Advanced Search

- Applications:
  - Combinatorial Optimization
  - Scheduling
- Algorithms:
  - Stochastic Local Search and others
- Analyses:
  - Phase transitions, structural analysis, statistical models

Combinatorial Problems

- Applications:
  - TSP, SAT, planning, scheduling, packet routing, protein structure prediction, …
- Informally:
  - find a grouping, ordering or assignment of a discrete, finite set of objects that satisfies given conditions (from Hoos and Stützle)

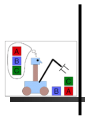
Decision vs. Optimization Problems

- Decision:
  - Solutions satisfy given logical conditions
  - For a given problem instance:
    - *Does a feasible solution exist?* [Decision variant]
    - *What is a feasible solution?* [Search variant]
- Optimization:
  - In addition to satisfying logical conditions, add an objective function, f, that quantifies the quality of the solution found.
  - "Best" can minimize or maximize f
  - For a given problem instance:
    - *Does a solution exist with f at least as good as some bound?* [Decision variant]
    - *What is a solution with optimal f?* [Search variant]
    - *What is the optimal value of f?* [Evaluation variant]

[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]

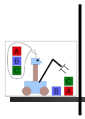
Travelling Salesperson Problem

- Given: Directed, edge-weighted graph G
- Objective: Find a minimal-weight Hamiltonian cycle in G



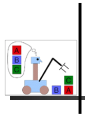
Search Approach Dimensions

- How to form solutions
  - Perturbation: start with complete solutions and modify them
  - Construction: start with null solution and add to it until complete
- How to traverse the search space
  - Systematic: guarantee completeness
  - Local Search: traversal based on information in current state
- How many points to explore (Population based)



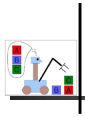
Search Algorithms from CS440

	Systematic	Local
Perturbative		
Constructive		



Issues

- Completeness: given enough time, all points in search space are explored
- Any-time property: a solution is always available, but additional time yields better solution
- Hybridization/Metaheuristics/Portfolios: combine different strategies
- Stochasticity: ubiquitous, how and when to best add it?



When to use which algorithms?

- Truthfully… more of an art than a science.
- Some guidelines:
  - Systematic when solution quality guarantees are important and problem knowledge is effective, but time is not critical. Or for decision problems (e.g., Satisifiable?)
  - Local when need solution quickly, little useful knowledge can be exploited and search space is huge.
  - Portfolios and hybrid algorithms try to balance pros/ cons.

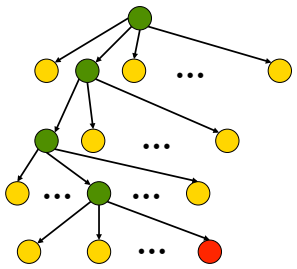
Classic "Strawmen" for Comparison

- Greedy
  - follow a domain heuristic to construct a solution
- Iterative Sampling (Langley 1992)
  - make random decisions in constructive search
  - when reach dead end, re-start

Search Algorithms for CS540

- Systematic Search
  - Randomized Choices
- Local Search
  - Randomized Iterative Improvement, Metropolis Sampling, Simulated Annealing, Tabu Search, Iterated Tabu Search, Dynamic Local Search, Variable Neighborhood Search
- Hybrid
  - GRASP, Squeaky Wheel Optimization
- *Population-Based (Evolutionary Computation)*
  - *Explore multiple points simultaneously*
  - *Ant Colony Optimization, Genetic Algorithms*

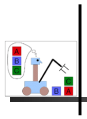
Constructive + Backtracking/ Restarts = Systematic



- Search is process of iteratively constructing a solution.
- Modeled as a tree in which each node is decision about what to add next based on heuristics.
- What happens when decisions do not lead to acceptable solution? *Backtracking* or *Restarts*

Backtracking

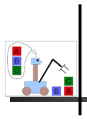
- Chronological:
  - Order states to be explored according to some notion of when encountered (e.g., depth, breadth, discrepancy)
- Knowledge Directed:
  - Application specific heuristics (e.g., min-conflicts, unit propagation)



Randomized Systematic Search

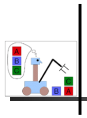
Randomization + Restarts [Gomes, Selman & Kautz AAAI1998]

