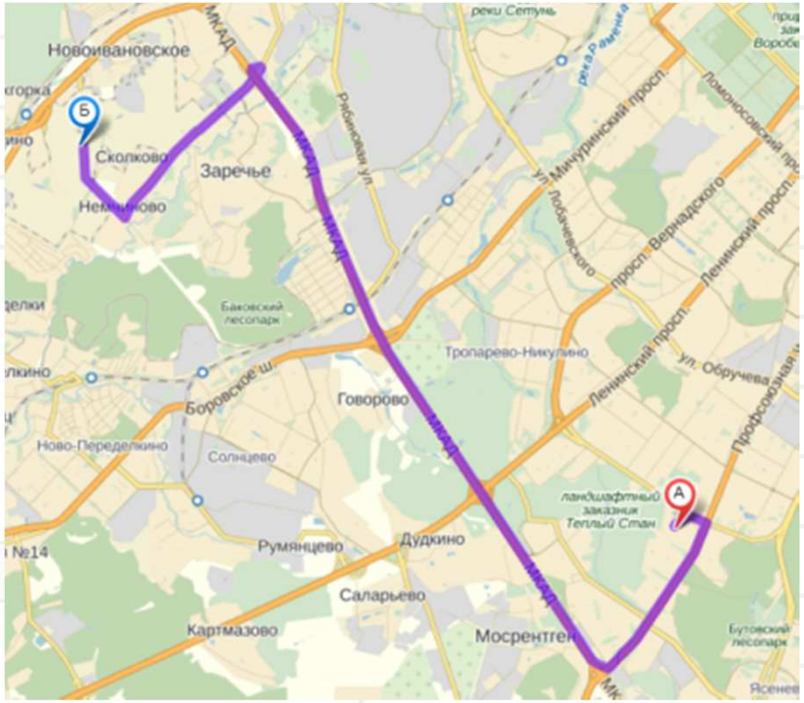
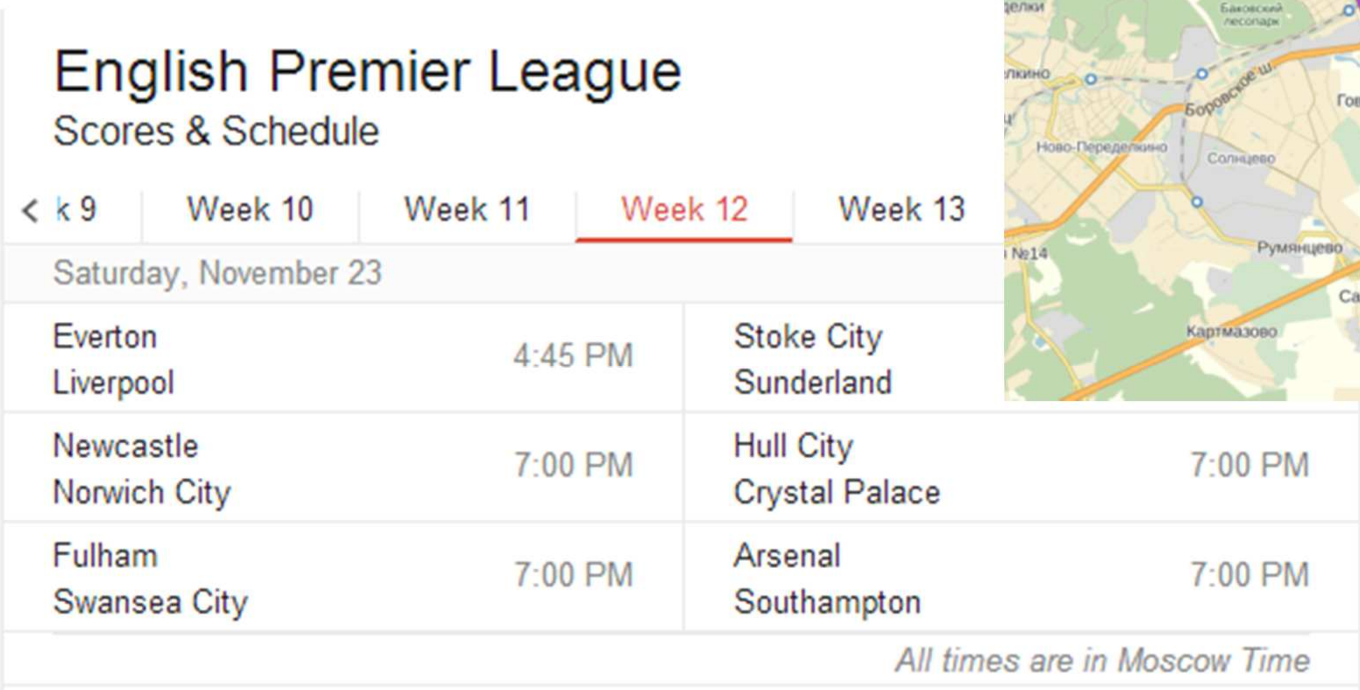
Lecture 2: Discrete optimization. Local methods.

