Lecture 7

- Review
- Blind search
- Chess & search

Depth First Search

- Key idea: pursue a sequence of successive states *as long as possible.*

```
unmark all vertices
    choose some starting vertex x
    mark x
    list L = x
    tree T = x
    while L nonempty
        choose the vertex v from front of list
        visit v
        for each unmarked neighbor w
            mark w
            add it to front of list
            add edge (v,w) to T
```

BFS

- Key: explore nodes at the same distance from the start at the same time

```
unmark all vertices
    choose some starting vertex x
    mark x
    list L = x
    tree T = x
    while L nonempty
        choose the vertex v from front of list
        visit v
        for each unmarked neighbor w
            mark w
            add it to end of list
            add edge (v,w) to T
```

Key issues in search

- Here's what to keep in mind.
- Completeness: are we assured to find a solution (if one exists)?
- Space complexity: how much storage do we need?
- Time complexity: how many operations do we need?
- Solution quality: how good is the solution

- Also…
  - Can we expect an interim solution?
  - Can we refine a partial/inadequate solution?
  - Can we cope with imperfect knowledge?

Search performance

- Key issues that determine the nature of the problem are:
  - **Branching factor** of the search space: how many options do we have at any time?
    - Typically summarized by the worst-case braching factor, which may be quite pessimistic.
  - **Solution depth**: how long is the path to the first solution?

Example: knight's tour

- Tour executed by a chess knight to cover (touch) every square on a chess board.
  - Sub-problem: Find a path from one position to another.
- States: possible positions on the chess board.
- Operators: the ways a knight moves.
- Goal: the positions of the pawns.
- Path cost: the number of moves used to get to a position.

Knight's

- How large is the state space?
  - A knight has up to 8 moves per turn.
  - Each possible tour must be verified up to the end of the trip. For a board of width N, there are N*N squares.
    - Thus, each tour can be up to N*N states in length.
  - If the correct solutions is found last, we might consider up every wrong tour first.

  $O(8^{N\wedge 2})$ states to examine!

What does this say about BFS? DFS?

Example: Knight's heuristics

- **Zero**: a board can be considered a dead end if any square has zero paths remaining to it. Any square with no paths to it would be unreachable, so no Knight's Tour would exist.
- **Two ones**: a board with more than one square with only one path to it is a dead end. A square with one path left is necessarily a dead end, so two of them indicate a dead end position.
- **Move to one**: the move finder should never choose a move to a square with one path left unless it is the last move; such a choice would otherwise lead to a dead end.

- How well will we do if we use “blind” DFS?
- That is, if we consider random tours?

Applet Viewer: Hw5_23c.class

Applet

Full tour produced at 3724122

The Distribution of Tour Length:

1	2	3	4	5	6	7	8
0	0	0	2184	2048	6187	5890	9887
9	10	11	12	13	14	15	16
10099	14776	14734	20561	19899	27129	26337	34769
17	18	19	20	21	22	23	24
34021	44222	42591	54671	52573	65745	63380	78348
25	26	27	28	29	30	31	32
74752	91527	87630	105141	100167	119356	112681	132464
33	34	35	36	37	38	39	40
124064	143282	132906	150826	137549	153940	138422	151938
41	42	43	44	45	46	47	48
133214	142374	121666	126676	104952	104410	83146	79196
49	50	51	52	53	54	55	56
59879	53945	37838	31949	20887	15887	9171	6208
57	58	59	60	61	62	63	64
3063	1784	723	322	96	36	2	1

Total elapsed time: 4.8029885 hour

- Three heuristics based on the number of paths remaining to each square were implemented and tested in combination, as well as a representational speedup. The optimal combination of the heuristics is to eliminate boards with either a square with zero paths remaining or a square with two ones remaining; this combination led to a 950-fold speedup on a 6x6 board. The representational speedup led to a 2.5-fold additional speedup on a 6x6 board.
  - Michael Bernstein

Heuristics: Knight's Performance

ZERO	TWO ONES	MOVE TO ONE	Time
0	0	0	9500 ms
1	0	0	415 ms
0	1	0	13.7 ms
0	0	1	6430 ms
0	1	1	13.2 ms
1	1	0	10.0 ms
1	0	1	350 ms
1	1	1	10.4 ms

BFS

- Consider a state space with a uniform branching factor of *b*
- At each level we have b nodes for every node we had before

  $1 + b + b^2 + b^3 + b^4 \ldots + b^d$

So, solution at depth d implies we must expand

$O(b^d)$ nodes!

- internal nodes (b**d-1)/(b-1)
- Leaf nodes (on **average**): (b**d+1)/2

BFS: solution quality

- How good is the solution quality from BFS?

- Remember that edges in the search tree can have **weights**.

- BFS will always find the shallowest (shortest depth) solution first.
- This will also be the minimum-cost solution if…
  the cost is a non-decreasing function of the depth.
- E.g. if the cost *g(n)* is a linear function of the depth.

