

Heuristics

Try to find a good solution, ~~not~~ usually without any guarantee of optimality.

Greedy Algorithms

Used to construct an initial solution.

With a greedy algorithm, once something is assigned, it cannot be unassigned.

Examples

① Minimum Spanning Tree:

~~Ranking edges~~

Order edges weights, so e_1, e_2, \dots, e_n .

Initiate with $T = \{e_1\}$.

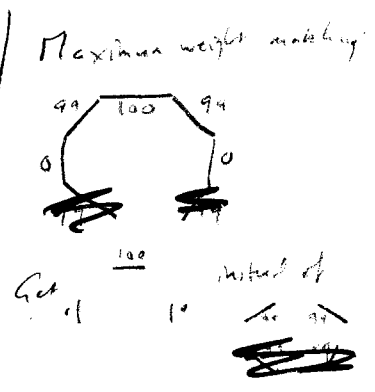
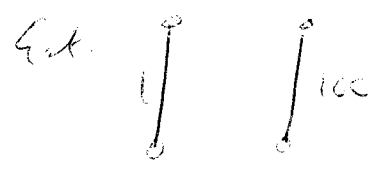
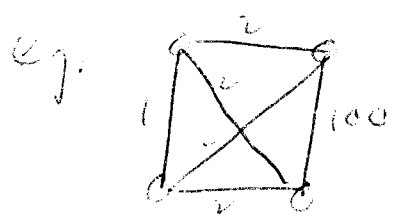
for $i=2, \dots, n$
 if $T \cup \{e_i\}$ is a tree, set $T = T \cup \{e_i\}$.

Greedy, because adds cheapest available edges at each step.

This alg. actually gives the optimal solution.

② Maximum:

pick ~~cheapest~~ largest available edges. (Maximum weight perfect matching)

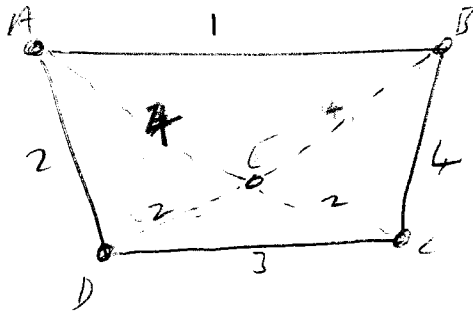


② TSP:

Cheapest insertion heuristic:

- Start with initial tour on three cities.
- Find the ~~a~~ vertex that can be included in the tour with the smallest increase in total length.
- Repeat until all vertices included

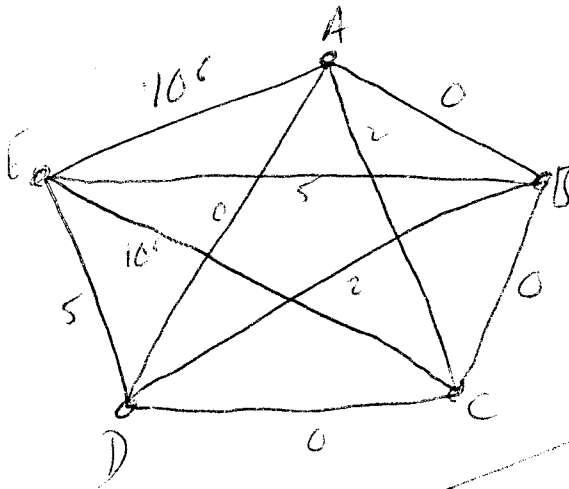
Eg: Euclidean problem



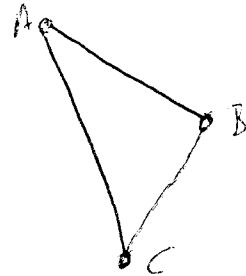
Have tour through four cities.
Have one vertex left.
When to include it?

So probably best to insert ~~the~~ vertex E between vertices C and D.

