}(\dot{I}(t,n-1) + \dot{I}(t+\triangle t, n-1))\\
\triangle t &= -t - \ln(\frac{(n+1)ABe^{-Bt} - 2\mathit{Rexe}}{(n-1)AB})/B &(9)
\end{aligned}
$$

$$
\begin{aligned}
&\triangle t = \infty \quad \Rightarrow\\
0 &= (n+1)ABe^{-Bt} - 2\mathit{Rexe} = \frac{(n+1)}{n}\dot{I}(t,n) - 2\mathit{Rexe}\\
\Rightarrow \dot{I}(t,n) &= 2\mathit{Rexe}\frac{n+1}{n} \approx 2\mathit{Rexe} \qquad (10)
\end{aligned}
$$

the control choice for the robot is whether to execute the next step in its current plan while continuing to optimize the rest of the plan or to just do the optimization. The optimization that is available is to run the two-opt algorithm on the unexecuted portion of the plan.

The expected improvement in the tour as a function of computation time for the two-opt algorithm is described by equation 6. In the equation, t is the computation time, n is the number of locations and A and B are parameters that can be determined empirically.

One approach to making the meta level control algorithm efficient is to use a simple greedy algorithm as suggested by Russell and Wefald (Russell & Wefald 1991). This algorithm selects the option that gives the highest expected rate of task accomplishment. For the robot courier, there are two ways of making progress on the task: optimizing the tour and moving.

Anytime Approach

Given the characterization of the two-opt tour improvement algorithm given in equation 6, any anytime deliberation schedule can be created. At first, it might seam obvious that the robot should compute until the expected rate of tour improvement falls below the rate at which the robot can execute the plan. This is not optimal though. Such a deliberation schedule is based on the initial idealized algorithms and ignores the fact that the robot can both execute a step while optimizing the rest of the plan. Basing a deliberation schedule on the revised idealized algorithm, the robot should wait at most until the rate of improvement falls below twice the rate of execution before it begins to act. This allows the robot to better overlap planning and execution. A mathematical justification will be given in the next section.

Step Choice Algorithm

For on-the-fly decision making, an action should be taken whenever the expected rate of task accomplishment for both performing the action and optimizing the rest of the plan exceeds the expected rate of task accomplishment by only optimizing.

Computing with an anytime algorithm can be treated as a continuous process. However, moving between locations is more discrete. Each move will be treated as an atomic action that, once started, must be completed. The rate of tour improvement for computing alone is given in equation 7. When the agent chooses to act and compute, the agent is committing to the choice for the duration of the action. The appropriate rate to use when evaluating this option is not the instantaneous rate but the average rate over the duration of the action. Equation 8 gives the average expected rate of accomplishment for acting and computing. In this equation, $\triangle t$ is the duration of the action. Equating 7 and 8 and solving for $\triangle t$ gives an expression for the time duration of a move the agent would be willing to accept as a function of time spent computing. Unfortunately, the resulting expression has no closed form solution[2]. If instead, the integral in equation 8 is replaced by a linear approximation, an expression for $\triangle t$ can be found (Equation 9). The linear approximation over estimates the rate of computational improvement for the move and compute option and biases the agent towards moving. Equation 6 characterizes the expected performance of the two-opt algorithm on the original problem. Using it in equation 9 under estimates the effect of reducing the size of the optimization problem. Since two-opt is an $O(n^2)$ algorithm per exchange, reducing the problem size by 1 has more than a linear affect. This tends to cancel out the first approximation.

Examining the form of equation 9, it is clear that $\triangle t$ is infinite if the argument to the *ln*() function reaches zero. This corresponds to an expected rate of tour improvement about twice the rate of path execution (Equation 10). At this point, the marginal rate of only computing is zero and the robot should execute any size step remaining in the plan. For this reason, the anytime strategy should wait only until the rate of improvement falls below twice the rate of executions. In fact, the wait should only be long enough for $\triangle t$ to be larger than the expected time for the next step.

In the range where $\triangle t$ is defined, increasing the rate of execution (Rexe) increased the size of an acceptable step.