BFS & Memory

- For BFS, we must record all the nodes at the current level.
  - BFS can have large (enormous) memory requirements!
- Memory can be a worst constraint than time.
- When the problem is exponential, however, we're in trouble either way.

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Comparison to DFS?

- Worst case:
  - If the search tree can be arbitrarily deep, DFS may never terminate!
  - If the search tree has maximum-depth *m*, the DFS (at worst) visits every node up to depth *m*.
  - Time complexity $O(b^m)$
- If there are lots of solutions (or you're lucky), DFS may do better then BFS.
  - If it gets a solution on the first try, it only looks at *d* nodes.
- Very good in space usage.

Exactly how many nodes? See DAA p. 139.

Uniform Cost Search

- Note the edges in the search tree may have costs.
- All nodes as a given depth may not have the same cost, or desirability.
- **Uniform cost search**: expand nodes in order of **increasing cost from the start**.

- This is assured to find the cheapest path first, so long as ….

  there are no negative costs.

Iterative-Deepening

- Combine benefits of DFS (less memory) and BFS (best solution depth/time).
    - Repeatedly apply DFS up to a maximum depth "**diameter**".
    - Incrementally increase the diameter.

Unlike BFS we do not store the leaves

Iterative-Deepening Performance

Idea: expand the same nodes as BFS, *and in the same order as BFS!*

Don't save intermediate results, so nodes must be re-expanded.

How can this be good?!

This is a classic time-space tradeoff.

Because the search tree is exponential, wasted work near the top doesn't matter much.

Asymptotically optimal, complete.

Time $O(b^d)$ like BFS Space $O(bd)$ like DFS

Not *always* preferred (if space small or depth known, or other knowledge available).

SEMINAR

Deep Blue: IBM's Massively Parallel Chess Machine

Wednesday, September 23, 1998

TIME: 11:00 A.M.

Strathcona Anatomy and Dentistry Building

3640 University Street

Room M-1

Dr. Gabriel M. Silberman

Program Director

Centre for Advanced Studies (CAS), IBM Toronto

CS-424 Gregory Dudek

Seminar abstract

IBM's premiere chess system, based on an IBM RS/6000 SP scalable parallel processor, made history by defeating world chess champion. Garry Kasparov. Deep Blue's chess prowess stems from its capacity to examine over 200 million board positions per second, utilizing the computing resources of a 32-node IBM RS/6000-SP, populated with 512 special purpose chess accelerators.

In this talk we describe some of the technology behind Deep Blue, how chess knowledge was incorporated into its software, as well as the attitude of the media and general public during the match.

- GABBY SILBERMAN'S BIO
- Gabriel M. Silberman is Program Director for the Centre for Advanced Studies (CAS) at the IBM Toronto Laboratory. Dr. Silberman comes to CAS from the IBM T.J. Watson Research Center, Yorktown Heights, NY, where he held various research positions from 1990 to 1997.

Computer Chess & DB

Not in text

- Background for tomorrow's seminar
  - Presentation on Deep Blue and Chess
    Courtesy Kautz&Selman (not available here)

Combinatorics of Chess

Opening book

Endgame

- **database of all 5 piece endgames exists; database of all 6 piece games being built**

Middle game

- **branching factor of 30 to 40**
- **$1000^{(d/2)}$ positions**
  - **1 move by each player = 1,000**
  - **2 moves by each player = 1,000,000**
  - **3 moves by each player = 1,000,000,000**

Positions with Alpha-Beta Pruning

Search Depth		Positions
2		60
4		2,000
6		60,000
8		2,000,000
10	(<1 second DB)	60,000,000
12		2,000,000,000
14	(5 minutes DB)	60,000,000,000
16		2,000,000,000,000

Formal Complexity of Chess

How hard is chess?

- Obvious problem: standard complexity theory tells us nothing about finite games!
- Generalizing chess to NxN board: optimal play is PSPACE-hard
- What is the smallest Boolean circuit that plays optimally on a standard 8x8 board?

Fisher: the smallest circuit for a particular 128 bit function would require more gates than there are atoms in the universe

History of Search Innovations

Shannon, Turing	Minimax search	1950
Kotok/McCarthy	Alpha-beta pruning	1966
MacHack	Transposition tables	1967
Chess 3.0+	Iterative-deepening	1975
Belle	Special hardware	1978
Cray Blitz	Parallel search	1983
Hitech	Parallel evaluation	1985
Deep Blue	ALL OF THE ABOVE	1987

Bidirectional search

Not in text

- Remember that the bottom of the search tree is where most of the work is.
- Idea:
  - keep the tree smaller
  - Search from **both** the initial state and the goal.
    - Recall forwards + backwards chaining.
  - Each leads to a half-size tree (and with luck, they meet)!
- If the goal is at depth d time is $O(b^{(d/2)})$ (space too)
  - Must be able to define predecessors
  - Must have **explicit** notion of the goal(s), and not too many of them.
  - Must be able to efficiently check for a match