

# Lecture 5: Local Search

*10:30 AM, Feb 5, 2009*

## Contents

<b>1 Overview</b>	<b>1</b>
<b>2 Definition and Example</b>	<b>2</b>
2.1 TSP . . . . .	2
2.2 TSP as Search . . . . .	3
2.2.1 Hamiltonian Paths . . . . .	3
2.2.2 Hamiltonian Circuits . . . . .	4
2.3 TSP as Combinatorial Optimization . . . . .	4
<b>3 Hill-Climbing</b>	<b>5</b>
3.1 Best-Improvement Search . . . . .	5
3.2 First-Improvement Search . . . . .	6
<b>4 Simulated Annealing</b>	<b>7</b>
<b>5 Local Beam Search</b>	<b>9</b>
<b>6 Genetic Algorithms</b>	<b>9</b>

## 1 Overview

The topic of this lecture is local search methods for combinatorial (i.e., discrete) optimization—the problem of finding a globally optimal state in very large search spaces. Such optimization problems pervade many fields, including engineering, the natural sciences, and management:

- electrical engineering: integrated circuit design
- biology and chemistry: gene sequencing, protein folding
- management science: inventory planning, procurement of raw materials, manufacturing and distribution of finished products

Informally, the following features characterize combinatorial optimization problems:

- combinatorial (i.e., very large) state space

- cost (value) function to be minimized (maximized)

When these additional features hold, local search is often the solution technique of choice:

- similar solutions have similar costs (“continuous” objective function)
- exhaustive, blind, and heuristic search are intractable
- reasonable solutions are satisfactory

In contrast to typical search problems, the solution to a combinatorial optimization problem does not usually include the path to a goal, only the goal itself. Instead, every state represents a complete solution to the problem, although not necessarily an optimal one. This observation suggests the use of **iterative improvement** algorithms to solve such optimization problems: initialize the algorithm at some (possibly random) state, and iteratively perturb the current solution, updating it if any of the tested perturbations yields an improvement.

Local search algorithms—the focus of this lecture—seek iterative improvements in the local neighborhood of the current solution. We study: (i) hill-climbing algorithms, which greedily improve upon the current solution; (ii) simulated annealing, which temporarily explores changes worse than the current solution, but ultimately only exploits improvements in the current solution; (iii) local beam search, which combines aspects of hill-climbing and simulated annealing, and (iv) genetic algorithms, which can be viewed as a form of parallel local search.

## 2 Definition and Example

An **optimization problem** consists of a state space  $X$  and an objective function  $\text{Obj} : X \rightarrow \mathbb{R}$ , where a (goal) state  $x^*$  is either a (global) minimum or a maximum (*i.e.*, it is an optimum). Given an optimization problem,  $x^* \in X$  is a minimum iff  $\text{Obj}(x^*) \leq \text{Obj}(x)$ , for all  $x \in X$ ; similarly,  $x^* \in X$  is a maximum iff  $\text{Obj}(x^*) \geq \text{Obj}(x)$ , for all  $x \in X$ . This lecture is tailored toward minimization problems and algorithms; maximization problems are handled analogously.

A **local search problem** is an optimization problem together with a **neighborhood** structure. The function  $N : X \rightarrow 2^X$  returns the neighborhood of state  $x$ , which typically consists of simple perturbations of  $x$  for which  $\text{Obj}(x)$  can be computed efficiently, perhaps incrementally. The state  $x \in X$  is a local minimum iff  $\text{Obj}(x) \leq \text{Obj}(y)$ , for all  $y \in N(x)$ . Local search algorithms, which seek to optimize in neighborhoods (*i.e.*, locally), return local optima.

### 2.1 TSP

Figure 1 presents an instance of the traveling salesperson problem (TSP), a classic NP-hard<sup>1</sup> combinatorial optimization problem. Providence, Boston, Hartford, Washington D.C., and New York City and their respective distances (as the crow flies) are depicted. A path through the graph that visits all cities *exactly once* and returns to its origin is called a **tour**. The objective in TSP is to find a tour of minimal distance. The optimal tour in Figure 1 is PBHWNP of distance 795.

---

<sup>1</sup>An NP-hard problem is at least as hard as the problems in the complexity class known as NP. NP stands for nondeterministic polynomial time. No (deterministic) polynomial time solution is known for any of the problems in this class, or for any NP-hard problems as a result.

