Branch-and-Bound (B&B)

- Variant of backtrack with *costs*
  - Associate a *cost* with a partial solution, such that the cost of a parent is always *less than or equal to* the cost of its child in the decision tree
  - do not *branch* from an internal node whose cost is higher than the current *bound* = cost of the minimum-cost complete solution found so far
  - the *bound* is updated if a better solution is found
- Key points
  - Used for *optimization* problems
  - Cost-driven
  - Bounding prunes the decision tree, saves time
Branch-and-Bound Example: Game Tree

• In games (e.g., chess) can model the different stages of the game by a rooted tree
  – Don’t need to consider all possible situations of a game
  – Can predict the outcome of the game using the concept of branch-and-bound

• Questions
  – Who is the winner if both players play optimally?
  – How much is the payoff?
    • E.g., we may initially only know the payoff of the terminal nodes (end-states) of the game.
Two-Player Game Tree

- Payoff of the Game = amount the 1^{st} player receives at the end of game

  1^{st} player wants to \textit{maximize} payoff (Call her Max)

  2^{nd} player wants to \textit{minimize} payoff (Call her Min)
We can see that the value of the game is 1 (node A) by examining (bottom-up) the entire game tree. In general, (bottom-up) examination of an entire game tree is not feasible. Applying DFS with branch-and-bound using “α-β pruning” improves efficiency.
Alpha-Beta Pruning in Game Tree

- Idea: Exploit upper and lower “cut-off” values in the game tree

  - If value of a Max child is $\alpha$, then $\alpha$ is a lower cut-off value on value of Max
  
  - If Max has lower cut-off value $\alpha$, then all Max’s children have lower cut-off value $\alpha$

Example: R has value $\alpha$, so any other descendant of Q with value less than $\alpha$ can be ignored, because Q is a “MAX” node. $\alpha$ is a lower bound or “lower cut-off” on values of S, T, U

Q, S, T, U only care about values that are $\geq \alpha$
Alpha-Beta Pruning (cont.)

- If value of a Min child is $\beta$, then $\beta$ is an upper cut-off value on value of Min.

- If Min has upper cut-off value $\beta$, then all Min’s children have upper cut-off value $\beta$.

Example: R has value $\beta$, so any other descendant of Q with value greater than $\beta$ can be ignored, because Q is a “MIN” node. $\beta$ is an upper bound or “upper cut-off” on values of S, T, U.

Q, S, T, U only care about values that are $\leq \beta$. 
After D has value 1, the lower cut-off value (= lower bound) of C is 1, and E has lower cut-off value 1 as well.

The left son of E is 1, so the value 1 becomes an upper cut-off value on E.

E has upper and lower cut-off values both equal to 1, so we can write 1 in E without looking at its right child.

Even though the “correct” value of E is -3, we can write 1 in E since this doesn’t affect the value of C.
The value of $C = \max(D,E) = \max(1,1) = 1$. This value of $C$ becomes the upper cut-off value of $B$ and its descendants, $F$, $G$, $H$ etc.

The value of $G = \min(3,2) = 2$, which is greater than the upper cut-off value 1.

Once we have value 2 for $G$, this is a lower cut-off value for $F$. (If we write 1 inside $G$, we get the same value at $B$.)

$F$ has lower cut-off value 2, and upper cut-off value 1, so the value of $F$ is determined without looking at $H$ or its children.

Write $F = 2$. (When lower cut-off > upper cut-off, use lower cut-off for a Max node, and upper cut-off for a Min node.)
The value of \( B = \min(C, F) = \min(1, 2) = 1 \). This is a lower cut-off value for \( A \) and all of \( A \)'s descendants \( I, J, K, L, M, \) etc.

Now the values of the children of \( K \) are upper cut-off values for \( K \), so we have \( 1 \leq K \), and \( K \leq -2 \). Since \( K \) is a MIN node, we use \(-2\) as the value of \( K \).

\( K = -2 \) is a lower cut-off value for \( J \), and since \( J \) has a lower cut-off value \( 1 \), we can write down the lower cut-off value \( 1 \) for node \( J \) (MAX node).
The value of J = max(K,L) = max(-2,-4) = -2. Since the value of J is an upper cut-off value for I, we have 1 ≤ I and I ≤ -2. We use -2 for I since I is a MIN node.

