9. Approximation algorithms

Topic 9: Approximation and probabilistic algorithms

1. Approximate and randomized algorithms
2. Iterative improvement
3. State-space search
4. Knowledge, search, and rationality
5. Evolutionary computation

Reading: Ch. 11

Topic objectives

9a. Explain approximation approaches to solving intractable problems
9b. Explain advantages and examples of randomized algorithms
1. Approximation and randomized algorithms

Why compute an approximation rather than an exact result?

- Exact solution algorithm may take too much time (Example: shortest tour through n cities)
- Exact results may have infinitely large representation (Example: square root)
- Approximate solution may be used as part of a larger exact algorithm

Approximation algorithms

- Many use heuristics: rules of thumb based on human experience; e.g., putting largest item in box first
- Example of heuristic: putting largest item in knapsack first
- Accuracy ratio: \( f(s_a) / f(s^*) \), where heuristic \( s_a \) solves the problem of minimizing \( f \) and \( s^* \) is an exact solution
- Performance ratio: lowest upper bound on accuracy ratio
- \( c \)-approximation algorithm: one whose performance ratio is at most \( c \), i.e., \( f(s_a) \leq f(s^*) \) for any instance
- Continuous functions: Newton’s method
  \[-x_{n+1} = (x_n + f(x_n) + f'(x_n)) / 2\]
gives approximate sequence; e.g., for square root
Randomized algorithms

- Use of probabilistic control usually leads, with sufficient time, to good approximate solutions
- Often used for optimization problems
- Examples: natural evolution; evolutionary computation, which generates a population of solutions stochastically, tests them, and uses test results to generate more

Randomized solution to SAT

- Flip coins repeatedly to set variables’ truth-values, seeking a set of values that satisfy formula
- Advantage: If many interpretations satisfy formula, this algorithm will find a solution quickly
- Disadvantage: Is likely to run long or give false negative result if a small number of interpretations satisfy formula
2. Iterative improvement

- **Problem class**: Optimization under constraints
- **Strategy**: Find a feasible solution, improve it by successive steps
- **Obstacle**: Global maxima/minima (the objective) may differ from local ones
- **Cases**:
  - Simplex method for linear programming
  - Maximal flow
  - Bipartite matching
  - Hill climbing

Linear programming and Simplex

- Linear-programming problem: maximize or minimize \( c_1 x_1 + \ldots + c_n x_n \) s.t. for \( i = 1 \ldots m \),
  \[ a_{i1} x_1 + \ldots + a_{in} x_n \leq b_i \text{ or } \geq b_i \text{ or } = b_i \]
- **Example**: Maximize \( 3x + 5y \), given simultaneous equations
  \( x + 3y \leq 6, x + y \leq 4, x \geq 0, y \geq 0 \)
- **Solution**: Feasible region (p. 338a), with maximal point (3, 1)
- **Simplex method**: Find one extreme point in feasible region; cohose adjacent extreme points as long as they give improved value
9. Approximation algorithms

Simplex method algorithm

- **Algorithm:**
  Initialize tableau
  While some entries in “objective row” are < 0
  Find entering variable (< 0) in obj. row
  Find departing var. in entry var.’s column
  Compute new tableau from old

- **Complexity:**
  - Non-terminating in rare cases
  - Worst case: $O(n^2)$
  - Expected in practical use: $O(m^2n)$

The maximum-flow problem

- Assume connected weighted digraph with $n$ vertices, edge set $E$
- Digraph has exactly one sink, one source
- Weight of edge, $u_{ij}$, is capacity to handle flow
- Maximum-flow problem is to maximize flow $v = \Sigma x_{ij}$ for all $j$ where $(1, j) \in E$, subject to constraint that inflow, outflow of each vertex are equal
9. Approximation algorithms

Solving maximal-flow problem

- Repeatedly find flow-augmenting paths starting at zero flow
- *Cut:* set of edges with tail in one partition of vertices (containing source) and head in the other partition (containing sink)
- *Edmunds-Karp:* Shortest-augmenting path algorithm, using breadth-first search to find shortest augmenting paths
- *Complexity:* \(O(nm^2)\)
- Edmunds-Karp uses theorem that value of maximum flow equals capacity of minimum cut

Maximum matching

- *Matching* (in a graph): a subset of edges s.t. no two edges share a vertex
- *Maximum matching:* A matching with a maximum number of edges
- *Example:* Where \(E\) represents common interests among people, find best pairing of a set of people so that the most pairs share a common interest. *Complexity:* \(O(2^{|E|})\)
- *Brute-force solution:* Generate all subsets \(S\) of \(E\); for each subset check all \(((u, v), (w, x)) \in S \times S\) for a common vertex
Maximum matching in bipartite graphs