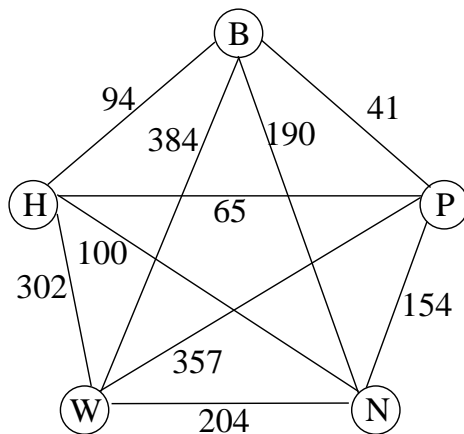


Figure 1: Traveling Salesperson Problem.

Before formulating TSP as a local search problem (states are tours), we formulate it as a typical search problem in which tours are constructed incrementally. Tours constructed via search are often used to initialize iterative improvement methods.

## 2.2 TSP as Search

Given a set of cities  $Y$  of cardinality  $N$ , and given the corresponding distances between cities ( $d(x, y)$ , for all  $x, y \in Y$ ), we formulate TSP as a search problem in two ways. In the first formulation, states are **Hamiltonian paths**: paths that do not visit any city twice. In the second formulation, states are **Hamiltonian circuits**: cycles that do not visit any city twice.

### 2.2.1 Hamiltonian Paths

Let the set of states  $X$  consists of all Hamiltonian paths: *i.e.*, paths that do not visit any city twice. Formally,

$$X = \{(x_1, \dots, x_n) \mid n = 1, \dots, N + 1, x_i \in Y \text{ for all } 1 \leq i \leq n, \\ x_i \neq x_j \text{ for all } 1 \leq i \neq j \leq n, \text{ unless } i = 1, j = N + 1\}$$

The set of goal states includes all states of length  $N + 1$ . The set of start states includes all states of length 1. The transition function between states is defined as follows: at state  $(x_1, \dots, x_n)$ ,

$$\delta(x_1, \dots, x_n) = \begin{cases} \{(x_1, \dots, x_n, x_{n+1}) \mid x_{n+1} \neq x_i \in Y, \text{ for all } 1 \leq i \leq n, \}, & \text{if } n < N \\ \{(x_1, \dots, x_n, x_1) \mid & \text{if } n = N \end{cases}$$

Figure 2 depicts the first three levels of the search space in this formulation for the traveling salesperson problem depicted in Figure 1.

One simple heuristic to solve TSP in this formulation is “nearest-neighbor.” Beginning at city  $x$ , the salesperson visits city  $y$  *s.t.*  $d(x, y)$  is minimal; from city  $y$ , s/he visits city  $z$  *s.t.*  $d(y, z)$  is minimal, *so long as doing so does not create a cycle of length less than  $N$* , in which case s/he visits a city of the next least distance; and so on. Applying this heuristic to the search problem in Figure 2 generates the suboptimal path PBHNWP of distance 796.

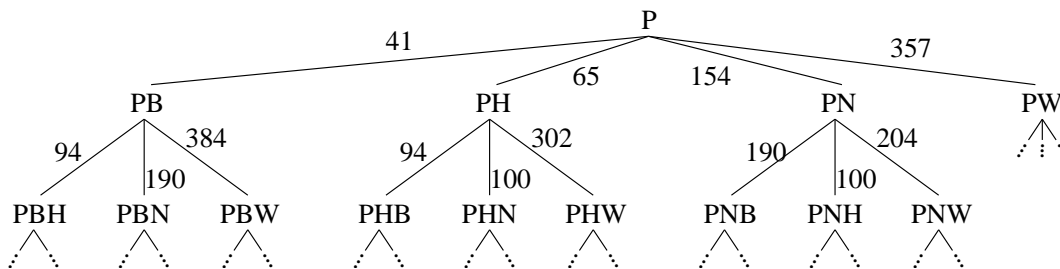


Figure 2: Search in TSP: States are Hamiltonian Paths.