Now the value of A is determined as A = max(B,I) = max(1,-2) = 1. The value of A is determined without looking at M or any of its children!
Game Tree Example (after pruning)
## Kinds of Algorithms

<table>
<thead>
<tr>
<th>Speed</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast</td>
<td>exact: Short and sweet</td>
</tr>
<tr>
<td>slow</td>
<td>exact: Slowly but surely</td>
</tr>
</tbody>
</table>
Metaheuristic

• Method for dealing with large, practical, intractable optimization problems in the real world
  – Planning and scheduling
  – Routing and flows
  – Assignment and placement
  – Clustering and classification

• Intractable problems
  – = NP-hard
  – = instance complexity (“n”) too large to handle with known efficient algorithms
Rest of Slides

- Iterative Global Optimization
- Simulated Annealing
- Tabu Search
- Genetic Algorithms
Iterative Global Optimization

- **S = universe of solutions** // aka “solution space”
  - \(n!\) TSP tours over \(n\) cities
  - \(C(n, n/2)\) graph bisections
  - \(k^n\) assignments of \(k\) colors to graph vertices

- **cost or objective function**
  - Cost of TSP tour
  - Cutsise (# edges cut) in graph bisection
  - #colors needed for valid graph coloring

- **N(s) = neighborhood of a given solution \(s \in S\)**
  - Interchange positions of two cities in tour \(\rightarrow C(n,2)\) neighbors
  - Swap two vertices between partitions \(\rightarrow n^2/4\) neighbors
  - Change the color of a vertex \(\rightarrow (k - 1) \cdot n\) neighbors
  - Flip a truth assignment of a variable in SAT \(\rightarrow k\) neighbors
Iterative Global Optimization

- S = universe of solutions  // aka “solution space”
- cost or objective function
- N(s) = neighborhood of a given solution s ∈ S

Neighbor solution in SAT = ?
Iterative Global Optimization

• $S = \text{universe of solutions} \quad \text{// aka “solution space”}$
• cost or objective function
• $N(s) = \text{neighborhood of a given solution } s \in S$

Neighbor solution in NumberPartition = ?
Iterative Global Optimization

• \( S = \) universe of solutions // aka “solution space”
• cost or objective function
• \( N(s) = \) neighborhood of a given solution \( s \in S \)

Neighbor solution in TSP = ?
Iterative Global Optimization

• $S =$ universe of solutions  // aka “solution space”
• cost or objective function
• $N(s) =$ neighborhood of a given solution $s \in S$

Neighbor solution in Graph Bisection = ?
Iterative Global Optimization

- $S = \text{universe of solutions}$  // aka “solution space”
- cost or objective function
- $N(s) = \text{neighborhood of a given solution } s \in S$

**Iterative Global Optimization**

start with an initial solution $s_0$
for $i = 1$ to $M$  // $M = \text{time limit, stop criterion, etc.}$
generate candidate solution $s \in N(s_{i-1})$
decline between $s_i = s_{i-1}$ or $s_i = s$
return $s_M$  // “where you are” == $s_M$
    // “best so far” == best over $s_0, ..., s_M$
Local, Global Minima
Simulated Annealing (SA)

- Kirkpatrick, Gelatt, Vecchi, *Science (1983)*: *One of the most cited scientific papers ever*

- SA is one of many “metaheuristics” that are used to deal with instances of intractable (NP-hard) combinatorial problems
  - Genetic algorithms (Holland, U. Michigan)
  - Tabu search (Glover, U. Colorado)
  - Etc.

- Combinatorial optimization has a physical analogy to the annealing (slow cooling) of metals to produce a perfectly-ordered, minimum-energy state: *a “state” is a “solution”, “energy” is “cost”, etc.*
Simulated Annealing Basic Idea