Overview of the discrete optimization

Many (the majority?) of problems have some discrete entities in them. E.g.:

- Making choices
- Assigning labels





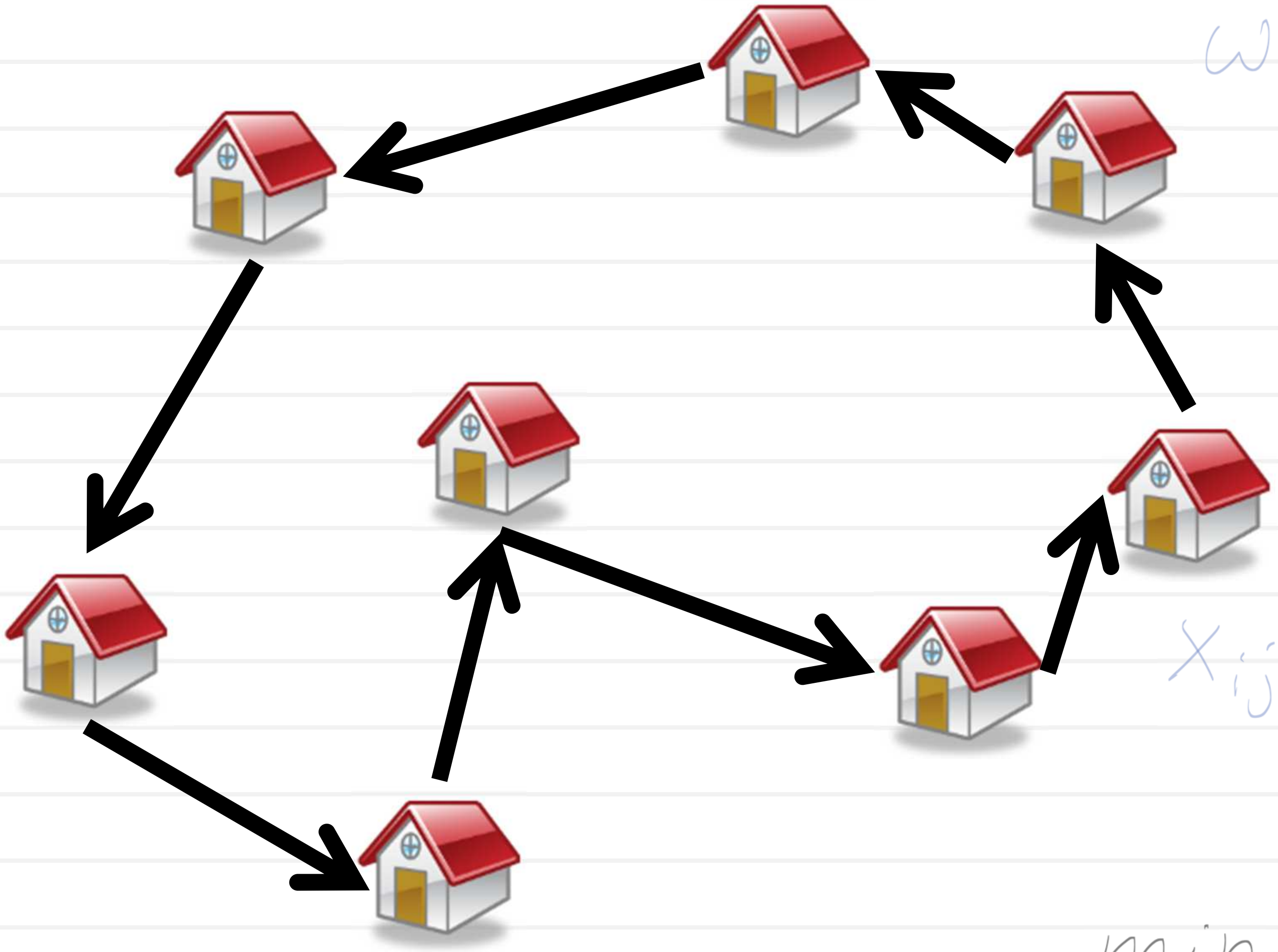
Everton Liverpool	4:45 PM	Stoke City Sunderland	
Newcastle Norwich City	7:00 PM	Hull City Crystal Palace	7:00 PM
Fulham Swansea City	7:00 PM	Arsenal Southampton	7:00 PM

Overview of the discrete optimization

- Discrete optimization is hard
- Unlike convex programming, global minima are often unattainable
- Finding a good non-optimal solution quickly is non-trivial

