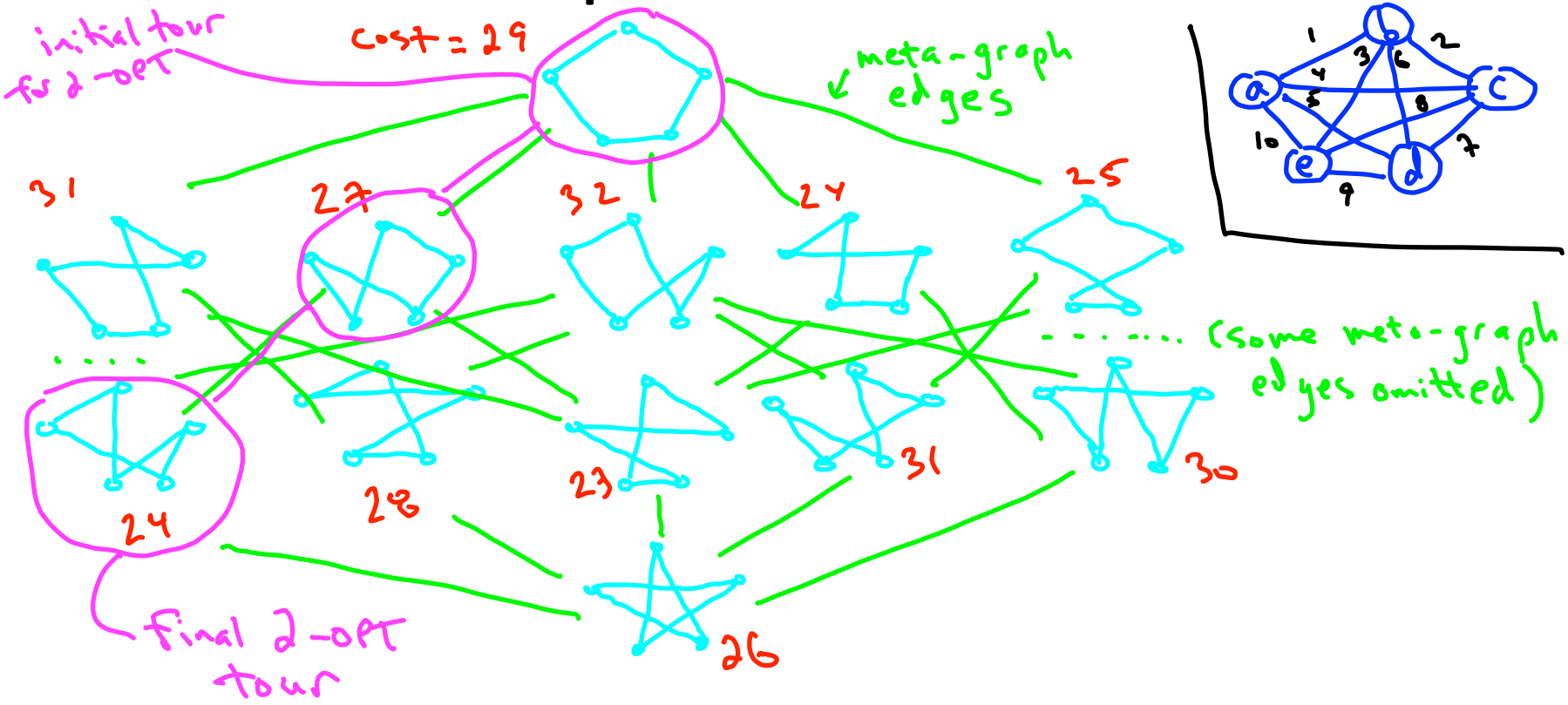
Section 20.5:
Principles of Local Search

The Meta-Graph of Feasible Solutions



Quiz #1

For a TSP instance with $n \geq 4$ vertices, how many vertices and edges does the corresponding meta-graph have?