- **Initialize** – Start with a random initial solution. Initialize high “temperature” = a parameter, “T”
- **Step 2: “Move”** – Perturb current solution to obtain a ‘neighbor’ solution
- **Step 3: Calculate cost change** – calculate the change in solution cost due to the move (minimization: negative change is better, positive change is worse)
- **Step 4: Accept/Reject** – Depending on the cost change, accept or reject the move. Probability of acceptance depends on current “temperature”.
- **Step 5: Update** – Update temperature, current solution. Go to Step 2.
- Continue until termination condition (‘freezing’ or ‘quenching’) is satisfied
Algorithm SIMULATED-ANNEALING
Begin
    \( temp = \text{INIT-TEMP}; \)
    \( currentSol = \text{INIT-SOLUTION}; \)
    for \( i = 1 \) to \( M \)
        \( candidateSol = \text{NEIGHBOR}(currentSol); \)
        \( \Delta C = \text{COST}(candidateSol) - \text{COST}(currentSol); \)
        if \( \Delta C < 0 \) then
            \( currentSol = candidateSol; \)
        else with \( Pr = e^{-(\Delta C/\text{temp})} \)
            \( currentSol = candidateSol; \)
    \( temp = \text{SCHEDULE}(\text{temp}); \)
End

What happens when \( temp = +\infty \) ?
What happens when \( temp = 0 \) ?
Simulated Annealing Facts

- **Fact 1.** NEIGHBOR(solution) defines a topology over all solutions in the solution space.
- **Fact 2.** At a fixed value of *temp*, SA behavior corresponds to a homogeneous Markov chain:
  - Fixed *temp* → fixed matrix of transition probabilities between states
Simulated Annealing Facts

- **Fact 3.** The steady-state (= equilibrium) probability of the Markov chain being in state $A$ is proportional to $e^{-\frac{\text{cost}(A)}{\text{temp}}}$
  - When $\text{temp} \to 0$, exponentially more likely to be in the global optimum state
  - “SA is optimal” (in the limit of ‘infinite time’)
  - Of course, we spend only a finite amount of time (#moves) at any temperature value
  - *Is cooling the best strategy with finite time?* See Boese/Kahng, 1993

---

Initial state

SA chooses uphill move with nonzero probability (“hill-climbing”)

Greed gets stuck here, in a local optimum

SA converges to global opt solution with $\text{Pr} = 1$
(in limit of infinite time, infinitely slow cooling)
Optimal SA Temperature Schedules

- 6-city Traveling Salesman instance
- $M = 160$ steps
- “Where-You-Are” (top)
- “Best-So-Far” (bottom)
Optimal SA Temperature Schedules

- 8-vertex Graph Bisection instance
- $M = 160$ steps
- “Where-You-Are” (top)
- “Best-So-Far” (bottom)
Useful Idea: “Large-Step Markov Chain”

• (1) Run with Temp = 0 to find a local minimum
• (2) Run (briefly) with Temp = ∞ as a “kick move” to escape the local minimum
• Alternate (1) and (2) until move budget is expended

• Cf. “optimal” (WYA) schedule on previous slide
• http://vlsicad.ucsd.edu/Publications/Journals/j29.pdf

Improved Large-Step Markov Chain Variants for the Symmetric TSP

INKI HONG, ANDREW B. KAHNG
UCLA Computer Science Department Los Angeles, CA 90095-1596 USA

BYUNG-RO MOON
Design Technology Research Center, LG Semicon Co., Ltd. 16 Woomyon-dong, Seocho-gu, Seoul, Korea

{inki, abh}@cs.ucla.edu
moon@lgsemicon.co.kr

(LSMC is a Greedy Algorithm)
(with a complicated “move”…)

Useful Idea: “Go With The Winners”

• (1) Run iterative global optimization on K processors
  == population of K “walkers”
• (2) Every so often, identify the K’ << K best solutions
  == “winners”
• (3) Replicate the K’ solutions onto the K processors
  == “go with the winners”
• (4) Return to (1)

Useful with parallel/distributed compute resources
(The “Big Valley”)
(The “Big Valley”)

“dist” from best tour

“big valley” (1993)

“adaptive multistart” (best)
Outline

• Iterative Global Optimization
• Simulated Annealing
• Tabu Search
• Genetic Algorithms
Tabu Search (Glover, 1986)

• What
  – Neighborhood search + memory
  • Neighborhood search
  • Memory
    – Record the search history – the “tabu list”
    – Forbid cycling search

[Source: Gang Quan, Van Laarhoven, Aarts, http://web.cecs.pdx.edu/~mperkows/CLASS_574/574-fall-08/0001_SimulatedAnnealingAndTabuSearch.ppt]
Tabu Search Algorithm

• (1) Choose initial solution $s_0$
• (2) Find best $s' \in N(s_i)$ that is not on tabu list
• (3) If $F(s') > F(s_i)$ // $x'$ is better than $x$
  – $s_{i+1} = s'$
  – update tabu list
• (4) Go to step (2)

Note: This is Iterative Global Optimization!