Travelling salesman problem

Task: find the shortest *tour*

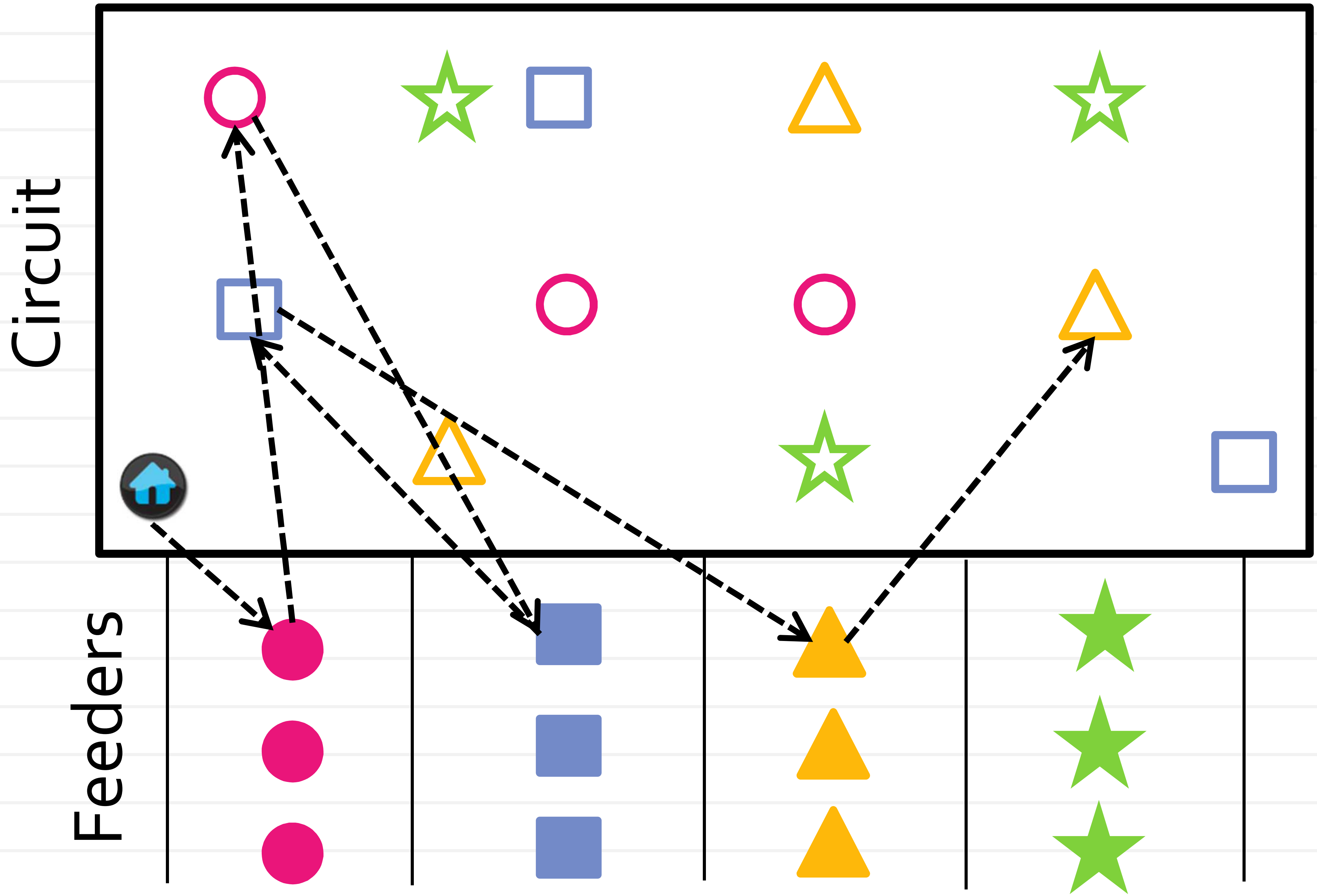


w_{ij} walking distance from house i to house j

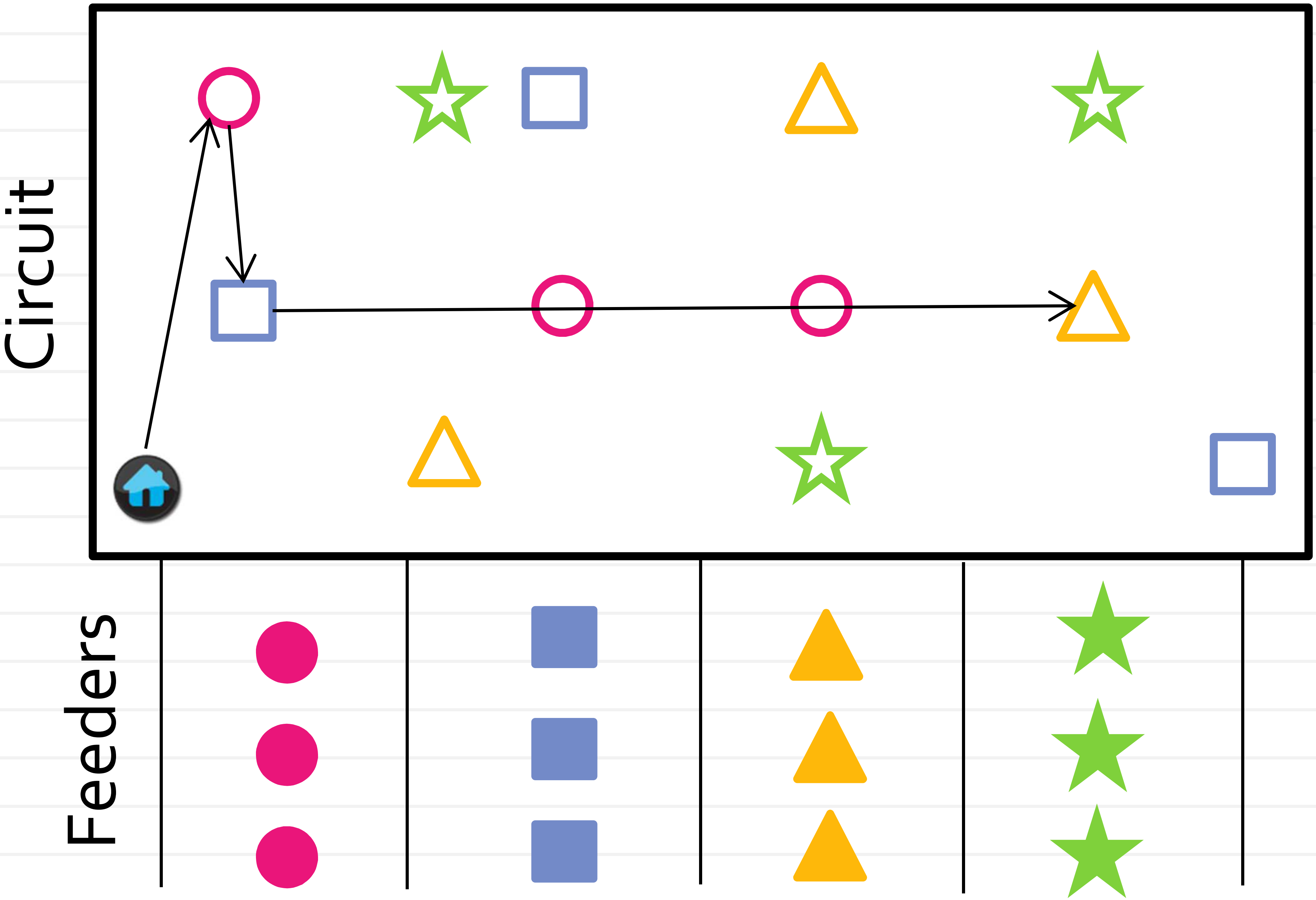
x_{ij} 0-1 decision (include into the tour or not)