[2] The result involves solving the omega function

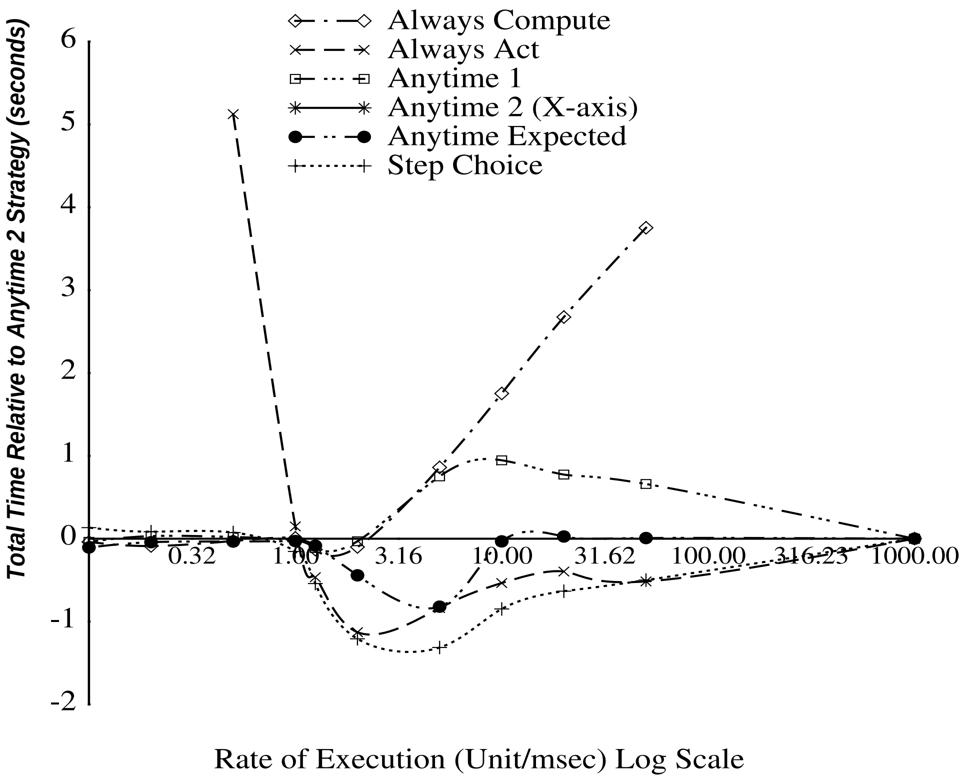

Figure 2: **Performance relative to Anytime 2 Control. Less Time is better.**

Similarly, increasing the rate of computation (B) decreases the size of the acceptable action. In the limit, the acceptable step correctly favours either always computing or always acting.

Does it make sense to perform an action even when the optimizer is currently improving the plan more than twice as fast as the robot can execute it? It does if the action is small enough. The amount of potential optimization lost by taking an action is limited by the size of the action. At best, the next step in the tour could be reduced to zero time (or cost). Performing small actions does not present a large opportunity cost in terms of potential optimizations lost and allows the optimizer to focus its efforts on a smaller problem. Taking actions that fall below the acceptable action size also allows the agent to be opportunistic. If at any point in the tour improvement process, the current next move is smaller that the acceptable action size, then the agent can begin moving immediately. In this way, the agent takes advantage of the optimizer's performance on the given problem rather than only on its expected performance.

Implementation

In order to empirically evaluate the utility of the control strategy given above, I have implemented a version of the simplified robot courier simulator and have performed an initial set of experiments. The six control strategies given in figure 3 were each run on a sample of 100 randomly generated problems each with 200 locations to be visited. The experiments were performed on a DECstation 3100 using the Mach 2.6 operating system. The code is written in C and all calculations, except calculating the acceptable step size, are done using integer arithmetic.

Always Compute : Complete the entire two-opt algorithm and then begin execution.

Always Act : Always perform the next action and run the two-opt algorithm in parallel.

Anytime1 : Run the two-opt algorithm and start execution when the expected rate of tour improvement falls below the rate at which the robot can execute the plan.

Anytime2 : Same as anytime1 except start execution when the expected rate of tour improvement falls below twice the rate at which the robot can execute the plan.

Anytime Expected : Same as the anytime1 strategy, except start execution when the acceptable step size equals the expected distance between cities.

Step Choice : Run the two-opt algorithm. Whenever the next move is smaller than the acceptable step size, take it.

Figure 3: **Tour improvement Meta-Level Control Algorithms.**

Since each exchange in the two-opt algorithm is $O(n^2)$ where n is the number of locations considered and since the size of the optimization problem shrinks as execution proceeds, each optimization step can't be treated as a constant time operation as was done by Boddy (Boddy 1991). For these experiments, I measured the CPU time used.