### 2.2.2 Hamiltonian Circuits

Let the set of states  $X$  consists of all Hamiltonian circuits: *i.e.*, cycles that do not visit any city twice. Formally,

$$X = \{(x, y_1, \dots, y_n, x) \mid n = 1, \dots, N - 1, x \neq y_i \neq y_j \in Y \text{ for all } 1 \leq i \neq j \leq n\}$$

In this case, the transition function between states is defined as follows: at state  $(x, y_1, \dots, y_i, y_{i+1}, \dots, y_n, x)$ ,

$$\delta(x, y_1, \dots, y_n, x) = \{(x, y_1, \dots, y_i, y, y_{i+1}, \dots, y_n, x) \mid y \neq y_i \neq x \in Y \text{ for all } 1 \leq i \leq n\}$$

at cost  $d(y_i, y) + d(y, y_{i+1}) - d(y_i, y_{i+1})$ . The set of goal states includes all states of length  $N + 1$ . The set of start states includes all states of length 2.

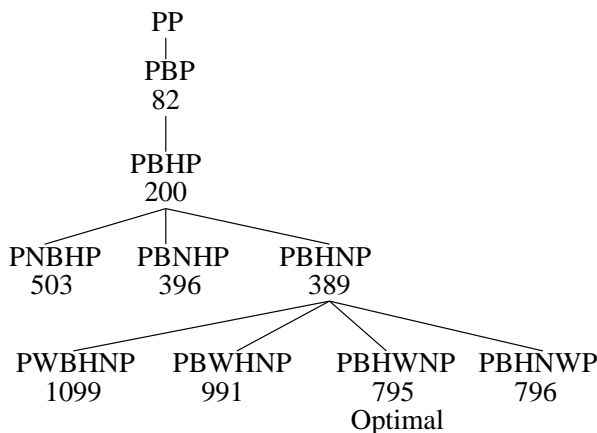


Figure 3: Search in TSP: States are Hamiltonian Circuits.

A greedy heuristic for solving TSP in this formulation is simply: insert a city into the tour that causes the smallest increase in the length of the tour. This heuristic finds an optimal tour in our example. (See Figure 3.)

### 2.3 TSP as Combinatorial Optimization

TSP can also be formulated as a local search problem, by defining states as tours. There are many possible operations for computing local neighborhoods. One example is inversion: given a tour,

consider swapping the order of visiting each consecutive pair of cities on the tour (except the origin). More generally, an operation called 2OPT eliminates any two edges, say  $(x_1, x_2)$  and  $(y_1, y_2)$ , and reconnects the nodes in the opposite way, as  $(x_1, y_2)$  and  $(y_1, x_2)$ . Inversion and 2OPT are useful in practice because their computation is cheap—just subtract the distances of the deleted edges and add the distances of the new edges.

How might a local search algorithm behave on our sample TSP? Suppose the initial state is PWBHNP of distance 1099. Applying inversion yields PBWHNP of distance 991, PWHBNP of distance 1097, and PWBNHP, of distance 1106. A local search algorithm would accept PBWHNP, since  $991 < 1099$ ; but, it could also reasonably accept PNBHWP since  $1097 < 1099$ . Assume PBWHNP is accepted. Applying inversion again yields PWBHNP (the original tour), PBHWNP of distance 795, and PBWNHP of distance 804. Now, if the tour PBHWNP is accepted, the algorithm would verify that it encountered a local (in fact, global) minimum and halt. (See Figure 4.)

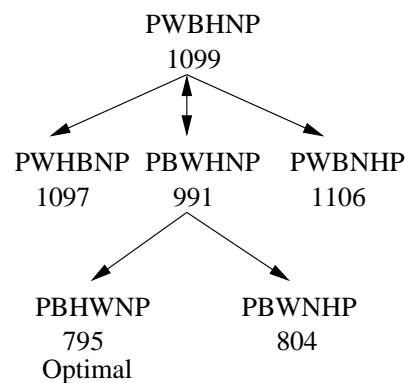


Figure 4: Local Search in TSP.

### 3 Hill-Climbing

Hill-climbing is a family of local search algorithms that consider changes in the local neighborhood of a given state and accept changes that improve upon the current solution. Best-improvement and first-improvement are examples of hill-climbing algorithms. Other variants include GSAT and WALKSAT, which are special purpose algorithms for solving satisfiability (see Lecture #7).

