Search Conclusion and CSP Introduction

CPSC 322 Lecture 9

January 23, 2006
Textbook §3.0 – 3.2

Lecture Overview

Branch-and-Bound Search Algorithm

- Follow exactly the same search path as depth-first search
  - treat the frontier as a stack: expand the most-recently added node first
  - the order in which neighbors are expanded can be governed by some arbitrary node-ordering heuristic
- Keep track of a lower bound and upper bound on solution cost at each node
  - lower bound: $LB(n) = cost(n) + h(n)$
  - upper bound: $UB = cost(n')$, where n' is the best solution found so far.
    - if no solution has been found yet, set the upper bound to ∞.
- When a node n is selected for expansion:
  - if $LB(n) \geq UB$, remove n from frontier without expanding it
    - this is called "pruning the search tree" (really!)
  - else expand n, adding all of its neighbours to the frontier

Summary of Search Strategies

Strategy	Frontier Selection	Complete?	Space
Depth-first	Last node added	No	Linear
Breadth-first	First node added	Yes	Exp
Lowest-cost-first	Minimal $cost(n)$	Yes	Exp
Best-first	Global min $h(n)$	No	Exp
A^*	Minimal $f(n)$	Yes	Exp
Branch-and-Bound	Last node added, with pruning	No	Linear

Other A^* Enhancements

The main problem with A^* is that it uses exponential space.
Branch and bound was one way around this problem.
Two others are:

- Iterative deepening
- Memory-bounded $A*$

Non-heuristic pruning

What can we prune besides nodes that are ruled out by our heuristic?

- Cycles
  - this one is really easy
- Multiple paths to the same node
  - if we want to maintain optimality, either keep the shortest path, or ensure that we always find the shortest path first

Direction of Search

- The definition of searching is symmetric: find path from start nodes to goal node or from goal node to start nodes.
  - Of course, this presumes an explicit goal node, not a goal test.
  - Also, when the graph is dynamically constructed, it can sometimes be impossible to construct the backwards graph
- Forward branching factor: number of arcs out of a node.
- Backward branching factor: number of arcs into a node.
- Search complexity is b^n. Should use forward search if forward branching factor is less than backward branching factor, and vice versa.

Bidirectional Search

- You can search backward from the goal and forward from the start simultaneously.
- This wins as $2b^{k/2} \ll b^k$. This can result in an exponential saving in time and space.
  - The main problem is making sure the frontiers meet.
  - This is often used with one breadth-first method that builds a set of locations that can lead to the goal. In the other direction another method can be used to find a path to these interesting locations.

Dynamic Programming

Idea: for statically stored graphs, build a table of $dist(n)$ the actual distance of the shortest path from node n to a goal.
This can be built backwards from the goal:

$$dist(n) = \begin{cases} 0 & \text{if } is\_goal(n), \\ \min_{\langle n,m \rangle \in A}(|\langle n, m \rangle| + dist(m)) & \text{otherwise.} \end{cases}$$

This can be used locally to determine what to do.
There are two main problems:

- You need enough space to store the graph.
- The *dist* function needs to be recomputed for each goal.

Complexity: polynomial in the size of the graph.

States and Features

- In practical problems, there are usually too many states to reason about explicitly
- However, the states usually have some internal structure
  - this is why people can understand the problem in the first place!
- Features: a set of variables that together define the state of the world

- Many states can be described using few features:
  - 10 binary features $\Rightarrow$ 1,024 states
  - 20 binary features $\Rightarrow$ 1,048,576 states
  - 30 binary features $\Rightarrow$ 1,073,741,824 states
  - 100 binary features $\Rightarrow$ 1,267,650,600,228,229,401,496,703,205,376 states

Variables

- So, we define the state of the world as an assignment of values to a set of variables
  - variable: a synonym for feature
  - we denote variables using capital letters
  - each variable V has a domain $dom(V)$ of possible values
- Variables can be of several main kinds:
  - Boolean: $|dom(V)| = 2$
  - Finite: the domain contains a finite number of values
  - Infinite but Discrete: the domain is countably infinite
  - Continuous: e.g., real numbers between 0 and 1
- We'll call the set of states that are induced by a set of variables the set of possible worlds

