CPS 170
Alternative/Advanced Search Techniques
Ron Parr

With thanks to Vince Conitzer for LP,(M)IP examples.

Overview

• Memory-bounded Search
• Searching with Incomplete Information
• Local Search and Optimization
Memory-bounded Search: Why?

• We run out of memory before we run out of time

• Problem: Need to remember entire search horizon

• Solution: Remember only a partial search horizon

• Issue: Maintaining optimality, completeness
• Issue: How to minimize time penalty

Attempt 1: IDA*

• Iterative deepening A*
• Idea: Like IDDFS, but use the f cost as a cutoff
  – Cutoff all searches with f > 1, then f > 2, f > 3, etc.
  – Motivation: Cut off bad-looking branches early
• Problems:
  – Excessive node regeneration
  – Can still use a lot of memory

Cutoff = 3

h=1

h=2

h=1

h=1
Attempt 2: RBFS

- Recursive best first search
- Objective: Linear space

- Idea: Remember best alternative
- Rewind, try alternatives if “best first” path gets too expensive
- Remember costs on the way back up

Assume $h=1$, initially along this path.

Replace with $alt = 11$

Alt $= 16$

Problem: Thrashing!
SMA*

- Idea: Use all of available memory
- Discard the worst leaf when memory starts to run out, to make room for new leaves
- Values get backed up to parents
- Optimal if solution fits in memory
- Complete
- Thrashing still possible

Searching with Partial Information

- Multiple state problems
  - Several possible initial states
- Contingency problems
  - Several possible outcomes for each action
- Exploration problems
  - Outcomes of actions not known a priori, must be discovered by trying them
Example

• Initial state may not be detectable
  – Suppose sensors for a nuclear reactor fail
  – Need *safe* shutdown sequence despite ignorance of some aspects of state

• This complicates search *enormously*

• In the worst case, contingent solution could cover the entire state space

State Sets

• Idea:
  – Maintain a set of candidate states
  – Each search node represents a set of states
  – Can be hard to manage if state sets get large

• If states have probabilistic outcomes, we maintain a probability distribution over states

![Diagram of state sets with three sets branching from a central node]
Searching in Unknown Environments

- What if we don’t know the consequences of actions before we try them?
- Often called on-line search
- Goal: Minimize competitive ratio
  - Actual distance/distance traveled if model known
  - Problematic if actions are irreversible
  - Problematic if links can have unbounded cost

Optimization

- Solution is more important than path
- Interested in minimizing or maximizing some function of the problem state
  - Find a protein with a desirable property
  - Optimize circuit layout

- History of search steps not worth the trouble
Goal: Find values of problem features that maximize objective function.

Note: This is conceptual. Often this function is not smooth.

Hill Climbing

- Idea: Try to climb up the state space landscape to find a setting of the problem features with high value.
- Approaches:
  - Steepest ascent
  - Stochastic – pick one of the good ones
  - First choice
- This is a *greedy* procedure
Limitations of Hill Climbing

- Local maxima
- Ridges – direction of ascent is at 45 degree angle to any of the local changes
- Plateaux – flat expanses

Getting Unstuck

- Random restarts
- Simulated annealing (minimization)
  - Take downhill moves with small probability
  - Probability of moving downhill decreases with
    - Number of iterations
    - Steepness of downhill move
  - If system is “cooled” slowly enough, will find global optimal w.p. 1
  - Motivated by the annealing of metals and glass
Genetic Algorithms

- GAs are hot in some circles
- Biological metaphors to motivate search
- Organism is a word from a finite alphabet (organisms = states)
- Fitness of organism measures its performance on task (fitness = objective)
- Uses multiple organisms (parallel search)
- Uses mutation (random steps)

Crossover

Crossover is a distinguishing feature of GAs:

Randomly select organisms for “reproduction” in accordance with their fitness. More “fit” individuals are more likely to reproduce.

Reproduction is sexual and involves crossover:

Organism 1:  \[1 1 0 0 1 0 0 1 0\]
Organism 2:  \[0 0 0 1 0 1 1 0\]
Offspring:   \[1 1 0 0 1 1 1 0\]
Is this a good idea?