- Simpler problem: $G = (V, E)$ where $V$ is partitioned into disjoint sets $V_1, V_2$, s.t. $E \subseteq V_1 \times V_2$ (every element of $V_1$ is paired with one in $V_2$ and vice versa)
- Suppose $M$ is a matching in $G$; then if every vertex in $V_1$ or every one in $V_2$ is matched, $M$ cannot be improved
- But $M$ can be improved if an augmenting path is found that yields a larger matching
- Complexity: $O(|V|^2 + |V||E|)$

Stable-marriage problem

- A bipartite matching problem (selecting pairs, one item from each of two sets)
- Each element of each partition has a totally ordered preference list
- Stable pairing: one in which no unmatched pair exists whose the members would prefer each other to their current matches
- Algorithm:
  - While some element $x$ of $V_1$ is unmatched
  - $x$ proposes to next-preferred $y \in V_2$
  - $y$ accepts if unmatched or if matched with a less-preferred partner
- Complexity: $O(n^2)$
3. State-space search

- **State space**: A set of possible arrangements of values, e.g.:
  - Board configurations in board games
  - Paths in a graph
  - Arrangements of items in a knapsack
  - Assignments of truth values in a formula

- **Transitions** may be defined from state to state, which we can express as a tree

- **Goal**: Reduce size of state-space tree explored by algorithm, by pruning branches that cannot lead to a solution

- **Example**: n-queens chess problem

---

**Generate and test**

A general-purpose algorithm for state-space search

To solve a given problem:

1. **Design**:
   a. a test of proposed solutions
   b. a generator of possible solutions

2. **While** (~problem-solved ∧ ~time-expired)
   - generate a possible solution
   - test it

[Simon, Newell, 1976]
State transition systems

- Let $S$ be state space of solutions to a problem, then $S_1$ are the 1-component prefixes to solutions (1-edge paths, first moves in a game, etc.)
- Let $T = \langle S, \Rightarrow \rangle$ be a transition system where
  - $S$ is a set of states
  - $\Rightarrow$ is a transition relation in $(S \times S)$
- For $q \in S$, $\text{next}(q) = \{ \sigma \in S \mid q \Rightarrow \sigma \}$
- Example: Rules of chess, paths in graph
- A solution path is a sequence of states $X[1..n]$ that satisfies a goal constraint, and $(\forall i \leq n) X[i] \Rightarrow X[i + 1]$

Example: Finding a path

- Problem: In graph $G$, is there a path from vertex $a$ to vertex $b$?
- Generate-and-test solution consists of exploring the tree of paths from vertex $a$ to others
- One way to generate paths is depth-first: the next possible solution to generate is the previous one, extended by an edge
- To test a path, just see if it ends at $b$
- Note: This problem is easier than most state-space searches
Iterated local search (hill climbing)

**Search (S)**

```
repeat
  y ← random (S)
  repeat
    for each t ∈ T
      c' ← t(c)
      if eval(c') > eval(c)
        c ← c'
        changed ← true
      else
        changed ← false
    until ¬changed
  until eval(c) is acceptable
return c
```

- Explores search space $S$ using randomization, transformation set $T$, and fitness function $eval$
- Addresses n-queens, TSP, SAT
- Similar algorithm, simulated annealing, solves independent-set problem

Approaches to some hard problems

- *Backtracking* and *branch-and-bound* solve some search problems in large state spaces
- It is very hard to predict whether these approaches will solve a particular problem in reasonable time
- *Backtrack example:* Finding an English word that satisfies a certain definition [why?]
- *Branch-and-bound example:* Finding best-score word in a turn at Scrabble
9. Approximation algorithms

**Backtrack** \((X[1 .. i])\)
If \(X[1 .. i]\) is a solution
   return \(X[1 .. i]\)
else
   for each state \(q\) in next(\(X[1 .. i]\))
      \(X[i + 1] \leftarrow q\)
   return Backtrack \((X[1 .. i + 1])\)
Return \textit{fail}

• How does this differ from depth-first search?

**Branch-and-bound**
• In optimization problems, solutions may be
  – \textit{Feasible} (satisfy constraints)
  – \textit{Optimal} (yield best value of objective function, e.g., least-weight path)
• In branch-and-bound, search of a state-space tree stops when
  – Node represents no feasible solution, or only one; or
  – Value of node’s bound \textbf{[define]} is not better than best found so far
• \textit{Example}: Assign \(n\) people to \(n\) jobs with best overall fit
4. Knowledge, goals, and rationality

• Levels of a computer system: device, circuit, symbol, … knowledge (Newell, 1982)

• Principle of rationality: An agent will select an action if it has knowledge that the action will lead to a system goal

• Knowledge: Whatever an agent has that enables it to compute its actions according to the principle of rationality

Bounded rationality

• Notion suggested by Herbert Simon, 1972, as alternative to classical rationality assumption of economic theory