#### 3.1 Best-Improvement Search

The main idea behind **best-improvement** search is to consider *all* states in the local neighborhood of the current state, and to accept one that best improves the objective function. Best-improvement search terminates at a local (but not necessarily global) minimum. Thus, the performance of best-improvement search is rarely globally optimal in landscapes characterized by foothills.

BEST( $X, \text{Obj}, N, x$ )	
Inputs	local search problem random start state $x$
Output	best state visited $x$
Initialize	$x' \neq x$
<b>while</b> ( $x' \neq x$ ) <b>do</b>	
1.	$x' = x, A = \{x'\}$
2.	for all $y \in N(x')$
	(a) if $\text{Obj}(y) < \text{Obj}(x), x = y, A = \{x\}$
	(b) else if $\text{Obj}(y) = \text{Obj}(x)$ , insert $y$ in $A$
3.	choose $x$ in $A$
<b>return</b> $x$	

Table 1: Best-Improvement Search.

### 3.2 First-Improvement Search

Unlike best-improvement search, **first-improvement** search does not consider all neighbors of the current solution. On the contrary, it considers its neighbors at random and accepts the first one that demonstrates an improvement. It halts if ever it loses its patience before finding an improvement. First-improvement search is also called **stochastic local search** and **randomized hill-climbing**.

FIRST( $X, \text{Obj}, N, p, x, \epsilon$ )	
Inputs	local search problem patience time limit $p$ random start state $x$ rate of exploration $\epsilon$
Output	best state visited $x$
Initialize	$t = 0$
<b>while</b> ( $t < p$ ) <b>do</b>	
1.	for some $y \in N(x)$
	(a) if $\text{Obj}(y) < \text{Obj}(x), x = y, t = 0$
	(b) else if $\text{Obj}(y) = \text{Obj}(x)$ and $\text{rand}[0, 1] \leq \epsilon, x = y, t = 0$
	(c) else increment $t$
<b>return</b> $x$	

Table 2: First-Improvement Search.

Given infinite patience, stochastic local search terminates at a local optimum with probability 1. But rather than run the algorithm with too much patience, it is usually run repeatedly with random-restarts. Given infinite patience and an infinite number of random restarts, stochastic local

search terminates at a *global* optimum with probability 1.

RANDOM( $X, \text{Obj}, N, p, n, \epsilon$ )	
Inputs	local search problem patience time limit $p$ number of restarts $n$ rate of exploration $\epsilon$
Output	best state visited $x^*$
Initialize	$\text{Obj}(x^*) = \infty$
for $i = 1$ to $n$	
	1. choose start state $x \in X$
	2. $z = \text{FIRST}(X, \text{Obj}, N, p, x, \epsilon)$
	3. if $\text{Obj}(z) < \text{Obj}(x^*)$ , $x^* = z$
return $x^*$	

Table 3: Stochastic Local Search with Random-Restarts.

## 4 Simulated Annealing

Simulated annealing generalizes stochastic local search. With some probability, say  $p$ , simulated annealing accepts state changes that adversely affect the value of the objective function. In other words, in accordance with  $p$ , it encourages **exploration** of harmful states; otherwise, it **exploits** successful states.

A probability  $p$  of exploration can be determined in several ways. One approach is simply to fix  $p$  at some small value  $\epsilon > 0$ . Another option is to let  $p$  decrease with time:  $p \sim 1/t$ . Yet another idea is to let  $p$  decrease as  $\Delta(x, y) = \text{Obj}(y) - \text{Obj}(x)$  increases, where  $x$  is the current solution and  $y \in N(x)$ .

Updating in the spirit of this third idea can be achieved as follows: if  $\Delta(x, y) < 0$  (i.e.,  $y$  is an improvement), then let  $p = 1$ : i.e., exploit; if  $\Delta(x, y) \geq 0$  (i.e.,  $y$  is not an improvement), let  $p = e^{-\Delta(x, y)}$ . In this way, small increases in the value of the objective function imply large values of  $p$  (exploration is likely; exploitation is not), whereas large increases in the value of the objective function imply small values of  $p$  (exploitation is likely; exploration is not). (As presented, note that all ties are broken in favor of  $y$ , since  $e^0 = 1$ .)

