HEURISTIC PREDICTIVE PATH SEARCH IN A PHYSICAL PUZZLE

Dana Cremer
Dana Vrajitoru
Indiana University South Bend,
Computer and Information Sciences
South Bend, IN, USA

ABSTRACT
This paper presents a few heuristic path search algorithms to solve a physical puzzle consisting of 3D maze and a marble, simulated in a physically accurate environment. An intelligent agent must move the marble to a target cell by rotating the maze itself. The physical nature of the puzzle provides an interesting challenge for the agent attempting to solve it, since it does not have complete control over the effects of its actions, and is not able to predict with certainty what those effects will be. The algorithms presented are based on building a physical state graph from past observations and using a predictive utility function to estimate the closeness to the target. The implemented algorithms incorporate varying levels of knowledge of the maze's geometry and of the physics involved.

KEYWORDS

search algorithm, physical simulation, heuristic

INTRODUCTION

The system presented in this paper is an intelligent agent that uses heuristic utility-based search algorithms to determine and execute the necessary moves required to solve a physical puzzle. The program is implemented using the Newton Game Dynamics engine and is rendered in real-time with the OpenGL API. The specific puzzle is a random 3D maze containing a marble that can be rolled using gravitation by rotating the maze. The goal is to roll the marble to a specified target location at the opposite corner. The main challenges of this problem consist in the fact that the result of an action cannot be known precisely before performing the simulation of that action, and that backtracking is in general not possible, since reversing a move might take the marble to a completely different location than the previous cell. As our application relies on a physics engine, higher dimension cannot be easily considered for the moment. Implementation details and a more in-depth description of all the methods can be found in (Cremer 2007).

This puzzle might seem simple, but considering all the physical phenomena that can affect the outcome of a move in reality however, the problem is more complex than it may appear first. The system responds to gravity, static and kinetic friction, angular momentum, and collisions. In fact, some of the moves encountered in this system are not repeatable in the simulation itself, which means that the agent can never be completely certain what the results of its actions will be.

One of the key elements of this study is the creation of an intelligent agent that can deal with events that are not directly initiated by the agent. These events, which are an unpredictable consequence of the agent's actions, can place the object in an unforeseen, potentially unrecoverable state. This added dependency on the physics simulation requires the agent to be able to learn the effects of other forces acting upon the object and be able to find a solution without complete control.

The paper differs considerably from other related path-finding algorithms present in the literature. In typical cases, like those presented in (Jones & Thuente 1990), it is assumed that the intelligent agent searching for a path to a desired location or state has complete control over the location to which the object is moved. The only exception seems to be the blind searches where it is possible to run into an obstruction that prevents a move (Koenig 2004). Even in such cases though, backtracking is usually possible, unlike in the present study.

(Duncan & Kumar 2006) presents a related problem, where a robot must explore a graph with a limited amount of fuel or tied by a rope of limited length. Their study is similar to ours in the sense that they also distinguish between the original graph and the graph known by the agent at any point in the algorithm, which is also constructed by exploration as in our case. However, their algorithm depends upon backtracking while that is not an option in our case. (Beckert 1999) presents a depth-first search without backtracking for automated reasoning.

Much research has been done in the area of graph theory to develop algorithms for determining optimal paths to follow on a graph or game tree to lead to a desired state. Well known heuristic searches (Russell & Norvig 1995) provide a basis for the algorithms presented in this paper. (Millington 2006) describes how graph search algorithms can be applied to path finding in games. (Cazenave 2006) presents optimizations to the A* and IDA* algorithms as applied to path-finding on maps. (Karpov et al. 2006) presents an artificial intelligence algorithm in the Unreal Tournament video game environment used for path-finding in a game map.

(Koenig 2004) describes how path-planning problems in computer games are different from traditional off-line search problems in other fields because autonomous agents in games will initially have incomplete knowledge of the terrain which results in a large number of contingencies that makes planning difficult.

In order to utilize incremental heuristic search algorithms, it is necessary to divide the search space into a finite number of discrete states, representing the marble's current location and the orientation of the maze. (M. Atkin 2000) suggests a method for using “critical points” for defining state boundaries to facilitate the use of state-based search algorithms in continuous dynamic search spaces. Their algorithm for implementing critical states is demonstrated in the 2D graphical game simulation “Capture the Flag”.

