Day 34

COPING WITH THE LIMITATIONS OF ALGORITHMIC POWER

Tackling NP Problems

Two principal approaches to tackling problems in NP:

- Use a strategy that guarantees solving the problem exactly but doesn't guarantee to find a solution in polynomial time
- Use an *approximation algorithm* that can find an approximate (sub-optimal) solution in polynomial time

Exact Solution Strategies

Exhaustive search (brute force).
- useful only for small instances

Dynamic programming.
- applicable to some problems (e.g., the knapsack problem)

Backtracking
- eliminates some unnecessary cases from consideration,
- yields solutions in reasonable time for many instances, but worst case is still exponential

branch-and-bound.
- further refines the backtracking idea for optimization problems

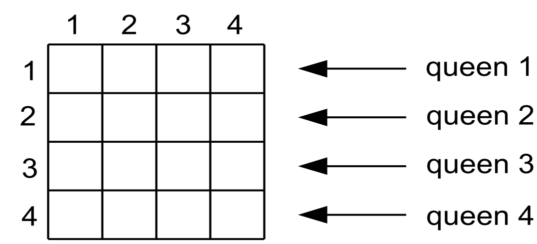
Backtracking

When applicable, backtracking is often much faster than brute force enumeration, since it eliminates many candidates with a single test

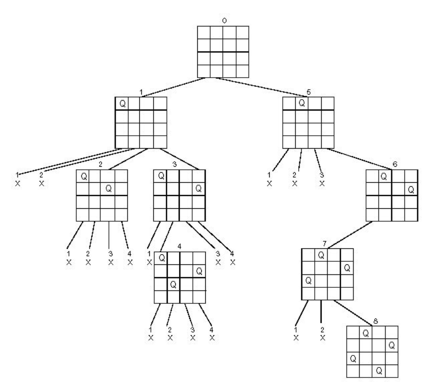
Process:
- Construct the ***state-space tree***
  - nodes: partial solutions
  - edges: choices in extending partial solutions
- Explore the state space tree using depth-first search
- "Prune" ***nonpromising nodes***
  - stop exploring subtrees rooted at nodes that cannot lead to a solution
  - backtrack to such a node's parent to continue the search

Example: *n*-Queens Problem

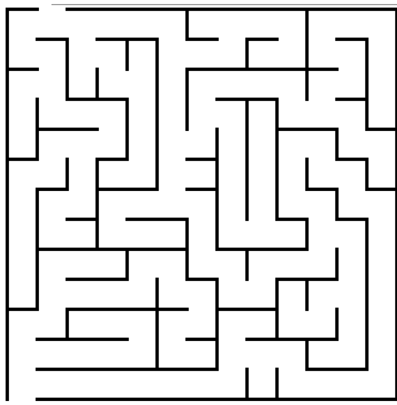
Place *n* queens on an *n*-by-*n* chess board so that no two of them are in the same row, column, or diagonal



State-Space Tree of the 4-Queens Problem



Familiar Problems



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Backtracking Notes

Generally just marginally better than exhaustive search (still asymptotically inefficient in the worst case)

Can exploit symmetries to further reduce cases

Rearrange data (e.g. presorting max remaining sums for Subset Sum)

Can estimate size of relevant state-space by generating random path from root to a leaf and counting the number of choices at each step

Branch-and-Bound

An enhancement of backtracking

For each node (partial solution) of a state-space tree, compute a bound on the value of the objective function for all descendants of the node (extensions of the partial solution)

Use the bound for:

- ruling out certain nodes as "nonpromising" to prune the tree – if a node's bound is not better than the best solution seen so far
- guiding the search heuristically through state-space

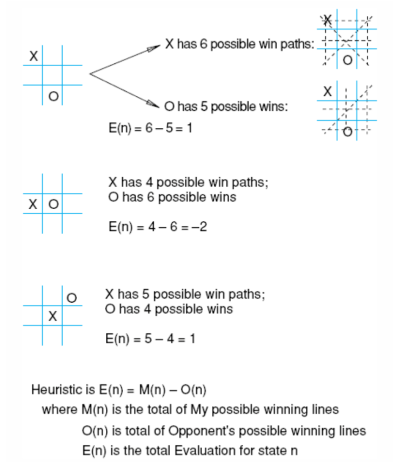
Branch-and-Bound in Playing Games

Chess, Go, Tic-Tac-Toe variants, etc.