- Start with complete, constructive search algorithm
- At choice points, select randomly from *heuristically equivalent* options (those with value within H% of best)
- *Cutoff* parameter limits search to a set number of backtracks which triggers restart
- Add bookkeeping to preclude repetition
- Significant improvement over base algorithms in three different applications



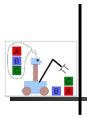
Observations about RSS

- Distribution of search costs across trials tends to be "heavy tailed": some outliers with very high search costs.
  - Hypothesis is that these outliers are caused by poor choices early on.
  - … which makes restarts important.
- Benefits: more robust, easy to parallelize
- Costs: hard to analyze and ensure completeness, more parameters



Local Search

- Start with complete solution and iteratively improve it
- aka hill climbing or gradient search
- chief variants: next or steepest descent
- usually Markovian
- requires fast evaluation function, some initialization, neighborhood definition but no domain knowledge*!



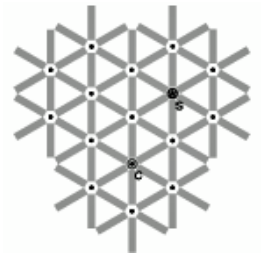
Local Search Definition

For a given problem instance π:

- Search space $S(\pi)$
- Solution set $S'(\pi) \subseteq S(\pi)$
- Neighborhood relation $N(\pi) \subseteq S(\pi) \times S(\pi)$
- Set of memory states $M(\pi)$
- Initialization function $\varnothing \rightarrow \mathcal{D}(S(\pi) \times M(\pi))$
- Step function $S(\pi) \times M(\pi) \rightarrow \mathcal{D}(S(\pi) \times M(\pi))$
- Termination predicate $S(\pi) \times M(\pi) \rightarrow \mathcal{D}(\{\perp, T\})$

[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]

Stochastic Local Search - Global View



- Vertices: solutions/states
- Edges: connect neighbors
- s: solution
- c: current state

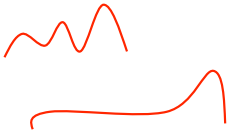
[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]

Optimization

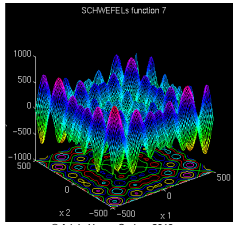
- Objective Function
  - Actual value of solutions to problem instance
- Evaluation function
  - maps candidate solutions onto real numbers
  - Used to guide search in SLS
  - May not be same as objective function, sometimes more efficient version

Local Search Pathologies

- Local minimum: state without improving neighbors
- Plateau: state with neighbors of equal value



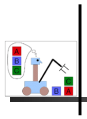
- Non-globally optimal basins of attraction



Search Strategies in LS

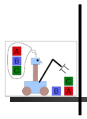
Search strategy: specified by

1. Initialization
   - Random
   - Greedy (e.g., steepest descent LS)
   - Heuristic (e.g., approximate backbone in SAT)



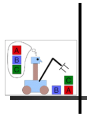
Search Strategies in LS (cont.)

2. Step function/Perturbation phase
   - can be multiple phases
   - Carefully applied randomness (e.g., random walk on plateaus)
   - Escape from local minima
     - Restarts
     - Accept non-improving moves
   - Modify neighborhood: larger neighborhood has fewer local minima, but may take longer to chose a move



Intensification vs. Diversification

- Intensification:
  - greedy improvement
  - Steepest descent (aka best improvement)
- Diversification:
  - encourage exploration
  - Randomized next descent (aka first improvement)
- Issues: efficiency, solution quality

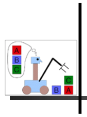


Randomized Iterative Improvement

With some fixed probability, choose randomly from neighborhood instead of choosing improving move.

1. Determine initial candidate solution s
2. While termination condition is not satisfied:
   1. With probability wp:
      - choose a neighbor s' of s uniformly at random
   2. Otherwise:
      - choose a neighbor s' of s such that $g(s') < g(s)$ or, if no such s' exists, choose s' such that $g(s')$ is minimal
   3. $s := s'$

[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]

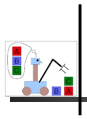


Metropolis Sampling

Metropolis, Rosenbluth, Teller & Teller in 1953 developed a Monte Carlo sampling method for simulating equilibrium of atoms.