- Has worked well in some examples
- Can be very brittle
  - Representations must be carefully engineered
  - Sensitive to mutation rate
  - Sensitive to details of crossover mechanism
- For the same amount of work, stochastic variants of hill climbing often do better
- Hard to analyze; needs more rigorous study

Continuous Spaces

- In continuous spaces, we don’t need to “probe” to find the values of local changes
- If we have a closed-form expression for our objective function, we can use the calculus
- Suppose objective function is: \( f(x_1, y_1, x_2, y_2, x_3, y_3) \)
- Gradient tells us direction and steepness of change

\[
\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)
\]
Following the Gradient

\[ \mathbf{x} = (x_1, y_1, x_2, y_2, x_3, y_3) \]

\[ \mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x}) \]

For sufficiently small step sizes, this will converge to a local optimum.

If gradient is hard to compute:
- Compute empirical gradient
- Compare with classical hill climbing

Constrained Optimization

- Don’t forget about the easier cases
  - If you have a linear objective function with linear constraints, solve as a linear program:
  - Maximize (minimize): \( f(\mathbf{x}) \leftarrow \) Linear function of vector \( \mathbf{x} \)
  - Subject to:
    \[ \mathbf{A} \mathbf{x} \leq \mathbf{b} \]
    \[ \mathbf{A} \mathbf{x} \geq \mathbf{b} \]
  - Can be done in polynomial time
  - Can solve some quadratic programs in poly time
- How is this search? Searches space of values of \( \mathbf{x} \).
Linear programs: example

• Make reproductions of 2 paintings

\[
\begin{align*}
\text{maximize } & 3x + 2y \\
\text{subject to } & 4x + 2y \leq 16 \\
& x + 2y \leq 8 \\
& x + y \leq 5 \\
& x \geq 0 \\
& y \geq 0
\end{align*}
\]

• Painting 1:
  • Sells for $30
  • Requires 4 units of blue, 1 green, 1 red

• Painting 2
  • Sells for $20
  • Requires 2 blue, 2 green, 1 red

• We have 16 units blue, 8 green, 5 red

Solving the linear program graphically

\[
\begin{align*}
\text{maximize } & 3x + 2y \\
\text{subject to } & 4x + 2y \leq 16 \\
& x + 2y \leq 8 \\
& x + y \leq 5 \\
& x \geq 0 \\
& y \geq 0
\end{align*}
\]

optimal solution: \( x=3, y=2 \)
Modified LP

*maximize* $3x + 2y$

subject to

$4x + 2y \leq 15$

$x + 2y \leq 8$

$x + y \leq 5$

$x \geq 0$

$y \geq 0$

Optimal solution: $x = 2.5$, $y = 2.5$

Solution value $= 7.5 + 5 = 12.5$

Half paintings?

Integer (linear) program

*maximize* $3x + 2y$

subject to

$4x + 2y \leq 15$

$x + 2y \leq 8$

$x + y \leq 5$

$x \geq 0$, integer

$y \geq 0$, integer

Optimal LP solution: $x = 2$, $y = 3$

(objective 12)

Optimal LP solution: $x = 2.5$, $y = 2.5$

(objective 12.5)
Mixed integer (linear) program

\[
\text{maximize } 3x + 2y \\
\text{subject to } \\
4x + 2y \leq 15 \\
x + 2y \leq 8 \\
x + y \leq 5 \\
x \geq 0 \\
y \geq 0, \text{ integer}
\]

Solving linear/integer programs

- Linear programs can be solved efficiently
  - Simplex, ellipsoid, interior point methods...
  - Standard packages for solving these
    - GNU Linear Programming Kit, CPLEX, ...

- (Mixed) integer programs are intractable to solve
  - No known efficient (guaranteed run time less than exponential) algorithms
  - Solvers use standard search-like algorithms
Conclusions and Parting Thoughts

- There are search algorithms for almost every situation
- Many problems can be formulated as search
- While search is a very general method, it can sometimes outperform special-purpose methods