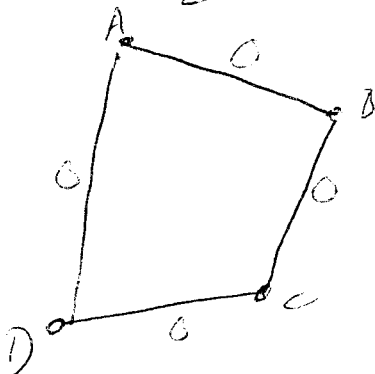
Eg:



Inclusion with ABC



Best insertion is
D between
C and A.



But now inserting E anywhere
gives a tour of length ≥ 10 .

Notice that tour ABEDCA has
length 12. So arbitrarily bad.

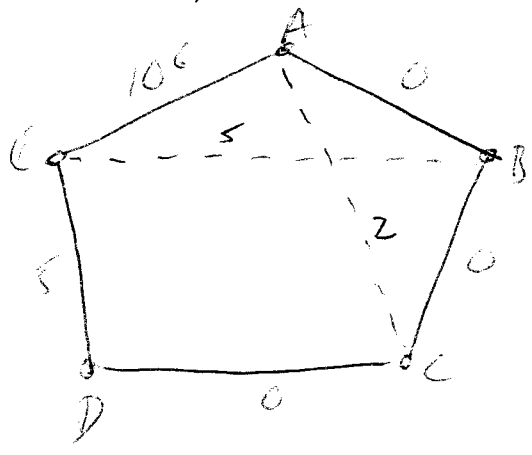
Local Improvement:

Once we have a solution, look for close by solutions that are better.

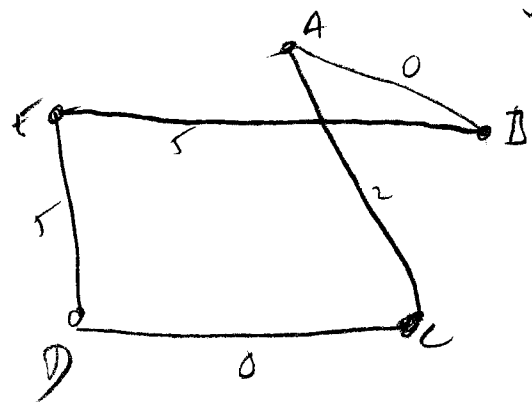
Neighbours.

Defn of this is problem specific.

Eg TSP 2-changes:



Replace EA and BC by EB and AC:



Produce improvement.

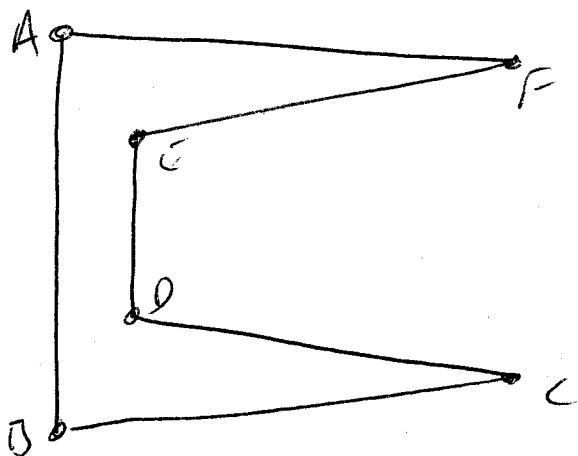
See example on p. 144

Eg: Travel pathfinding:

See SAL-JAS of Lin-Kernighan heuristic.

Bad case for TSP 2-changes:

Euclidean distances:



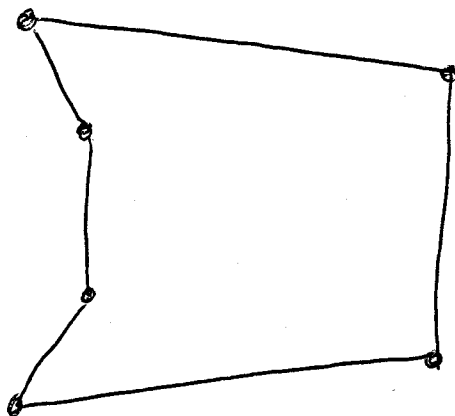
~~No~~

This is 2-opt.