$$\min \; w_{ij} x_{ij}$$
$$\text{s.t.} \; \{x_{ij}\} \in Tour(G)$$

Electronics assembly



Electronics assembly



Scheduling (single machine)

- Machine “visits” jobs
- Switching between jobs takes various times
- Seek for optimal schedule for repetitive manufacturing process

Job1

Job2

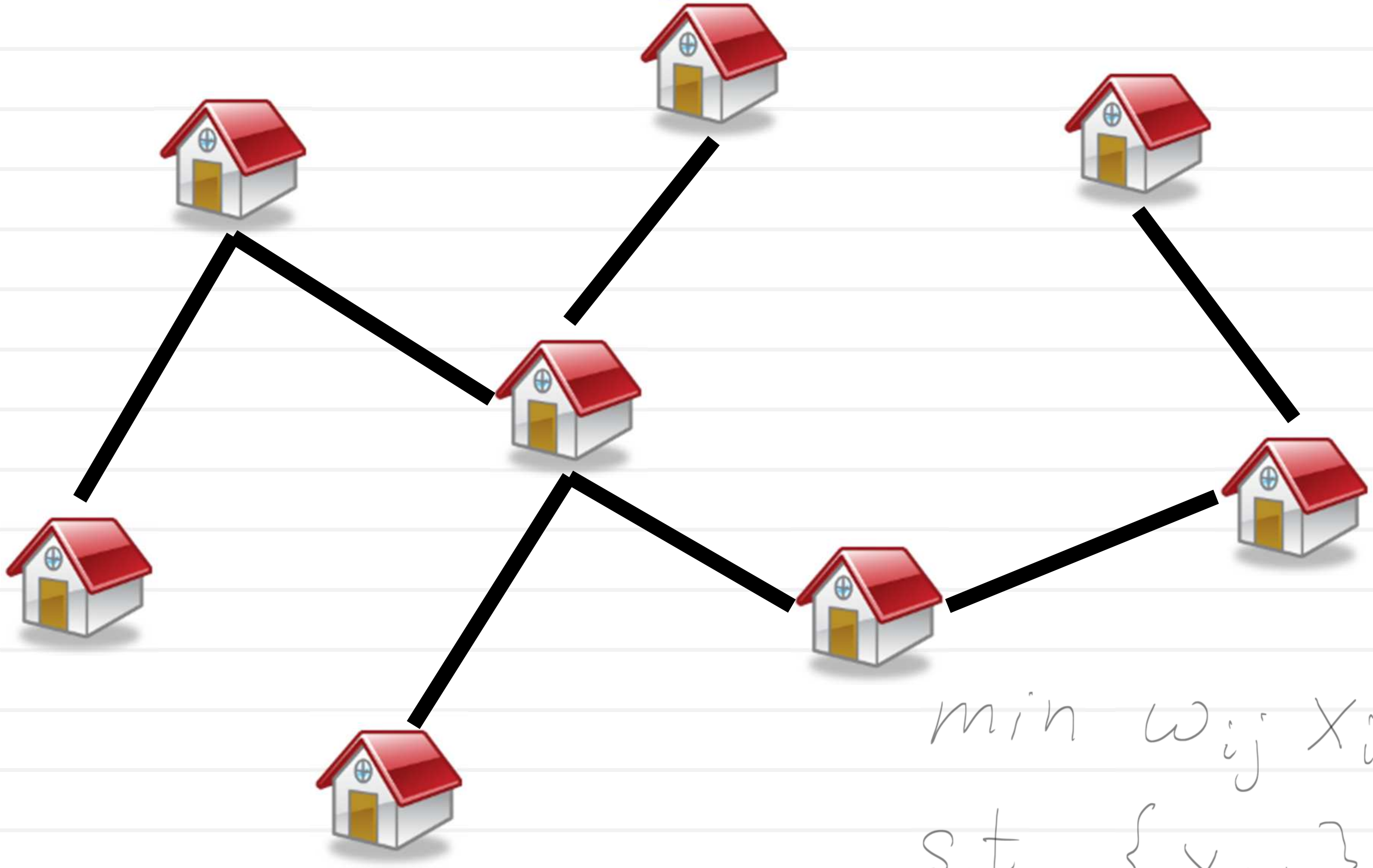
Job3

Job4

Job5

Job6

Minimum spanning tree



$$\min \; w_{ij} x_{ij}$$

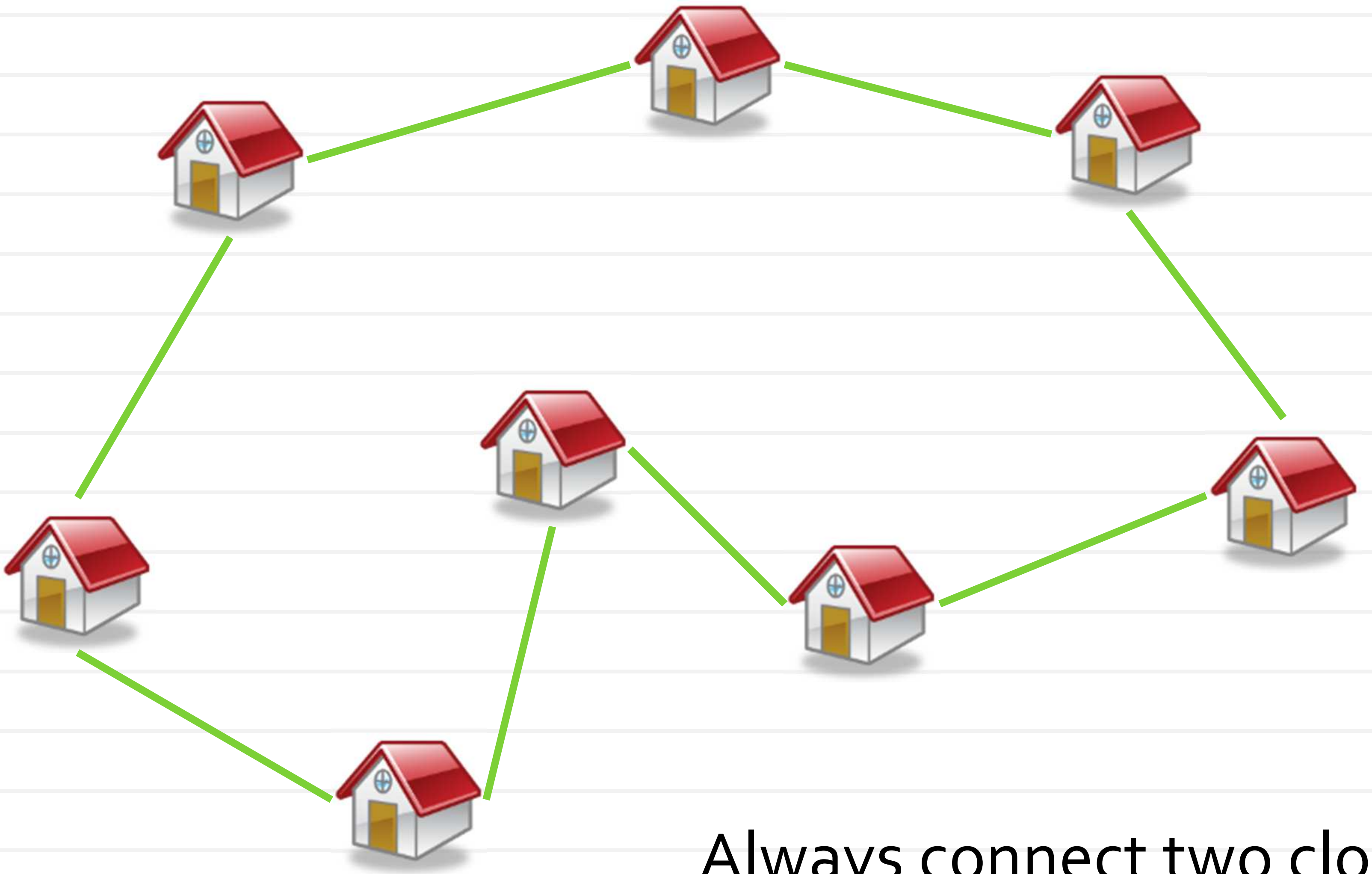
s.t. $\{x_{ij}\} \in$  (G)

- Task: find the shortest spanning tree
- Initial interest: planning electrical networks