Algorithm as adapted for combinatorial optimization:

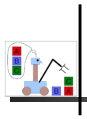
1. Starting from some solution
2. Loop
   1. perturb solution and update counter
   2. If better than previous solution, store it and reset counter
   3. else
      1. if counter exceeds threshold, stop
      2. else with probability $e^{-(\Delta f)/T}$ return to previous solution and reset counter; otherwise increment counter



Simulated Annealing [SA]

Kirkpatrick, Gelatt & Vecchi 1983

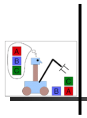
- High values of T lead to more variability and may expedite escapes from local optima.
- SA= Metropolis Sampling plus a means of setting "*T*" (temperature)
  - define a schedule to gradually decrease *T*
  - e.g., use exponentially decreasing function such as Y_1=10, Y_i=0.9*Y_{i-1}



Simulated Annealing Algorithm

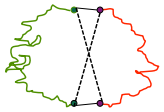
1. Determine initial candidate solution s
2. Set initial temperature T according to annealing schedule
3. While termination condition is not satisfied:
   1. probabilistically choose a neighbor s' of s using proposal mechanism
   2. If s' satisfies probabilistic acceptance criterion (depending on T): s := s'
   3. update T according to annealing schedule

[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]



SA Example: TSP

Neighborhood: 2-exchange



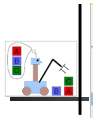
Proposal mechanism: uniform random choice

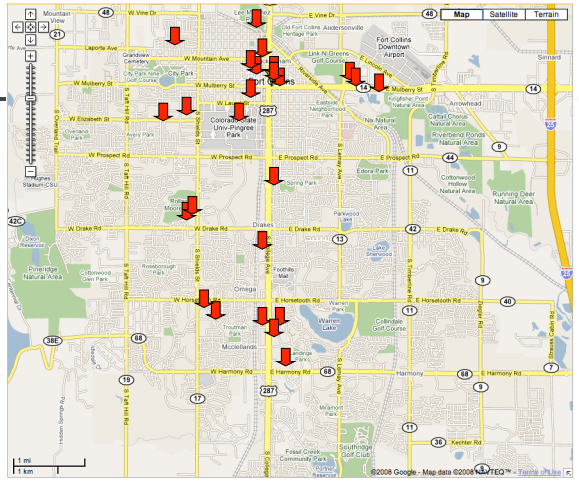
Acceptance criterion: Metropolis condition

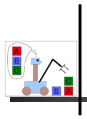
Annealing schedule: geometric cooling T=0.95*T, threshold on counter = N*(N-1) where N is number of vertices in graph

Termination: no improvement in 5 successive steps and fewer than 2% of proposed steps are accepted

[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]

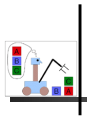






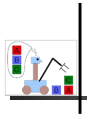
Tabu Search

- Glover 1986
- Add *memory* to local search to prevent re-visiting states
  - prevent cycles
  - increase diversity of exploration
  - encourage escape from local optima
- Meta-heuristic framework



Basic Tabu Search Algorithm

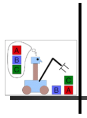
1. initialize solution s and *best-so-far*
2. While termination criteria are not satisfied:
   1. update memory (add last solution and refresh)
   2. generate neighbor solutions
   3. prune neighbors that are tabu (in memory already)
   4. Set s to best of remaining neighbors
   5. update *best-so-far*, if necessary
3. return *best-so-far*



Tabu Search Parameters

- Size of memory or time to be tabu (tabu tenure)
- Contents of memory
  - solution based
  - attribute based: e.g., changes to solution features
- Long term memory?
  - aspiration level: accept a tabu move that improves over recent fitness levels
  - longer cycle detection

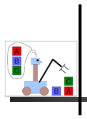
Efficient implementation requires caching evaluations & special data structures for tabu list



Tabu Search Variants

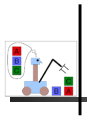
- Robust Tabu Search [Taillard 1991]
  - Repeatedly choose tabu tenure randomly from an interval
  - Force moves if not done in set large number of steps
- Reactive Tabu Search [Battiti & Tecchiolli 1994]
  - Dynamically adjust tabu tenure during search
  - Force series of random changes to escape a region