Simulated annealing is a local search algorithm based on a combination of ideas: (i) exploration decreases with time and (ii) exploration is more likely if it makes things only slightly worse, and less likely if it makes things significantly worse. Specifically, simulated annealing relies on the following exploration probabilities: for  $T > 0$ ,

$$p = \begin{cases} e^{-\Delta(x, y)/T} & \text{if } \Delta(x, y) \geq 0 \\ 1 & \text{otherwise} \end{cases}$$

Simulated annealing derives its name from the interpretation of  $T$  as a temperature, which through slow cooling strengthens metal during its production. Thus, exploration is more likely than ex-

SA( $X, \text{Obj}, N, m, x, C$ )	
Inputs	local search problem number of iterations $m$ random start state $x$ cooling schedule $C$
Output	best state visited $x^*$
Initialize	$x^* = x, T$
<b>for</b> $i = 1$ <b>to</b> $m$	
1. <b>for some</b> $y \in N(x)$	
(a) compute $\Delta(x, y) = \text{Obj}(y) - \text{Obj}(x)$	
(b) <b>if</b> $\Delta(x, y) < 0$ , <b>then</b> $x = y$ /* $x$ is an improvement */	
(c) <b>else</b> $\text{rand}[0, 1] \leq e^{-\Delta(x, y)/T}$ , <b>then</b> $x = y$	
(d) <b>if</b> $\text{Obj}(x) < \text{Obj}(x^*)$ , $x^* = x$	
2. decay $T$ according to schedule $C$	
<b>return</b> $x^*$	

Table 4: Simulated Annealing.

exploitation initially, when  $T$  (and hence  $p$ ) is high, but later in the search when  $T$  (and hence  $p$ ) is low, exploitation is more likely than exploration. (See Figure 5.)

The original cooling schedule proposed by Kirkpatrick is to let  $T_{t+1} = T_0 \alpha^{\lfloor t/l \rfloor}$ , for  $T_0 \geq 0$ ,  $\alpha \in (0, 1)$ , and  $1 \leq l \leq m$ , where  $m$  is the number of iterations. For  $T_0 = 10$ ,  $\alpha = 0.99$ ,  $l = 10$ , and  $\Delta(x, y) = 1$ , the cooling schedule and exploration probabilities are shown in Table 5.

$t$	$\lfloor t/l \rfloor$	$\alpha^{\lfloor t/l \rfloor}$	$T_t$	$p_t$
1	0	1	10	0.9
10	1	0.99	9.9	0.9
100	10	0.9	9	0.895
1000	100	0.367	3.67	0.76
10000	1000	$4.32 \times 10^{-5}$	$4.32 \times 10^{-4}$	0.0
100000	10000	0.0	0.0	0.0

Table 5: A Sample Cooling Schedule with Exploration Probabilities

Note that, if  $T_0 = \infty$ , then simulated annealing behaves like a random walk. On the other hand, if  $T_0 = 0$ , then simulated annealing reduces to first-improvement search. Thus, simulated annealing terminates at the local optimum that arises as  $T_i \rightarrow 0$  and  $m \rightarrow \infty$ , with probability 1.

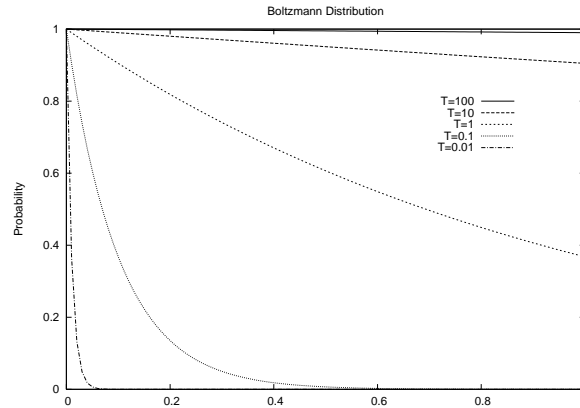


Figure 5: Boltzmann Distribution.

## 5 Local Beam Search

Local beam search is a local search algorithm that operates like best-improvement search, except that it stores a beam of  $w$  improvements at each iteration instead of just 1. More specifically, the beam is initialized to contain  $w$  random states; then, all the successors of each state on the beam are generated; next, the beam is set to contain the best  $w$  states among those on the beam and their successors. This generate and set process continues until no further improvements are found.