[Source: Gang Quan, Van Laarhoven, Aarts, http://web.cecs.pdx.edu/~mperkows/CLASS_574/574-fall-08/0001_SimulatedAnnealingAndTabuSearch.ppt]
Tabu Search Tuning Parameters

- Local search procedure
- Neighborhood structure
- Criteria for tabu moves
- Update (and maximum size) of tabu list
  - Smaller list: “intensification” of search
  - Larger list: “diversification” of search
- Aspiration conditions
  - “aspiration” allows a tabu move to be used if it is sufficiently helpful
- Stopping condition/rule

Lots of parameters to tune!

[Source: Lecture slides from Lei Li, HongRui Liu, Roberto Lu, http://www.cs.ucla.edu/~rosen/161/TabuSearch.ppt]
Outline

• Iterative Global Optimization
• Simulated Annealing
• Tabu Search
• **Genetic Algorithms**
Genetic Algorithms

• A genetic algorithm (or GA) uses techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination)

• (1) Given a population at Generation i of solution representations (encodings)
• (2) Evaluate individuals’ fitness according to solution attributes
• (3) Propagate individuals’ attributes to next Generation i + 1 via selection (according to fitness), mutation, crossover, inversion, etc. operators

[Source: Muhannad Harrim, Western Michigan University]
https://www.cs.wmich.edu/elise/courses/cs6800/Genetic-Algorithms.ppt
Evolutionary Biology Metaphor

- **Individual** - Any possible solution
- **Population** - Group of all *individuals*
- **Chromosome** - Blueprint for an *individual*
- **Genome** - Collection of all *chromosomes* for an *individual*

- **Trait** - Possible aspect (*features*) of an *individual*
- **Allele** - Possible settings of trait (blue, brown, etc.)
- **Locus** - The position of a *gene* on the *chromosome*

[Source: Muhannad Harrim, Western Michigan University]  
https://www.cs.wmich.edu/elise/courses/cs6800/Genetic-Algorithms.ppt
Chromosome, Genes and Genomes

[Source: Muhannad Harrim, Western Michigan University]
https://www.cs.wmich.edu/elise/courses/cs6800/Genetic-Algorithms.ppt
GA Pseudocode

Choose initial population

Evaluate the fitness of each individual in the population

Repeat

   Select best-ranking individuals to reproduce

   Breed new generation through crossover and mutation (genetic operations) and give birth to offspring

   Evaluate the individual fitnesses of the offspring

   Replace worst-ranked part of population with offspring

Until <terminating condition>

[Source: Muhannad Harrim, Western Michigan University]
https://www.cs.wmich.edu/elise/courses/cs6800/Genetic-Algorithms.ppt
Representation

Chromosomes could be:

- Bit strings (0101 ... 1100)
- Real numbers (43.2 -33.1 ... 0.0 89.2)
- Permutations of element (E11 E3 E7 ... E1 E15)
- Lists of rules (R1 R2 R3 ... R22 R23)
- Program elements (genetic programming)
- ... any data structure ...

[Source: Muhannad Harrim, Western Michigan University]
https://www.cs.wmich.edu/elise/courses/cs6800/Genetic-Algorithms.ppt
A Fitness Function

[Source: Muhannad Harrim, Western Michigan University]
https://www.cs.wmich.edu/elise/courses/cs6800/Genetic-Algorithms.ppt
Crossover and Mutation Operators

[Source: Muhannad Harrim, Western Michigan University]
https://www.cs.wmich.edu/elise/courses/cs6800/Genetic-Algorithms.ppt
Provably Good Heuristics:
Engineer’s Method for Number Partitioning
Next-Fit Bin Packing
Christofides’ Euclidean TSP Approximation
Number Partitioning

- Number Partitioning Problem: Given a set of numbers, divide it into two piles, such that the sums of numbers in each pile differ by as little as possible.

- How would you solve this?