Any 2-change introduces edge crossings.

But not 3-opt.

Replace $(A, B), (C, D), (E, F)$ by $(A, E), (D, B), (C, F)$.



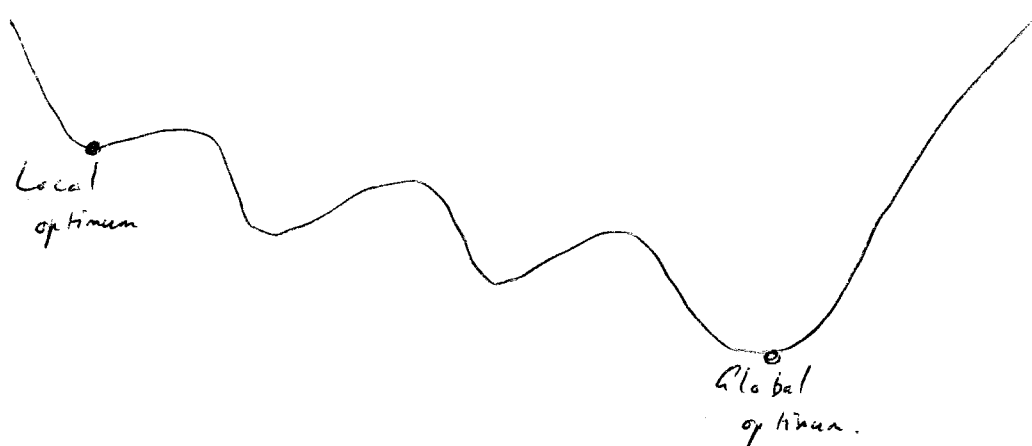
SIMULATED ANNEALING

Probabilistic algorithm. - All algorithms we've considered so far have been deterministic, so they always give the same result when started from the same point. Simulated annealing may give completely different answers ^{on a given problem}, even when started from same initial solution.

Simulated annealing is a represent of local search:

Local search: Have a current solution. Look in a neighbourhood of the solution for a better point. If find a better solution, move to it, otherwise stop - 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 neighbour 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 $T = rT$ (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 a 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 ? 0.4
- (iii) How do we pick r and L ? $r = 0.95, L = 16 \times \text{size of neighbourhood}$
- (iii) What do we mean by "FROZEN"? Go through Step 3 5 times with 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 of time $\sim 400,000, 500$
 $\sim \text{days on max 11/780.}$

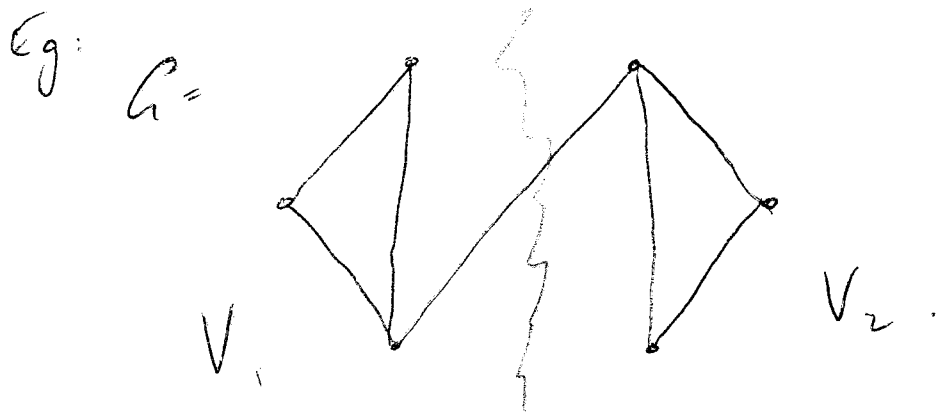
Alternative: run local search from many different starting points.

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

Need to pick r, L and initial temperature ^{large enough} 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: Graph partitioning: (Hard ^{good} comparison for simulated annealing, $\therefore \{$)

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.



Have a current partition ^{into two equal sized sets}. What is a neighbour?

Move one vertex from V_1 to V_2 and one from V_2 to V_1 .