TSP: complexity

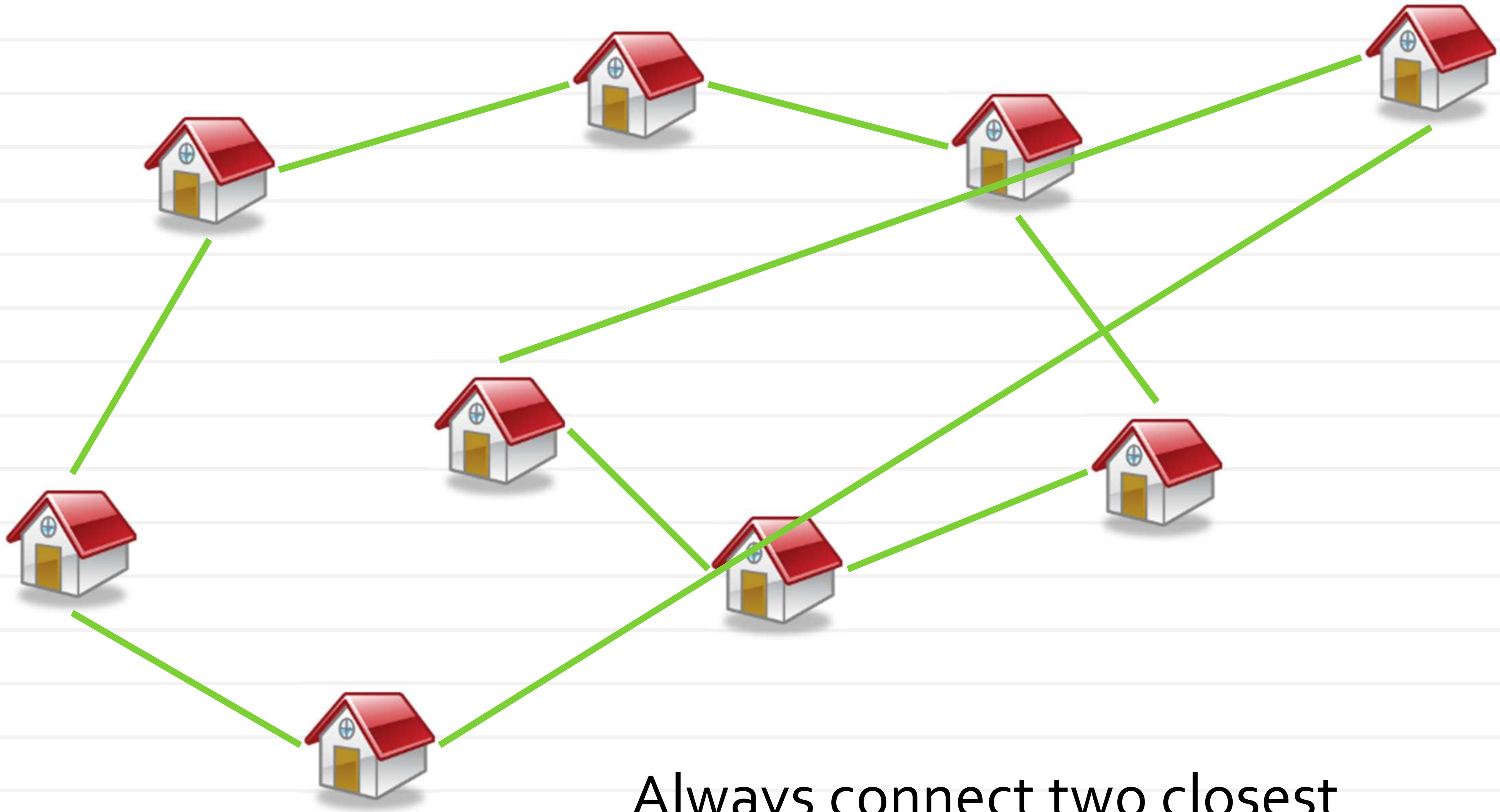
- The problem is NP-complete
- Polynomial algorithms are unknown/unlikely
- For large instances, have to consider *approximate* algorithms