**Exercise:** Explain how local beam search differs from running  $w$  parallel best-improvement searches.

In practice, when a search space contains clumps of low-valued nodes concentrated in certain areas, local beam search can reduce to an even more expensive version of best-improvement search, searching essentially the same part of the space  $w$  times instead of just once. An alternative idea, known as stochastic local beam search, again maintains a beam of width  $w$ , but the nodes to be put on the beam are chosen stochastically with probabilities that depend on their values.

## 6 Genetic Algorithms

Genetic algorithms (GAs) are a class of parallel local search algorithms based on the principles of natural selection and survival of the fittest. GAs maintain populations (or generations) of individuals (or chromosomes) made up of genes, in which each individual represents a solution. New populations are bred from old using three operations: **selection** (a form of exploitation), or asexual reproduction, in which individuals survive in proportion to their respective fitness; **crossover** (a form of exploration), which mimics sexual reproduction, in which the genetic material of two individuals is exchanged; and **mutation**, which alters genetic material, thereby preventing the population from stagnating.

A **fitness function** determines the probability with which an individual survives from one generation to the next. The following examples apply to maximization problems: let  $x_0, \dots, x_N$  denote individuals sorted by quality (*i.e.*,  $\text{Obj}(x_i)$ ) from greatest to least:

1. **Standard method:**

$$f_i = \frac{\text{Obj}(x_i)}{\sum_{j=0}^N \text{Obj}(x_j)}$$

2. **Tournament Selection:** Uniformly choose two individuals at random from the population. With probability  $p$  select that individual of greater fitness (break ties randomly).
3. **Rank by Quality:** For fixed probability  $p$ , let  $f_i = p(1 - p)^i$  for an individual of rank  $i$ . In other words, an individual of rank 0 survives with probability  $p$ ; an individual of rank 1 survives with probability  $p(1 - p)$ , and so on.
4. **Rank by Diversity:** Let  $d_i$  denote the distance between individual  $x_i$  and the individual of highest rank, namely  $x_0$ : *e.g.*, Hamming distance. Now diversity rank is determined as follows: the individual with the largest distance is of rank 0 (*i.e.*, that which is farthest from  $x_0$ ), the individual with the next largest distance is of rank 1, and so on.
5. **Rank by both quality and diversity:** For fixed probabilities  $p$  and  $q$ , and weight  $w \in [0, 1]$ , compute fitness as a weighted average of rank and diversity: for individual  $x_i$  of quality rank  $i$  and diversity rank  $j$ , let  $f_i = wp(1 - p)^i + (1 - w)q(1 - q)^j$ .

The generic form of a genetic algorithm is depicted in Table 6.

GA( $X, \text{Obj}, N, n, c, m, f$ )	
Inputs	local search problem number of iterations $n$ crossover probability $c$ mutation probability $m$ fitness function $f$
Output	fit individual
Initialize	population $G$ of $N$ individuals
<ol style="list-style-type: none"> <li>1. <b>for</b> <math>i = 1</math> to <math>n</math> <b>do</b> <ol style="list-style-type: none"> <li>(a) <b>for</b> <math>i = 1</math> to <math>N</math> <b>do</b> <ol style="list-style-type: none"> <li>i. evaluate fitness function <math>f_i</math> for individual <math>x_i</math></li> </ol> </li> <li>(b) <b>for</b> <math>i = 1</math> to <math>\lfloor N/2 \rfloor</math> <b>do</b> <ol style="list-style-type: none"> <li>i. <i>select</i>: generate two parents <math>x</math> and <math>y</math> in <math>G</math> according to <math>f</math></li> <li>ii. <i>crossover</i>: with probability <math>c</math>, swap contiguous bits in <math>x</math> and <math>y</math></li> <li>iii. <i>mutation</i>: with probability <math>m</math>, mutate bits in <math>x</math> and <math>y</math></li> <li>iv. insert offspring of <math>x</math> and <math>y</math> into <math>G'</math></li> </ol> </li> <li>(c) let <math>G = G'</math></li> </ol> </li> <li>2. <b>return</b> individual in <math>G</math> of maximum fitness, or <b>return</b> an individual in <math>G</math> generated at random, according to <math>f</math></li> </ol>	

Table 6: Genetic Algorithm.