PROBLEM DESCRIPTION AND REPRESENTATION

In this section we introduce the problem and the main concepts used by the search algorithms.

2.1 The Maze

The mazes that the agent must solve are a three dimensional extension of the common two dimensional labyrinth. The maze is a cube composed of cells which are cubes instead of squares, and each cell has 6 walls corresponding to the six directions that one can move to from that cell. Figure 1 shows an example of such a maze. The marble is in the bottom corner at the start and the target is the top corner. The physical restrictions in the solvability of the maze are not considered by the construction algorithm. Thus, a theoretical path to the target exists in every maze, but we do not know if all the mazes are physically solvable as the only way to find out is to run our search algorithms.

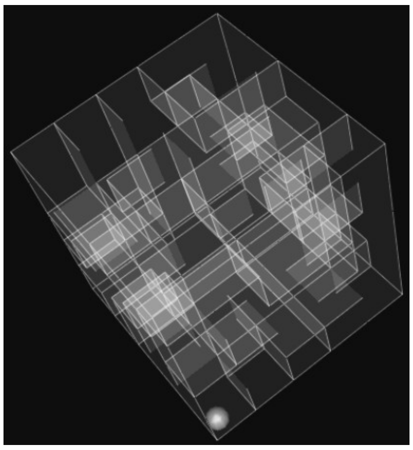
Figure 1. An example of a maze.

The methodology for generating the geometry of the mazes begins with generating all possible walls, then randomly selecting and removing walls until there is a single unique path connecting any two cells. This methodology of generating mazes is analogous to Kruskal's algorithm for generating minimum spanning trees (Weiss 2006). The paths between the cells of the maze can be viewed as edges between the nodes of a graph, and the goal is to make sure that there is a single path connecting any two nodes with no cycles. Unlike Kruskal's algorithm, we select the edge (the walls to remove) at random.

2.2 Theoretical Graph and Physical State Graph

The maze itself can be seen as a graph where each cell represents a node, and the direct connections between them are the edges. We will refer to this data structure as the theoretical graph (TG). A path from the starting cell to the target in this graph can easily be found with a classic search algorithm, but there is no guarantee that this path can be physically achieved through the rotations of the maze.

For the physical solution, we start by defining the states of the maze. A state is a combination of a maze cell and an orientation. Each rotation of the maze will be referred to as a move, and is implemented as a continuous rotation from one well-defined orientation of the maze to another. The states and the moves between them can then be represented as a graph or network called the physical state graph (PSG) and elements of graph theory can be applied when determining the physical solution.

Describing this system in terms of states and moves that cause the transition between them has some interesting analogies to traditional game theory. The agent attempting to select the optimal moves can be considered as one 'player', and the physical simulation as an opposing player making a move in response to the agent's. Some of the algorithms we implemented do in fact involve the agent attempting to anticipate the responding 'move' that will be made by the physics engine when choosing its next move to make, similar in concept to the popular Minimax algorithm (Russell & Norvig 1995).

Figure 2 shows an example of a 2D maze, the theoretical graph corresponding to it, and a partial physical state graph constructed by the agent. Since this is only an example, we have not included the orientation of the maze in the states. This figure illustrates the difference between the two graphs. For example, in the physical graph from the state 3, a clockwise rotation would bring the marble to state 9, rolling past state 6. The major difference is that unlike the theoretical graph, the physical state graph is directed, which makes backtracking not always possible.

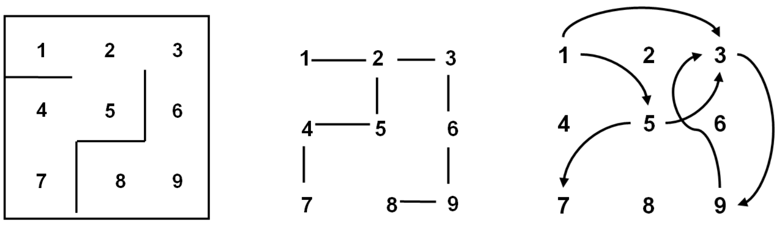

Figure 2. A 2D maze, the theoretical graph and a partial physical state graph.

3. PATH-FINDING ALGORITHMS

This section presents the heuristic path-finding algorithms that we propose to solve the physical maze problem.