Heuristic 1: Greedy NN



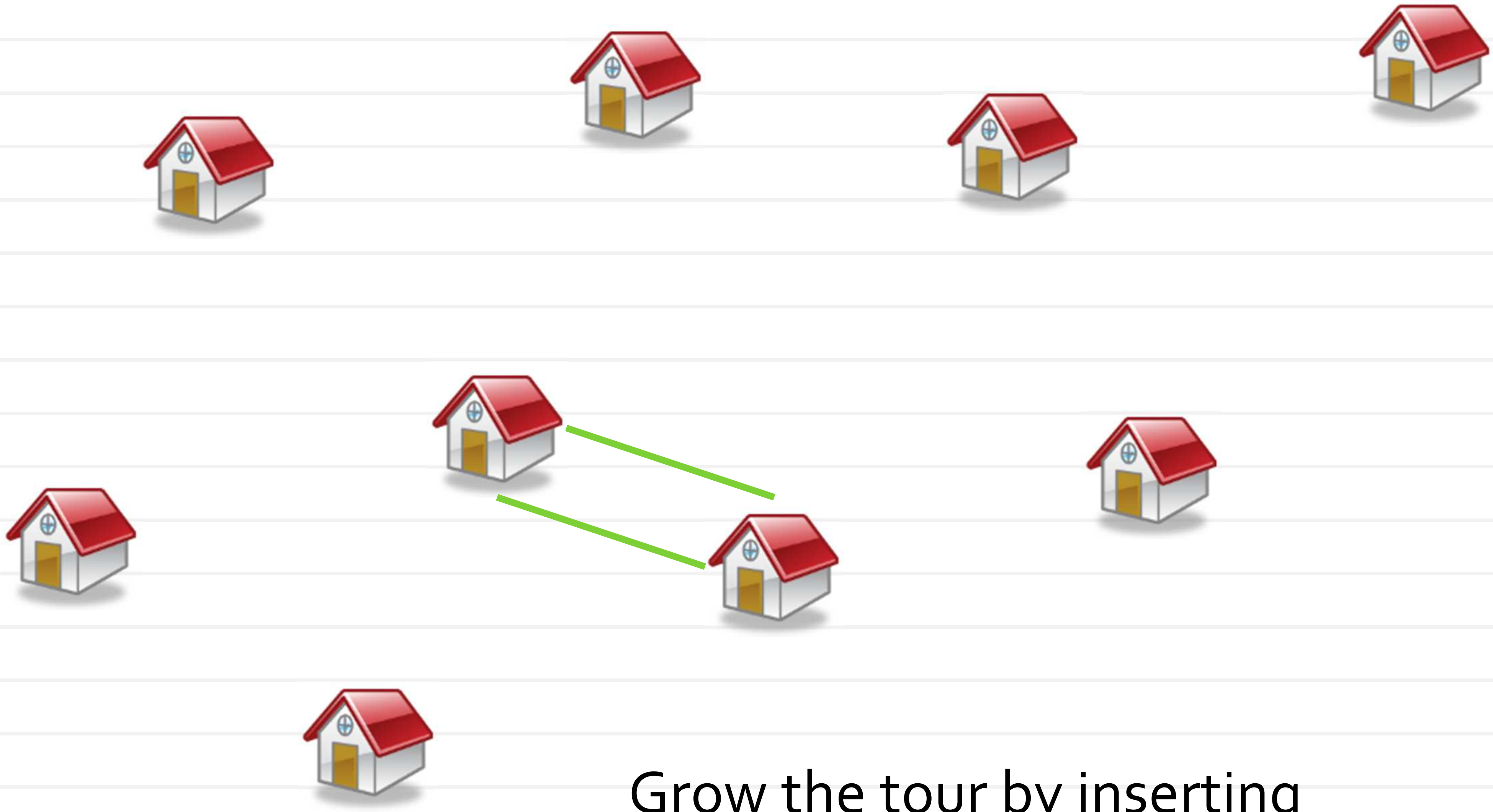
Always connect two closest neighbors, which are not in the middle of the chain

Heuristic 1: Greedy NN



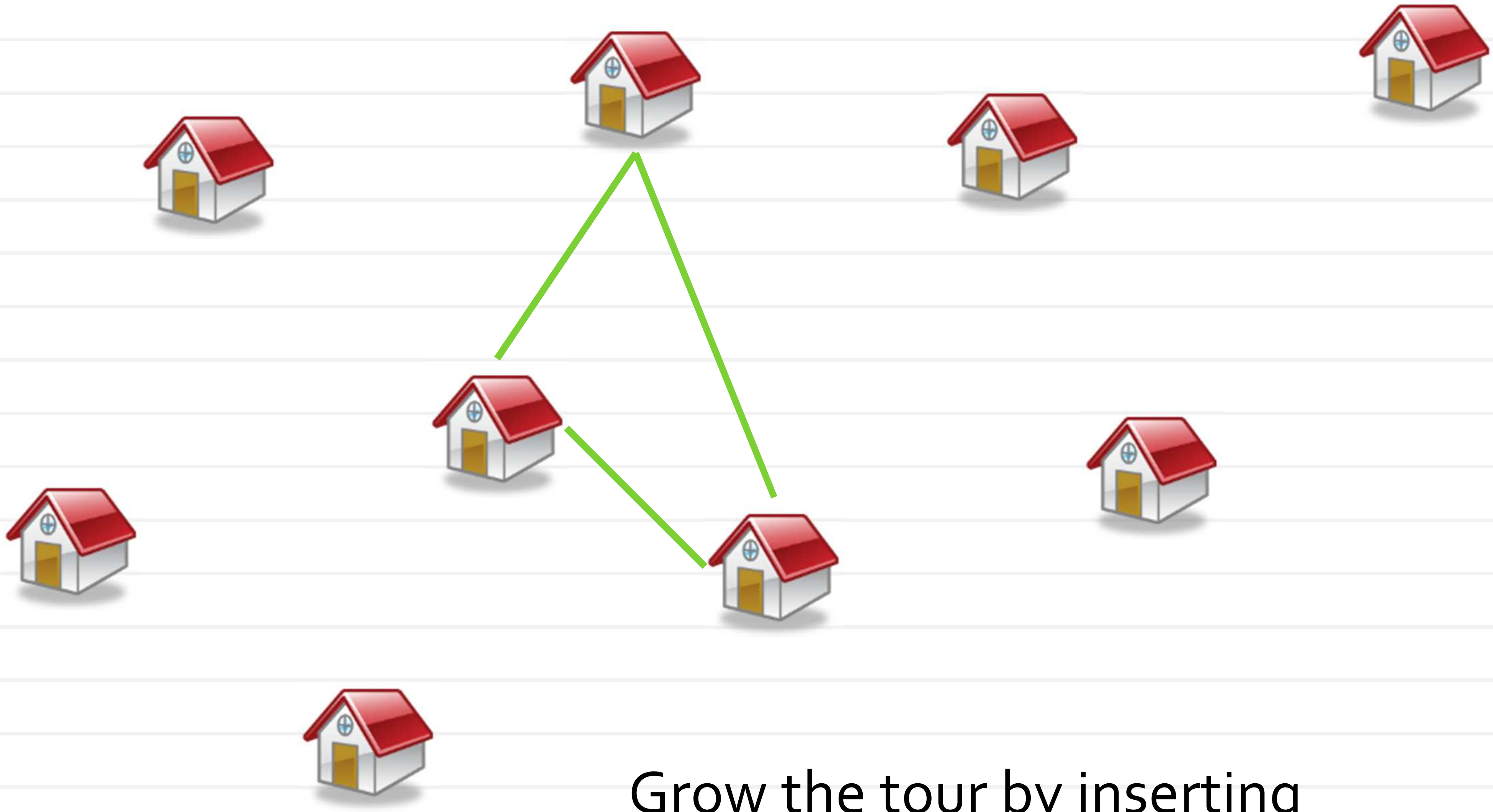
Always connect two closest neighbors, which are not in the middle of the chain