- The “Engineer’s Method” for Number Partitioning:
  - Sort the numbers from largest to smallest.
  - Beginning with the largest number, put each successive number into the pile that has smaller sum.

- Performance Ratio: For a given instance (set of numbers) S of Number Partitioning, the performance ratio of the Engineer’s Method (EM) is
  \[
  \frac{\text{sum of #'s in larger EM pile}}{\text{min possible sum of #'s in larger pile}}
  \]

- Worst-case example: 3,3,2,2,2 → Performance Ratio = 7/6
Bin Packing Problem

• Given an infinite supply of unit-capacity bins, and a list of items $i_1, i_2, \ldots$, pack the items into a minimum number of bins without exceeding any bin capacity.

• Online Bin Packing: Must assign each item to a bin as soon as it arrives (!)
  – Bagging groceries, cutting stock (pipes, lumber), etc.

• Performance Ratio: For a given instance (list of items) $L$, the performance ratio of an online bin packing heuristic $H$ is the limit, as the number of items in $L \rightarrow \infty$, of the maximum ratio

($\#\text{bins used by } H \text{ to pack } L) / (\text{Opt } \#\text{bins needed to pack } L)$
Iterated 1-Steiner Algorithm (1990)

Given a pointset $S$, what point $p$ minimizes $\text{MST}(S \cup \{p\})$

Algorithmic idea: Iterate! (greedily)

Theorem: within $4/3$ of OPT for “difficult” pointsets

In practice: solution cost is within 0.5% of OPT on average


Adapted from Prof. G. Robins, UVA
Euclidean TSP: Recall the 2-Approximation

DFS traversal of MST

Shortcutting of the DFS tour (e.g., replacing a-b-c-b-a-d, by a-b-c-d)

\[ \text{Tour}_{\text{Heur}} \leq 2 \times \text{MST} \leq 2 \times \text{Tour}_{\text{opt}} \]
Now: A 3/2-Approximation for Euclidean TSP

• Improving the conversion from the tree traversal into a TSP tour: (Christofides 1976)
  – New way to look at previous conversion: we build an Eulerian circuit on top of the tree, by doubling each edge. Then we obtain the TSP tour by taking shortcuts from the Eulerian circuit.
  – Intuition: Tour_Heur has less cost than the cost of the Eulerian graph. If we can start with a lower-cost Eulerian graph, we will get a better bound.

⇒ What is a cheaper augmentation of the MST, such that the resulting graph is Eulerian?
3/2 Approximation for Euclidean TSP

• Key property of Eulerian graph: every node has even degree

• Basic property of the MST (or any graph): there is an even number of odd-degree nodes
  – Handshake Lemma: Sum of node degrees = 2 * # of edges
  – Idea: add edges to MST to make it Eulerian: +1 edge for each odd-degree node in the MST

• Specifics
  – Find a minimum-cost matching among the odd-degree vertices of the MST
  – Add an edge between every matched pair
  – Result == an Eulerian graph, which we can traverse and shortcut exactly as we did with the doubled MST
3/2 Approximation for Euclidean TSP

MST plus matching: red line = Euler circuit

TSP tour obtained by shortcutting Euler circuit

Previous 2-approximation
3/2 Approximation for Euclidean TSP

- Consider optimal TSP tour. Nodes marked by triangles are odd-degree nodes in MST. The solid line represents the opt TSP tour. The red dashed lines and blue dashed lines represent two possible matchings among those odd-degree nodes.

Either total length of blue lines or total length of red lines is \( \leq 0.5 \times \text{Tour}_{\text{opt}} \)

The \textbf{minimum} matching cost is \( \leq \) either the red or blue matching cost \( \Rightarrow \) If we use Min Matching, total edge cost of the Euler circuit (before any shortcuts) will be \( \leq 1.5 \times \text{Tour}_{\text{opt}} \)
List of Provably Good Heuristics

- 1-Steiner Tree (4/3)
- Bounded-Radius-Bounded-Cost Tree
  \((1 + \varepsilon \text{ radius}, 1 + 2/\varepsilon \text{ cost})\)
- Engineer’s Method for Number Partitioning (7/6)
- Next-Fit Bin Packing (2)
- Christofides’ Euclidean TSP (3/2)
- Greedy “farthest-first” k-center (2)