3.1 Basic Solution

The algorithm starts with a PSG containing no edges. Each move performed by the agent takes the marble from one state to another, thus creating a new edge in the PSG. Figure 3 represents an example of a PSG after several moves have been made. The nodes represent states and the directed edges correspond to the moves (rotations) of the maze that were already made to transition from one state to the other. The nodes identified by a question mark indicate unknown states that can be reached by a single move from an already known state. Since these are moves that have not yet been made, it is unknown whether the nodes represent states that have been previously visited, states that have not yet been discovered, or even the target cell.

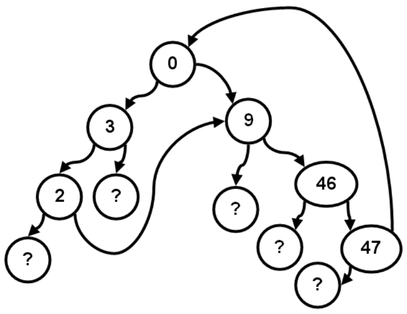

Figure 3. A partial physical state graph.

Implementing this graph to find the physical solution differs in several significant ways from most search algorithms typically utilized in artificial intelligence.

- Whenever the next move must be chosen, the agent does not have knowledge of the entire graph; only of the part that has been developed based on previous actions. It therefore cannot use a complete search algorithm to find a physical path leading to the target cell.
- Search algorithms typically involve selecting a node to expand, and then evaluating all the children of that node using various criteria for determining the order in which to evaluate the child nodes. For this system, evaluating a child node involves making a physical move to the new state. Since backtracking is not feasible, once a child is chosen, the option to evaluate its siblings no longer exists.
- There is nothing known about each unknown state, so we therefore have no immediate way of determining which may be the best branch to take.

Although physical backtracking is not considered possible, the system can search for the goal by choosing from the available branches off the current state. In general, a move to an unknown state will have one of two possible outcomes: it can lead to a previously undiscovered state, which will be added to the physical state graph, or it can lead to a state that has already been encountered, and is therefore already in the PSG. In the later case, the edge corresponding to the move that was made is connected to the node already representing that state. In this system, branches cannot lead back to the same state because the orientation of the maze changes after the move. Proceeding in this manner, the agent can 'explore' the maze, building a graph of the results as it proceeds.

When searching the PSG for an unknown state reachable from the current state, backtracking can be used. The reason is that this graph represents actions and resulting states that have already been discovered. We can safely assume that the same actions taken from the same states will again lead to the results indicated in the graph. Experimentally this is not always the case, but the repeatability is reliable enough that we can base decisions on it with reasonable confidence. To help clarify this, assume that the right branch from state 47 in Figure 3 is taken and found to lead back to state 0. If we now want to find a path back to state 46, we can use any traditional search algorithm with backtracking (e.g., depth-first, breadth-first), to find the path 0 -> 9 -> 46.

3.2 Blind Search

The first two algorithms developed to find a physical solution use no additional knowledge of the maze other than the PSG that has been developed based on moves already made. As the algorithms employ no knowledge of the domain when choosing the path to expand, they can be classified as 'blind' searches.

As previously stated, the PSG represents the known states and the moves that have been shown to lead between them. At the time a move needs to be chosen, the graph only reflects what has already been discovered, so the agent cannot find the complete path to the target cell. The best it can do with no further knowledge is to find a path from its current state to one of the unknown states in the PSG. By such experimentation, the PSG is expanded until a state corresponding to the target cell is found.

This exploratory methodology differs considerably from the way that search algorithms are generally employed. Whereas depth-first and breadth-first algorithms are generally used to find a desired state in known data, here they are being employed to intentionally find unknown states.

Such use of blind search algorithms to find the physical solution to the maze is not likely to be efficient, but could be considered 'complete' under three conditions. First, the algorithm must avoid loops as they can lead to infinitely long sequences of physical moves. Second, a solution must be possible given the control scheme (the physical moves that can be simulated), which may not always be the case. Third, the agent must not run into states that it is not capable of moving out of. As this situation happens sometimes, we report it as such in the results section. If these conditions are met, the physical solution to the maze will be found. The system decides that it cannot find a solution when it is no longer able to find any unknown state reachable from its current state, implying that it has already been everywhere it is capable of going.