[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]



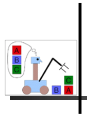
Tabu Search Applications

- State of the art or close:
  - MAX-SAT, CSPs, scheduling
- Implementations often include tweaks:
  - Well designed neighborhoods
  - Improved solution evaluation
  - Restarts at elite solutions
  - Manipulate solution components: freeze some, force changes periodically



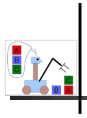
Dynamic Local Search

- Modify the evaluation function to bias search trajectory
- Usually done through modifying weights on components of evaluation function or of solution (penalties)
- Trigger modification when in local optimum or boundary region



TSP Game

- http://www.tsp.gatech.edu/games/tspOnePlayer.html



Dynamic Local Search Algorithm

1. Determine initial candidate solution s
2. Initialize penalties
3. While termination criterion is not satisfied:
   1. compute modified evaluation function g' from g based on penalties
   2. perform subsidiary local search on s using evaluation function g'
   3. update penalties based on s

[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]

Dynamic Local Search Parameters

- Modified evaluation function:

$$g'(\pi',s) := g(\pi,s) + \sum_{i \in SC(\pi',s)} penalty(i)$$

where SC(π',s) = set of solution components of problem instance π' used in candidate solution s

- Penalty initialization: For all i: penalty(i) := 0
- Penalty update in local minimum s: Typically involves penalty increase of some or all solution components of s; often also occasional penalty decrease or penalty smoothing
- Subsidiary local search: Often *Iterative Improvement*

[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]

DLS Example: TSP

Guided Local Search

[Voudouris & Tsang 1999]

Search Space: Hamiltonian cycles in graph G with n vertices; 2-exchange neighborhood; solution components = edges of G

Penalty initialization: Set all to 0

Subsidiary local search: Iterative first improvement

Penalty update: Increment for all component edges by

$$\lambda = 0.3 * \frac{w(s_{2-opt})}{n}$$

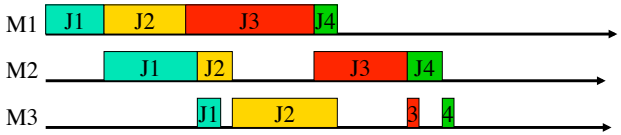
s_{2-opt} = **2 - optimal tour**

[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]

Example: Scheduling Permutation Flow Shop

	M1	M2	M3
Job	Dur	Dur	Dur
1	5	8	2
2	7	3	9
3	10	7	1
4	2	3	1

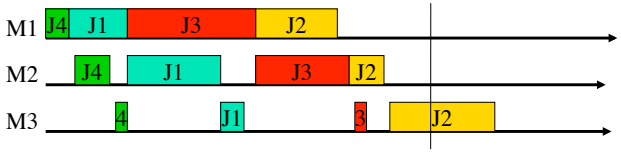
Solution: 1,2,3,4

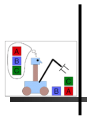


Example Permutation FSP

	M1	M2	M3
Job	Dur	Dur	Dur
1	5	8	2
2	7	3	9
3	10	7	1
4	2	3	1

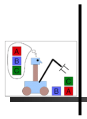
Solution: 4,1,3,2





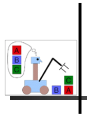
Variable Neighborhood Search (VNS) [Mladenovic and Hansen 1997]

- Intuition: A point is locally optimal wrt a neighborhood. So change the neighborhood to escape the local optimum.
- Use k neighborhood relations
- Algorithm:
  1. Input: problem + set of neighborhoods K_i sorted in increasing size
  2. Repeat until largest neighborhood tried
     1. Find solution in neighborhood Ni
     2. If better solution than best so far, then i=1; else i=i+1



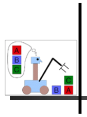
VNS example: TSP

- Neighborhoods:
  - Based on GENIUS heuristic:
    - pick city for deletion/insertion, consider *p* nearest cities already in tour as connection point. Vary *p*.
  - Based on 2-opt:
    - Use 1-opt, 2-opt, 2.5-opt and Lin-Kernighan heuristics



