Heuristics

Try to find a good solution, not necessarily with any guarantees of optimality.

Greedy Algorithm

Used to construct a suboptimal. With a greedy algorithm, once something is assigned, it cannot be unassigned.

Example

(i) Minimum Spanning Tree:

Order edges weights, so \( e_1, e_2, \ldots, e_n \).

Initialize \( T = \{ e_0 \} \).

For \( i = 2, 3, \ldots, n \):

- If \( T \cup \{ e_i \} \) is a cycle, set \( T \leftarrow T \cup \{ e_i \} \).

Clearly, because adds cheapest available edge at each step.

This alg. actually gives the optimal solution.

(ii) Maximum:

Pick shortest available edge. (Plays in case perfect matching)

\[
\begin{align*}
\text{Maximize:} & \quad \text{pick shortest available edge.} \\
\text{Cuts:} & \quad 1, 1, 1, \text{ etc.}
\end{align*}
\]
Cheapest setup: beams and wires.  
- Star will exist in some other.  
- Find the vertex that can be included in the beam with the smallest sum of total length.  
- Repeat until all vertices included.

**Example:** Euclidean problem.

![Diagram](image)

Have beam through four wires.  
Wires are vertex left.  
Wires to include it?

So, probably best to insert a vertex E between vertices C and D.

**Example:**

![Diagram](image)

Initialization with ABC.

![Diagram](image)

Best move is D.  
D between C and A.

Best move is C.  
C between A and B.

Back now nearest C again be.  
gives a beam of length ≥10cm.

Notice that by ABCDC A has length 12.  So, arbitrarily best.
Local Improvements

Once we have a solution, look for no close by solutions that are better neighbours.

Replace EA and BC by EB and AC:

See except any doubt.

By good positioning.

See SAT-SAT if LSE is right location.
Bad case for 751 2-chains:

Euclidean distance:

But not 3-opt:
Replace $(A, B), (c, D), (C, E)$ by $(A, C), (D, B), (C, E)$.
Simulated Annealing

Probabilistic algorithm. All algorithms we've considered so far have been deterministic, so they always gave the same result when started from the same point. Simulated annealing may give completely different answers even when started from some initial solution.

Simulated annealing is a refinement of local search:
Local searches have a current solution, look in a neighborhood of the solution for a better point. If find a better solution, move to it; otherwise stay. We've found a local optimum.
Problem with local search: may get trapped in bad local optimum.

We are minimizing.

Simulated annealing: allows uphill moves with a certain probability.
Basic algorithm:

1. Get an initial solution $S$
2. Get an initial temperature $T > 0$
3. While not yet frozen, perform the following loop $L$ times
   3.1 Perform the following loop $L$ times
      3.1.1 Pick a random neighbor $S'$ of $S$
      3.1.2 Let $\Delta = \text{cost}(S') - \text{cost}(S)$
      3.1.3 If $\Delta \leq 0$ (downhill move), Set $S = S'$
      3.1.4 If $\Delta > 0$ (uphill move), Set $S = S'$ with probability $e^{-\Delta / T}$
3.2 Set $T = \alpha T$ (reduce temperature)
4. Return $S$

Analogy with physical process of annealing:

Have material in molten state. Cool it to get crystal.
If cool too fast, get widespread irregularities in crystal structure, and trapped energy level is much higher than perfectly structured crystal. So cool in small increments, and at each temperature, let material find its equilibrium.
Need to describe several parameters etc in the algorithm:

(i) What is a neighbour
(ii) How do we pick initial temperature $T$? $T_0 = 0.95$, $L = \text{Gauss distribution of neighbours}$
(iii) What do we mean by "FROZEN"? $G$ along edge 3, 5 times and very few updates and no downhill moves.
(iv) What is a neighbour? Return to this for a particular example.

Algorithm needs a very large number of iterations and a great deal from $n \times 10^3$ to $n \times 10^5$

Alternative: run local search from many different starting points.

Even after equalizing the time used in this manner, simulated annealing is still considerably better than local search.

Need to pick $r$, $L$ and initial temperature to let simulated annealing get good solutions by being run long enough so that it can get good solutions. If have the time, should make several (5, say) runs of simulated annealing, and take the best.
Eg: Good partitioning is (hard comparison for simulated annealing)
good algorithms.

Given a graph $G = (V, E)$ with an even number of nodes,
partition $V$ into two sets $V_1$ and $V_2$ so that $|V_1| = |V_2|$ and the number of edges between $V_1$ and $V_2$ is minimized.

\[ G = \begin{array}{c}
\begin{array}{ccc}
\text{V}_1 & \text{V}_2
\end{array}
\end{array}\]

Have a current partition. What is a neighbor?
Move one vertex from $V_1$ to $V_2$ and one from $V_2$ to $V_1$.
So each solution has $\frac{n^2}{4}$ neighbors.

Consider different construction, which allows the sizes of $V_1$ and $V_2$
to be different.

Use objective function
\[
c(G, V) = \sum (u, v) \in E : u \in V_1, v \in V_2 \cdot f + \alpha (|V_1| - |V_2|)^2.
\]

So realize a partition which is not balanced.
Get a neighbor by moving any vertex from $V_1$ to $V_2$ or vice versa.
So have $n^2$ neighbors.
Why use this alternative?

Extra solutions are allowed — i.e., the unbalanced partitions. This allows extra "escape routes" out of local minima.

For graph partitioning, there is a good heuristic available: Kernighan-Lin.

Have balanced partition, with value $c$.

Label all vertices "unmoved".

Pick the two vertices $u$ in $V_1$ and $v$ in $V_2$ which give greatest decrease / smallest increase in $c$, and swap them.

Repeat until no vertices unmoved.

We now have the same partition back again — everything $u V_1$, but moved to $V_2$ and vice versa. So replace $V_1$ and $V_2$ with the best partition found during the inner loop. If provided this improves on the old $V_1, V_2$.

Repeat.
Comparison vs. Kernighan-Lin:

On random graphs (i.e., for each pair of vertices, the edge exists with probability $p$):

Simulated annealing is better on one run than Kernighan-Lin.

Simulated annealing is far slower, so compare one run of simulating annealing to several runs of Kernighan-Lin from different initial solutions.

Simulated annealing is still slightly better, and the advantage is greater for denser, larger graphs.

On geometrical graphs:

(i.e., generate $p$ vertices in square, all edges of length less than $d$ are included)

Kernighan-Lin is better.

Simulated annealing is improved by starting it from the Kernighan-Lin solutions, rather than from random permutations.