We have tested both the breadth-first search (BFS) and the depth-first search (DFS) under these conditions. While they can both be used to find the solution, the depth-first is particularly inefficient, requiring about twice as many moves as the breadth-first. More detailed results will be presented in the next section.

3.3 Heuristic Search

In order to find a more efficient physical solution to the maze, knowledge of the maze itself can be utilized to evaluate the possible unknown states that can be investigated. The next algorithms we present are heuristic searches based on the Best-First Search (Russell & Norvig 1995), with the major difference that due to the physical constraints only one branch of the tree can be expanded, i.e. physically simulated in each step. All of them use an estimation of how likely it is that a given state will lead to the solution faster.

The algorithms search the PSG to find all the unknown states that are reachable from the current state, and are within one move from a known state. A heuristic value is evaluated either for the parent state or for the unknown state, depending on the definition of the function. The unknown state of the best heuristic value is chosen and a move is made from the parent of this state. After simulating the move in the physical maze, the unknown state becomes known and it is added to the PSG. Note that the simulation involves moving the marble from the current state to the parent of the unknown state, and only then executing the new move. The algorithms can be summarized as shown bellow, where H is the heuristic function.

Heuristic based search algorithm

```
current_state = origin
PSG = {origin, unknown children}
while current_state != target:
  best_ukn_state = nil,   best_h_value = infinity
  for each unknown state U in PSG:
    P = parent of U
    find path from current_state to P in PSG by BFS
    h = H(P or U)
    if h < best_h_value:
      best_ukn_state = U,   best_h_value = h
      parent = P
new_node = move from parent
current_state = new_node
add new_node to PSG
add children of new_node as unknown in PSG
```

Simple heuristics such as the Manhattan distance have been shown to be very misleading in mazes, and therefore not of much value (Koenig 2004). Our first heuristic algorithm uses the length of the theoretical path from the marble's location to the target cell to decide which unknown state to explore next. Equation 1 expresses this function.

$$H_1(state) = length(path(state; target) \in TG) \quad (1)$$

This heuristic is reasonable since it takes into account the actual physical nature of the maze. The method by which the maze is generated (as a spanning tree) guarantees that there is a unique theoretical path from each cell to the target cell. Since that theoretical path can be quickly determined for each unknown state with a DFS, evaluating this function for unknown states is very fast compared to the computational expense of the physical simulation.

We cannot evaluate H_1 for an unknown state since nothing is known about them. We can however use the hypothesis that there is a reasonable correlation between the desirability of this unknown state and the H_1 value of its parent state. In other words, the agent will act on the assumption that if a state is close to the goal, a single move from that state is likely to get the marble close to the goal. This heuristic does not distinguish between unknown children of the same known parent, so an arbitrary decision is made between them. The next algorithm proposes a more intelligent way to chose between the unknown child states.

3.4 Single Move Prediction

The next heuristic is designed to improve the efficiency of the search by anticipating the physical result of a move. Instead of evaluating the parent of the unknown state, the algorithm predicts the path on which the marble will travel, and uses the theoretical distance from the predicted end cell to the target to judge the value of the move. This prediction will be made for every reachable unknown state in the physical state graph, and the move believed to be the best will be pursued. As with the previous method, BFS is employed to find the shortest path (the fewest physical moves) to reach each unknown state.

To make the prediction of the path followed by the marble from a given state in response to a specific move (rotation to another orientation), a recursive algorithm was developed. The first adjacent cell that the marble will move to in response to a given move can be determined by following the direction of movement. From this cell one of two events can occur. The marble can continue moving in the direction of its current velocity, or the velocity-dominate direction, or it can fall in one of two gravity-dominate directions. The prediction of which direction is taken is based on the geometry of the maze (presence or absence of walls) and the marble's estimated speed.

After determining which cell the marble is likely to move to next, we adjust the speed and direction of the marble and apply the procedure recursively until we reach a cell where no further progress is possible. This represents the predicted final location for the marble in response to the move. Equation 2 expresses this second heuristic, where Predict denotes the procedure we described.

$$H_2(state) = 1 + H_1(state_0) \qquad \text{where} \quad state_0 = Predict(state) \tag{2}$$

The prediction function is designed to give a very fast response as compared to the simulation, and thus it embeds a simplified model of the physics involved. This extreme simplification of reality is in fact shown to be over 97% accurate in the test cases we used in our experiments.