(a) $\frac{1}{2}(n-1)!$ and $\frac{n! \cdot (n-3)}{8}$

(b) $\frac{1}{2}(n-1)!$ and $\frac{n! \cdot (n-3)}{4}$

(c) $(n-1)!$ and $\frac{n! \cdot (n-3)}{4}$

(d) $(n-1)!$ and $\frac{n! \cdot (n-3)}{2}$

$$\#\text{edges} = \underbrace{\frac{1}{2}(n-1)!}_{\#\text{vertices}} \cdot \underbrace{\frac{n(n-3)}{2}}_{\substack{\#\text{incident} \\ \text{edges per} \\ \text{vertex}}} \cdot \underbrace{\frac{1}{2}}_{\substack{\text{for} \\ \text{double} \\ \text{counting}}}$$

$$= \frac{1}{8} n! (n-3)$$

The Local Search Algorithm Design Paradigm

[most local search algorithms = analogous walk through a meta-graph]

[gradient descent – similar but for continuous optimization]

1. Define the feasible solutions (vertices of meta-graph).
2. Define the objective function ("heights" of vertices).
3. Define the allowable local moves (edges of meta-graph).

(1)–(3): modeling decisions

4. Decide how to choose initial solution.
5. Decide how to pick among multiple improving local moves.

(4)–(5): algorithmic decisions

6. Perform local search!

The Generic Local Search Algorithm

S := initial solution [as specified in Step 4]

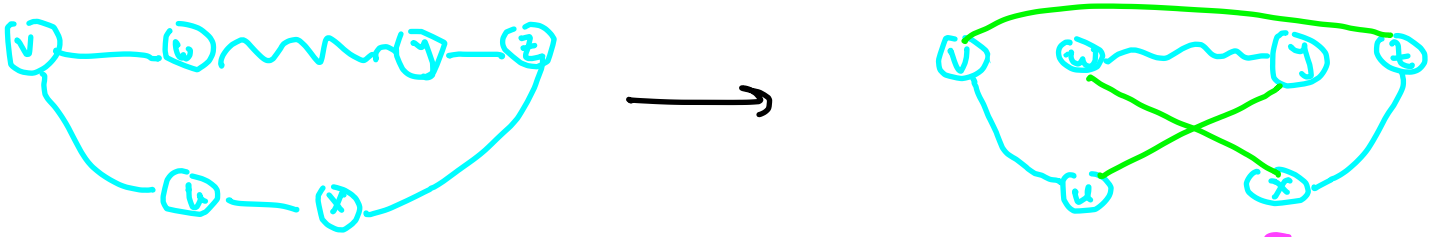
while improving local move L exists:

- S := MakeMove(S, L) [choice of L specified in Step 5]

return S ["local optimum"]

3-Change Neighborhood for the TSP

3-Change: replace 3 edges of a tour with three different edges in a way that produces a new tour.



[in general, have 7 tours like this to check]

3OPT: allow 3-changes in addition to 2-changes.

Quiz #2

Fix an instance of TSP. Let H_2 and H_3 denote the meta-graphs corresponding to the 2-OPT & 3-OPT heuristics. Which of the following are true?

(a) Every edge of H_2 is also an edge of H_3.

(b) Every edge of H_3 is also an edge of H_2.

(c) Every local minimum of H_2 is also a local minimum of H_3.

(d) Every local minimum of H_3 is also a local minimum of H_2.

Two Algorithm Design Decisions

Initialization:

- use output of a greedy heuristic
- use a random feasible solution

Choosing among improving moves:

- use the first one
- use the best one
- use a random one

Running Time and Solution Quality

(1) Guaranteed to halt. (Assuming finite # of feasible solutions.)

(2) Not guaranteed to halt in polynomial # of iterations.

(3) In applications, usually halts in tolerable number of iterations.

(4) Can interrupt at any time (use best solution found).

(5) No provable approximate correctness guarantees.

(6) On realistic inputs, often produce high-quality local optima, sometimes produce low-quality optima.

Avoiding Bad Local Optima

(1) Allow more local moves.

(2) Randomization!

(3) Allow non-improving moves.

E.g.: (cf., "simulated annealing")
- pick local move at random
- if makes objective worse by Δ, make move w/ probability $p(\Delta)$.

(4) History-dependent neighborhoods.
- cf., tabu search; Lin-Kernighan

(5) Maintaining a population of solutions.
- cf., genetic algorithms; beam search

When to Use Local Search

1. No time to compute an optimal solution.
2. Don't care about running time + approximate correctness guarantees.
3. Minimize coding time.
4. No-downside post processing.
5. Want an interruptible algorithm.
6. State-of-the-art solvers not good enough.

Remember: experiment!!