The graph in figure 2 shows the total time of each strategy relative to the anytime2 control strategy. Points below the x-axis indicate better performance. The graph covers a range of robot speeds where sometimes always acting dominates always computing and sometimes the converse it true.

The performance of the always act and the always compute strategies at either extreme are as expected. The anytime strategies and the step choice algorithm perform well for the entire range of speeds. At relatively fast robot speeds, these strategies correctly emulate the always act strategy. At relatively slow speeds, they correctly emulate the always compute strategy. Between these extremes, the anytime2 strategy outperforms the anytime1 strategy by starting execution earlier and better overlapping execution and optimization. For some ranges of execution speed, the always act strategy does better than either of these anytime strategies. These strategies don't take full advantage of the potential for overlapping planning with execution. The expected anytime strategy does better because it starts execution sooner.

The step choice algorithm produces the best results over the full range of execution speeds. It outperforms the anytime strategies by opportunistically taking advantage of small initial steps in the tour plan to begin execution earlier. It is interesting to note that even at relatively fast robot speeds, the behaviour of the always act and the step choice strategy are not identical. The step choice algorithm will occasionally pause if the next action to be taken is large. This allows the optimizer an opportunity to improve the next step before it is taken. The affect of this is to reduce the average length of the tour slightly while maintaining the same performance.

None of the algorithms made full use of the computational resources available. For the methods that overlap computation and acting, the computation completed before the robot reached the end of the path. The computational resource was then unused for the rest of the duration of run. This extra computation resource could have been used to apply more complex optimization routines to the remaining tour to further improve performance.

Future Work

The greedy meta-level control algorithm works for the simplified robot courier because the cost of undoing an action is the same as the cost of performing the action. In fact, because of the triangle inequality, it is rare that the action has to be fully undone. The shortest route proceeds directly to the next location. At the other end of the spectrum are domains such as chess playing where it is impossible to undo a move. In between, there are many domains with a distribution of recovery costs. Consider as an example leaving for the office in the morning. Let's assume that it is your first day at work, so there is no pre-compiled optimized plan yet. The first step in your plan is to walk out the door and then to the car. Suppose while you are walking, your plan optimizer discovers that it would be more optimal to bring your brief case. The cost of recovery is relatively cheap. Just walk back in the house and get it. On the other hand, if you forgot your keys, you may now be locked out of the house. In this sort of environment, a greedy control algorithm may not be the best choice to use when deciding when to begin acting.

Related Work

In resent work, Tom Dean, Leslie Pack Kaelbling and others have been modeling actions and domains using Markov models (Dean *et al.* 1991). Plans for these representations are policies that map states to actions. Planning consists of modifying the envelope of the current policy and optimizing the current policy. Deliberation scheduling is done while acting in an attempt to provide the highest utility for the agent. Work has also been done on deliberation scheduling when the agent has a fixed amount of time before it must begin acting. Such work has not directly addressed the question of when the agent should begin acting. It is assumed that the agent is given a arbitrary deadline for starting to execute. Techniques developed in this paper could be adapted to decide when to start execution, eliminating the need for an arbitrary start deadline.

Other related work includes Russell and Wefald's work on Decision Theoretic A* (Russell & Wefald 1991). This algorithm makes use of the initial idealized algorithm and is applicable to domains such as chess.

Acknowledgements

This research was supported in part by NASA under contract NAGW-1175. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of NASA or the U.S. government.

References

Boddy, M. 1991. Solving time-dependent problems: A decision-theoretic approach to planning in dynamic environments. Technical Report CS-91-06, Department of Computer Science, Brown University.

Dean, T., and Boddy, M. 1988. An analysis of time dependent planning. In *Proceedings AAAI-88*, 49–54. AAAI.

Dean, T.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1991. Planning with deadlines in stochastic domains. In *Proceedings, Ninth National Conference on Artificial Intelligence*. AAAI.

Lin, S., and Kernighan, B. 1973. An effective heuristic for the travelling salesman problem. *Operations Research* 21:498–516.

Russell, S., and Wefald, E. 1991. *Do the Right Thing*. MIT Press.

Simon, H., and Kadane, J. 1974. Optimal problem-solving search: All-or-nothing solutions. Technical Report CMU-CS-74-41, Carnegie Mellon University, Pittsburgh PA. USA.