While using this prediction algorithm, the PSG is updated with the actual results of each move made, as determined by the simulation. Before predicting the results of a move, the algorithm first consults the PSG to see if the move has already been made, and uses the previous results if available. This gives the agent the ability to both learn from experience and correct prediction errors.

3.5 Complete Solution Prediction

The algorithm described in this section attempts to improve the performance of the agent solving this puzzle by predicting a complete physical solution path to the target cell. The 'Complete Solution Prediction' algorithm is based on repeated use of the 'Single Move Prediction' algorithm covered in the previous subsection. The method is applied to predict the resultant state of each potential move. Each prediction is assumed to be correct, and a new prediction can be made for each potential move from the new state. Since all possible moves can be analyzed, and backtracking can be employed (since no actual physical moves are being performed), a traditional search algorithm can be employed to find a sequence of moves that should lead all the way to the target cell. Thus we can express the new heuristic recursively as shown in Equation 3.

$$\begin{aligned} &H_3(state) = 0 && \text{if } state = target \\ &H_3(state) = 1 + \mathrm{H}_3(\mathrm{state}_0) && \text{otherwise,} \\ &\text{where } state_0 = Predict(state). \end{aligned} \tag{3}$$

This method works by building and searching a different graph, the Predicted State Graph, for a state corresponding to the target cell. The theory behind this method is nearly identical to the Blind Search Algorithms employed on the PSG. The predicted state graph has the same form and properties as the PSG except that the graph is expanded with the predicted results for potential moves, rather than the results of simulating the physical moves. This algorithm can therefore build and search the entire graph, employing backtracking as required, until an entire path to the goal is found. The Complete Solution Prediction algorithm is fast enough that it can be applied numerous times to develop a complete solution before a single move is decided upon.

We used BFS to search the predicted state graph for a solution, as it will find a path with the fewest possible number of physical moves. This is important, not only because it will improve the performance of our agent, but because it will improve reliability as well. Since each move is based on predicted results subject to inaccuracies, minimizing the number of moves required will reduce the chances of the marble deviating from the predicted path.

This algorithm actually attempts to find an optimal physical solution. If the results of every possible move are predicted accurately, and the path with the fewest number of moves is chosen, this should result in the optimal path being followed. It turns out that complete optimality is not possible because of prediction inaccuracies. However, for mazes in which the selected predicted path was followed without errors, a dramatic improvement in performance was made.

Unlike the previous algorithms developed, which are based on finding and exploring states that were previously unknown in the PSG, this algorithm is based on predicting and following a complete path from the marble's current location to the target cell. This fundamental difference has two significant implications that must be dealt with in the implementation. First, the algorithm may not be able to find a complete solution to a target state. If this situation occurs, the Single Move Prediction algorithm is employed instead. Second, due to the inaccuracies of the prediction, the result of the move may not be the predicted state. The solution to this is for the agent to search for a new complete path to the solution whenever it deviates from the predicted path.

3.6 Improved Control Scheme

The algorithms presented so far have been using a control scheme with 8 possible alignments of the cubic maze, each with one vertex of the maze pointing straight up. While there are some advantages to it, it can also occur that the marble rolls balanced on the edge between the two walls and comes to rest in an undefined state (a frozen state) (Cremer 2007). Due to round-off errors and other effects, this selection of alignments also results in a significant chance for unrepeatable moves which decreases the reliability of the agent.

To deal with these issues, a more complex set of possible alignments of the maze was developed. The alignments were carefully chosen such that the marble would be more likely to travel in one direction than in the others.

The strategy for choosing the possible orientations was to deviate the top corners slightly away along each axis. This results in 24 possible orientations of the maze. This scheme gives the agent a more control and thus can improve the performance in terms of number of solved mazes and reliability. A drawback is an increased number of states that can lead to more moves that are necessary to solve the problem or to determine that a maze is unsolvable.