Syntax and Semantics

- Syntax: the symbols that are manipulated by the computer, and the rules that are used to perform the manipulation
- Semantics: the meaning assigned to the symbols by the system designer
  - for example, the variable *black_queen_location* might correspond to the location on the chessboard of the black queen
- Important point: the computer only works at the syntactic level
  - it doesn't understand what the symbols mean!
  - things that seem obvious to us must be made explicit

Examples

- Crossword Puzzle:
  - variables are words that have to be filled in
  - domains are English words of the correct length
  - possible worlds: all ways of assigning words
- Crossword 2:
  - variables are cells (individual squares)
  - domains are letters of the alphabet
  - possible worlds: all ways of assigning letters to cells
- Sudoku
  - variables are cells
  - domains are numbers between 1 and 9
  - possible worlds: all ways of assigning numbers to cells

More Examples

- Scheduling Problem:
  - variables are different tasks that need to be scheduled (e.g., course in a university; job in a machine shop)
  - domains are the different combinations of times and locations for each task (e.g., time/room for course; time/machine for job)
  - possible worlds: time/location assignments for each task
- *n*-Queens problem
  - variable: location of a queen on a chess board
    - there are *n* of them in total, hence the name
  - domains: grid coordinates
  - possible worlds: locations of all queens

Constraints

Constraints are restrictions on the values that one or more variables can take

- Unary constraint: restriction involving a single variable
  - of course, we could also achieve the same thing by using a smaller domain in the first place
- k-ary constraint: restriction involving the domains of k different variables
  - it turns out that k-ary constraints can always be represented as binary constraints, so we'll often talk about this case
- Constraints can be specified by
  - giving a list of valid domain values for each variable participating in the constraint
  - giving a function that returns true when given values for each variable which satisfy the constraint
- A possible world satisfies a set of constraints if the set of variables involved in each constraint take values that are consistent with that constraint

Examples

- Crossword Puzzle:
  - variables are words that have to be filled in
  - domains are valid English words
  - constraints: words have the same letters at points where they intersect
- Crossword 2:
  - variables are cells (individual squares)
  - domains are letters of the alphabet
  - constraints: sequences of letters form valid English words
- Sudoku
  - variables are cells
  - domains are numbers between 1 and 9
  - constraints: rows, columns, boxes contain all different numbers

More Examples

- Scheduling Problem:
  - variables are different tasks that need to be scheduled (e.g., course in a university; job in a machine shop)
  - domains are the different combinations of times and locations for each task (e.g., time/room for course; time/machine for job)
  - constraints: tasks can't be scheduled in the same location at the same time; certain tasks can't be scheduled in different locations at the same time; some tasks must come earlier than others; etc.
- n-Queens problem
  - variable: location of a queen on a chess board
  - domains: grid coordinates
  - constraints: no queen can attack another

Constraint Satisfaction Problems: Definition

A constraint satisfaction problem consists of:

- a set of variables
- a domain for each variable
- a set of constraints

Model: an assignment of values to variables that satisfies all of the constraints

Constraint Satisfaction Problems: Variants

We may want to solve the following problems with a CSP:

- determine whether or not a model exists
- find a model
- find all of the models
- count the number of models
- find the best model, given some measure of model quality
  - this is now an optimization problem
- determine whether some property of the variables holds in all models

CSPs: Game Plan

It turns out that the general CSP problem with finite domains is $\mathcal{NP}$-hard, so we can't hope to find an efficient algorithm.

However, we can try to:

- find algorithms that are fast on "typical" cases
- identify special cases for which algorithms are efficient
- find approximation algorithms that can find good solutions quickly, even they may offer no theoretical guarantees
- develop parallel or distributed algorithms so that additional hardware can be used