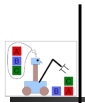
Hybrid Methods

- Combine all or parts of algorithms to yield improvements in some applications
- Examples
  - Initialize one algorithm with solutions from another
  - Restart with Uninformed Random Picking or some other algorithm



Iterated Local Search

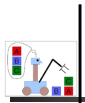
- Two main steps:
  - Subsidiary local search for reaching local optima (intensification)
  - Perturbation for escaping from local optima (diversification)
- Switch between based on acceptance cirterion



ILS Algorithm

1. Determine initial candidate solution *s*
2. Perform subsidiary local search on *s*
3. While termination criterion is not satisfied:
   1. *r* := *s*
   2. Perturb *s*
   3. Perform subsidiary local search on *s*
   4. If acceptance criterion is true, keep *s*; otherwise *s* := *r*

[adapted from slides for Hoos & Stützle's "Stochastic Local Search"]

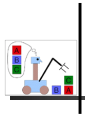


ILS Parameters

Subsidiary Local Search: e.g., Tabu search, Lin-Kernighan for TSP

Perturbation: move away from current local optimum, e.g., use larger neighborhood, adaptively change perturbation

Acceptance criteria: balance between accepting better solution vs. more recent solution

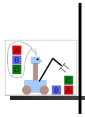


Greedy Randomized Adaptive Search Procedures (GRASP)

[Feo & Resende 1995]

Intuition: Heuristic may only be good enough to create a solution near optimal; may need to make minor improvements to initial solution.

- Hybrid meta-heuristic combining constructive and local search
- like *iterative repair* schemes



GRASP Framework

For *n* iterations:

Greedy Randomized Adaptive construction:

1. incrementally add elements to solution that preserve feasibility and that are randomly selected from those within $1-\alpha$ of best cost (Restricted Candidate List)
2. adaptively evaluate element costs based on current solution

Local Search

1. until no improving neighbor found
   1. set state to improving neighbor

Return best locally optimal solution found

GRASP Parameters

- number of iterations
- constructive heuristic
- cost evaluation
- composition of restricted candidate list (α)
- neighborhood (size and structure)
- steepest or next descent

GRASP Extensions

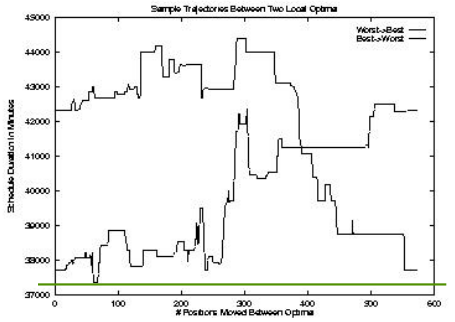
Issues: parameters must be selected, stateless across iterations

- Reactive GRASP
  - periodically modify α based on observed solution qualities – tend to select α that has led to higher mean solution quality
  - tended to find more optimal solutions at cost of additional computation

Path-Relinking [Glover 1996]

- meta-heuristic that exploits information from multiple solutions
- search trajectories between two elite solutions
  - use solutions ala GAs, picking attributes of each
  - perform local search from intermediate points between two solutions
  - Population based search

Path Relinking Example



Adding Path-Relinking to GRASP

- method of introducing memory
  - intensification strategy to optimize local solutions
    - apply to local optimum and randomly selected other elite
  - final phase to optimize all pairs of elite solutions

Adaptive Iterated Construction Search

- Like GRASP, alternate construction and perturbation
- Unlike GRASP, associate weights with decisions/components; weights computed based on quality and components of current solution
- Balance between time for construction and time for local search

Squeaky Wheel Optimization (SWO) [Joslin & Clements 98]

- Constructive, heuristic guided search that makes large moves in search space
- Algorithm:
  1. Construct a greedy initial solution
  2. Analyze to find the "squeaky wheels" (tasks that contribute most to poor solution quality), assigning each an amount of blame
  3. Prioritize to increase the values of squeaky wheels so that they are moved forward during greedy construction

Other Hybridization Methods

- Portfolios/Ensembles
  - Collect *n* different algorithms
  - Extract features from problem instance (on-line versus off-line)
  - Create strategy for selecting between them predicated on problem instance features
  - Apply selected algorithm
  - Iterate if can change selection or use on-line features