4. EXPERIMENTAL RESULTS

Table 1 shows the results of testing the methods on 100 mazes measuring 5 cells along each edge. As one would expect, the blind search methods performed poorly. DFS took over 24 hours to run on all 100 mazes, and could solve only 20% of them (100 total mazes minus 4 unsolvable and 76 frozen). The Single-Move prediction algorithm (H_2) was able to solve mazes with the fewest moves, but the Complete Solution Prediction algorithm using 24 orientations (H_3') was the most reliable, solving 89% of the mazes tested. Complete Path Prediction (H_3) did not show improvement over the Single Move Prediction (H_2) in terms of number of moves. This is primarily due to the fact that by increasing the number of predicted moves, the prediction errors also increased.

Comparing the Complete Solution Prediction with 24 Orientations (H_3') to the first (DFS), we see that the additional intelligence incorporated into the agent was able to improve its performance dramatically. The average number of moves required to solve a maze was reduced by over 62%. The reliability, as measured by the percentage of mazes that could be physically solved, was increased from 20% to 89%.

Table 1: Results for 100 5x5x5 mazes; Average length of the theoretical path from starting cell to target = 18.4

	Moves	Solved	Unsolvable	Frozen
DFS	181.1	20	4	76
BFS	114.6	34	15	51
H1	74.3	61	9	30
H2	18.6	62	11	27
H3	21	68	7	25
H3'	57	89	4	7

CONCLUSION

This paper has presented several algorithms that were developed for finding the physical solution to a random three dimensional maze. These algorithms all use elements of graph theory to expand the agent's knowledge of the maze being solved and the results of specific actions. The algorithms differ in the level of additional knowledge utilized pertaining to the maze's geometry and the physics involved. In general, it was shown that increasing the level of knowledge and reasoning capacity of the agent significantly improves its performance. A modified control scheme was also developed which improves the physical actions that can be made by the agent. This was shown to reduce problems which occurred as part of the physical simulation.

In terms of efficiency, all of the solution algorithms are complete in the sense that they are able to solve any random maze, but only under the conditions that a physical solution is possible, the control mechanism is capable of performing the necessary actions, and the marble does not enter a section of the maze from which it cannot escape. None of the algorithms developed can be guaranteed to find the optimal physical solution, since this would require perfect knowledge of the results of a given move before it is made. However, despite the limitations imposed by the physical simulation, the best algorithm developed was able to improve the performance (as measured by the number of mazes that could be solved) over a simple blind search by over 400%, while requiring 69% fewer moves.

In this project, artificial intelligence, physics simulation, and computer graphics are successfully integrated to develop and display the solution to an interesting problem that could not be solved without each of the three technologies.

REFERENCES

Beckert, B. 1999. Depth-first proof search without backtracking for free variable semantic tableaux. *In Position Papers, International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pp. 33-54.

Cazenave, T. 2006. Optimizations of data structures, heuristics, and algorithms for path-finding on maps. *In IEEE Symposium on Computational Intelligence and Games*, pp. 27-33.

Cremer, D. 2007. *The Application of Artificial Intelligence to Solve a Physical Puzzle*. Master Thesis, Indiana University South Bend, Department of Computer and Information Sciences. http://www.cs.iusb.edu/thesis/DCremer_thesis.pdf

Duncan, C. A., and Kumar, A. S. A. 2006. Optimal constrained graph exploration. *ACM Transactions on Algorithms* Vol 2, Nr. 3, pp. 380-402.

Jones, R., and Thuente, D. J. 1990. The role of simulation in developing game playing strategies. In *Proceedings of the 23rd Annual Symposium on Simulation*, pp. 89-97. Piscataway, NJ, USA: IEEE Press.

Karpov, I.; D'Silva, T.; Varrichio, C.; Stanley, K.; and Miikkulainen, R. 2006. Integration and evaluation of exploration-based learning in games. *In IEEE Symposium on Computational Intelligence and Games*, pp. 39-44.

Koenig, S. 2004. A comparison of fast search methods for real-time situated agents. *In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 864-871. IEEE Computer Society.

M. Atkin, P. C. 2000. Using simulation and critical points to define states in continuous search spaces. *In Proceedings of the 32nd Conference on Winter Simulation*, pp. 464-470. Society for Computer Simulation International.

Millington, I. 2006. *Artificial Intelligence for Games*. Morgan Kaufmann.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice-Hall.

Weiss, M. A. 2006. *Data Structures and Algorithm Analysis in C++*. Addison Wesley, third edition.