So each solution has $\frac{n^2}{4}$ neighbours.

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

Use objective function

$$c(V_1, V_2) = |\{ (u, v) \in E : u \in V_1, v \in V_2 \}| + \alpha (|V_1| - |V_2|)^2$$

So penalize a partition which is not balanced.

Get a neighbour by moving any vertex from V_1 to V_2 or vice versa.

So have n neighbours.

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 ^{unmoved} vertices u in V_1 and v in V_2 which give greatest decrease / smallest increase in c , and swap them.

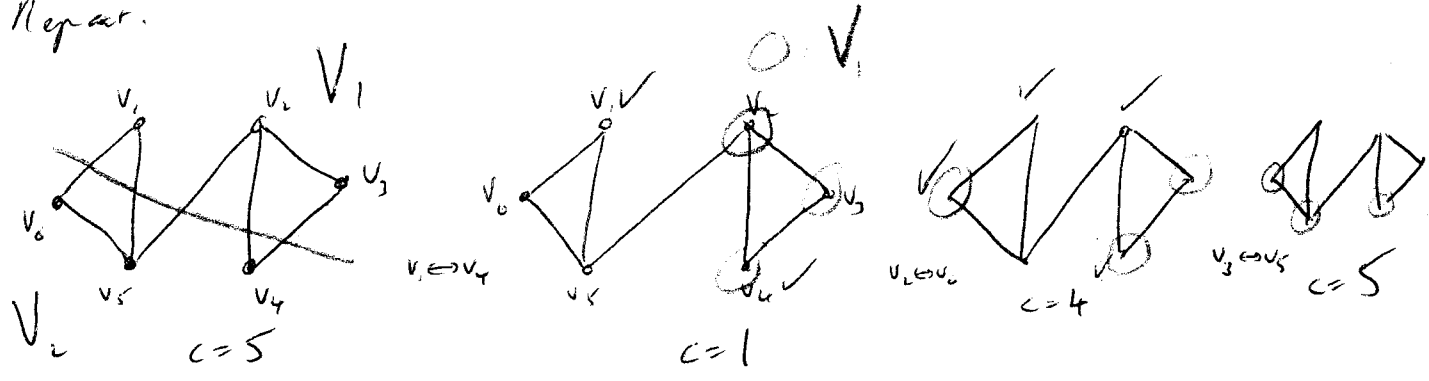
Repeat until no vertices unmoved.

We now have the same partition back again - everything in V_1 has moved to V_2 and vice versa. So replace V_1 and V_2 with the best partition found during the inner loop, provided this improves on the old V_1, V_2 .

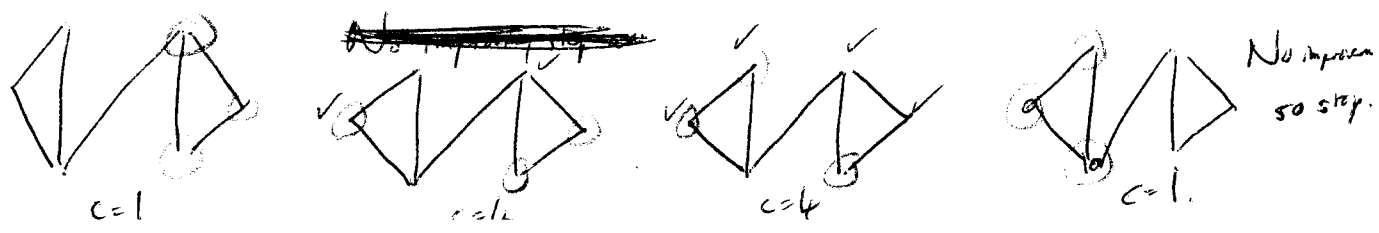
Repeat.

Eg:

Step 1



Step 2:



Comparison vs. Kernighan-Lin:

On random graphs (i.e., for each pair of ~~edges~~ 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 simulated 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 geometric graphs:

(I.e., generate ~~put~~ 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 K-L solutions, rather than from random ~~put~~ solutions.