Minimax rule: **mini**mize the possible loss for a worst-case (**max** loss) scenario (i.e., max the min gain)

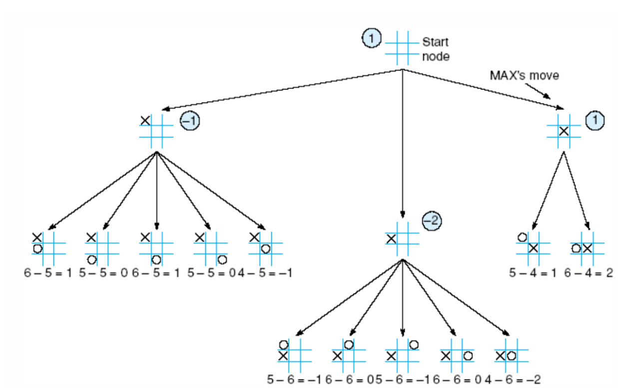
- For each state we compute a "score"
  - Ideally, ∞ for a winning position and –∞ for a losing position. But, usually too hard to compute this.
  - More practically, stop at base cases where we use a heuristic measure of "quality of position".
- Then, ancestors are
  - If my move (max): max of children's scores
  - If opponent's move (min): min of children's scores

MinMax for TicTacToe



Luger: Artificial Intelligence, 6th edition. © Pearson Education Limited, 2009

MinMax for TicTacToe



Luger: Artificial Intelligence, 6th edition. © Pearson Education Limited, 2009

Alpha-Beta Pruning

Branch-and-bound for minimax algorithm

- α: (maximum) lower bound on score of possible moves so far
- β: (minimum) upper bound on score of possible moves so far

Min nodes: opponent's moves. Take β = min of children's α's.

Max nodes: my moves. Take α = max of children's β's.

Stop evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move

E.g. if current α for max node is ≥ the β at a child min node, then nonpromising and can prune.

Alpha-Beta Pruning

Choose sidebar display

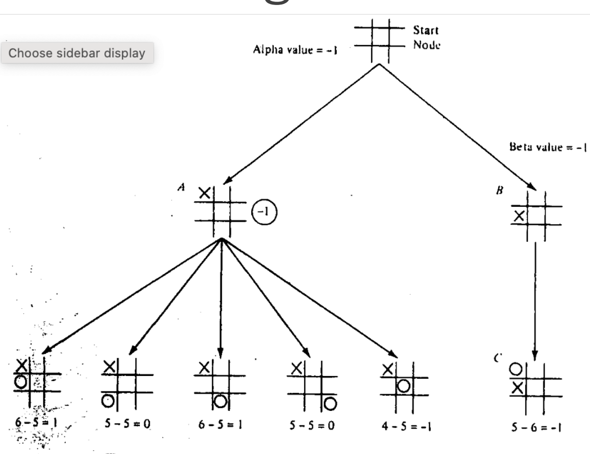


Fig. 3.11 Part of the first stage tic-tac-toe tree.

Image source: Nilson: Principles of AI, Tioga

Approximation Approach

Apply a fast (i.e., polynomial-time) approximation algorithm to get a solution that is not necessarily optimal but hopefully close to it

Accuracy measures:

- Change goes beyond a certain delta
- Alternatively, limit by CPU cycles and take and run

Approximation Approach

Genetic Algorithms for TSP

https://www.heatonresearch.com/aifh/vol2/tsp_genetic.html

Ant-colony optimization:

https://baobabsoluciones.es/en/blog/2020/10/01/travelling-salesman-problem-methods/