• Argument: Humans have limited knowledge and resources for decision making

• Alternative goal to optimality: satisficing

• Rational agent: one that chooses actions that yield maximum expected utility averaged over all outcomes
## Game theory

- Originated in economic theory  
  - mathematical methods for study of decisions  
  - notion of rational behavior (acting in own interests)  
  - notion of utility  
  - generalization of the notion of game strategy  
- Variants (Von Neumann-Morgenstern, 1947):  
  - zero-sum, non-zero-sum;  
  - 2-player, multi-player;  
  - perfect-information, imperfect-information

### Example: Playing a game

- To choose a move by *Generate-and-test*, generate possible moves, test them with an evaluation function (e.g., allocating points for pieces-won by the move)
- Finding good moves often entails *lookahead* and *backtrack* in the game tree
The minimax algorithm

- Used for problems with an adversary; e.g., two-player games
- Assuming an optimal opponent, let the best move be the one that would yield the best-valued situation if the opponent replies with his/her best move
- To determine that, apply the same algorithm from opponent’s viewpoints

Example: Tic-tac-toe

- State space: the set of all possible board positions
- Let a position’s value be 1.0 if we win, 0.0 if we lose
- Also, if we can force a win (as at right), value is 1.0
POMDPs

• **Partially observable Markov decision processes** are problems in inaccessible stochastic environments with the Markov property.

• **Markov property**: Probability that system will be in a given state at next step depends only on current state, not on past history (L. Lipsky)
5. Evolutionary computation

- A probabilistic way to develop approximate solutions to difficult problems
- Modeled on natural evolution of species and behaviors of some life forms
- A growing research area encompassing genetic algorithms, evolutionary programming, evolution strategies, ant computing, swarm computing, particle swarm optimization

The function-optimization problem

- Let $f : \mathbb{N}^k \to \mathbb{R}$ for some $k$ (the arity of $f$)
- Problem: Find some $x \in \mathbb{N}^k$ s.t. $f(x)$ is maximal
- Example: Suppose $x$ is the set of proportions of ingredients in a fuel mixture, $f(x)$ is fuel efficiency under this mixture
- Optimizing $f(x)$ means finding the most efficient mixture
- For an algorithm to optimize a function we must have $f : X \to Y$ with $X, Y$ finite
Fitness and function optimization

- **Example**: Suppose $f$ is viewed as a fitness function and $x$ is the set of attributes of individuals of a population.

- Then finding $x$ s.t. $f(x)$ is maximal is finding the fittest possible individual of the species, i.e., those with the best attributes to assure survival.

- Evolution by natural selection tends to optimize fitness, over many generations.

Evolutionary computation

- State space is explored using and comparing a population of states rather than one at a time.

- Evolutionary algorithm repeatedly modifies and selects from a population of solutions.

- Selection is driven by fitness of each member of the population.

- Randomization is used to explore state space.
The evolutionary algorithm

\[ t \leftarrow 0 \]
\[ \text{Initialize} \ (P_0) \]
\[ V \leftarrow \text{Evaluate} \ (P_0) \]

While not \text{Terminate} \ (V, t) do

\[ t \leftarrow t + 1 \]
\[ P_t \leftarrow \text{Select} \ (P_{t-1}, V) \]
\[ P_t \leftarrow \text{Alter} \ (P_t) \]
\[ V \leftarrow \text{Evaluate} \ (P_t) \]

Return \( P_t \)

- \text{Evaluate} applies fitness function to individuals
- \text{Select} chooses some members of \( P \) to survive
- \text{Alter} changes \( P \) randomly, e.g., by mutation or crossover
- \text{Terminate} ends algorithm after a given goal or a time deadline is reached

Genetic algorithms

- At each step, the population is selected and regenerated using genetic operators, e.g., mutation and crossover
- \text{Parameters}: % of population to retain at each generation, which mate and produce offspring, which probability measures, if any, to use
Example: Checkers (Samuel, 1950s)

- Let fitness function $f: \mathbb{N}^k \rightarrow \mathbb{R}$ be an evaluator of checkers board positions from a black or red angle.
- Let $x \in \mathbb{N}^k$ be a $k$-tuple of weights for each of $k$ different criteria for evaluating a checkers position.
- Example: let $x_1$ be relative importance of number of kings, $x_2$ be relative importance of number of opponent checkers threatened, etc.
- Then writing a good checkers-playing program reduces to finding a good set of relative weights $x_1, \ldots, x_k$ for these criteria.
- Result: machine play at very high level.

Concepts

- approximation algorithm
- backtracking
- bounded rationality
- branch-and-bound
- evolutionary computation
- fitness
- function optimization
- game theory
- generate-and-test
- global maxima
- local maxima
- iterative improvement
- Markov property
- maximal flow
- maximal matching
- Minimax
- POMDP
- probabilistic algorithm
- rationality
- simplex method
- stable marriage
- state-space search
9. Approximation algorithms

References


References cont’d