Heuristic 2: Greedy insertion



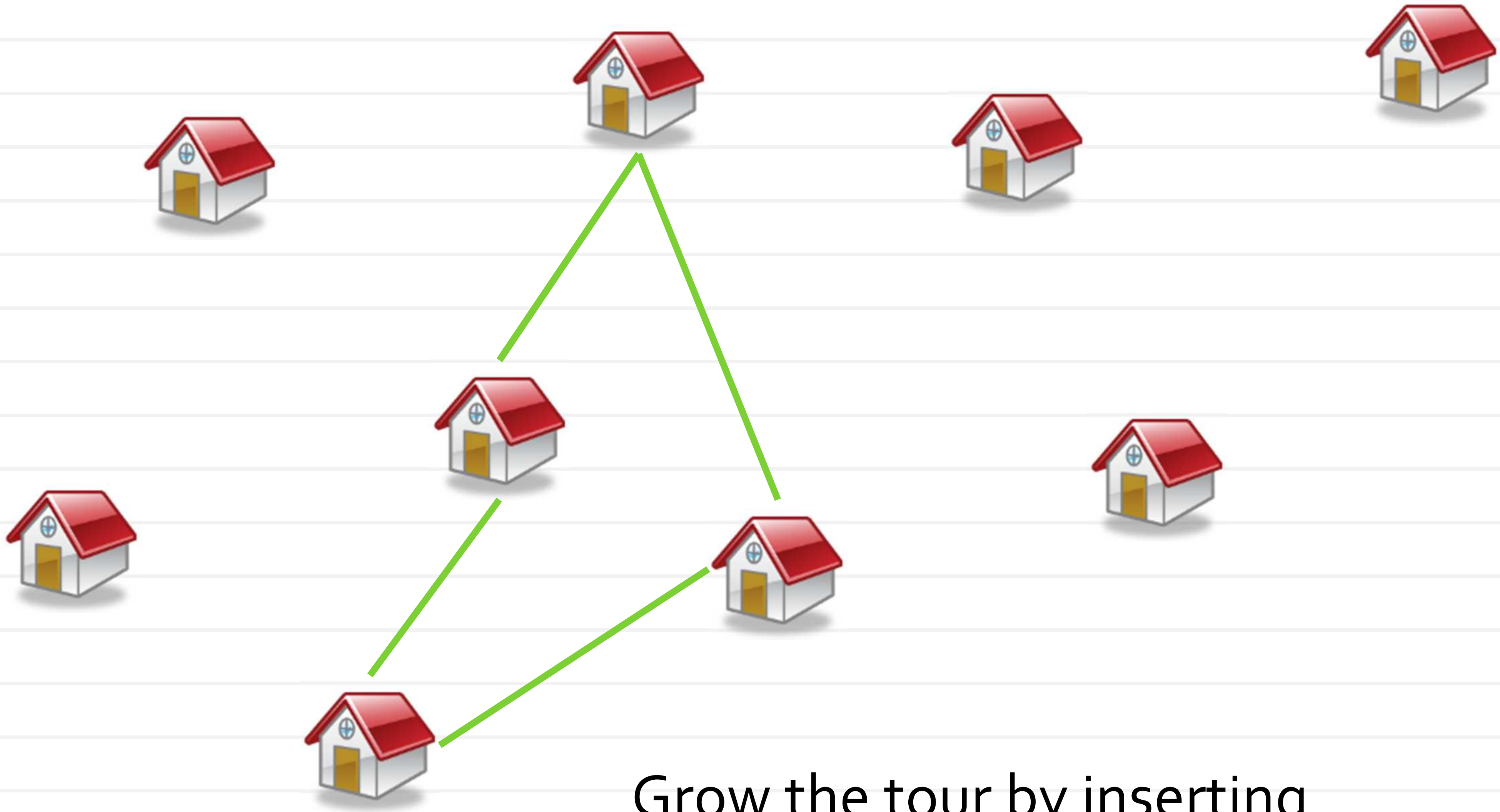
Grow the tour by inserting cheapest to include vertex

Heuristic 2: Greedy insertion



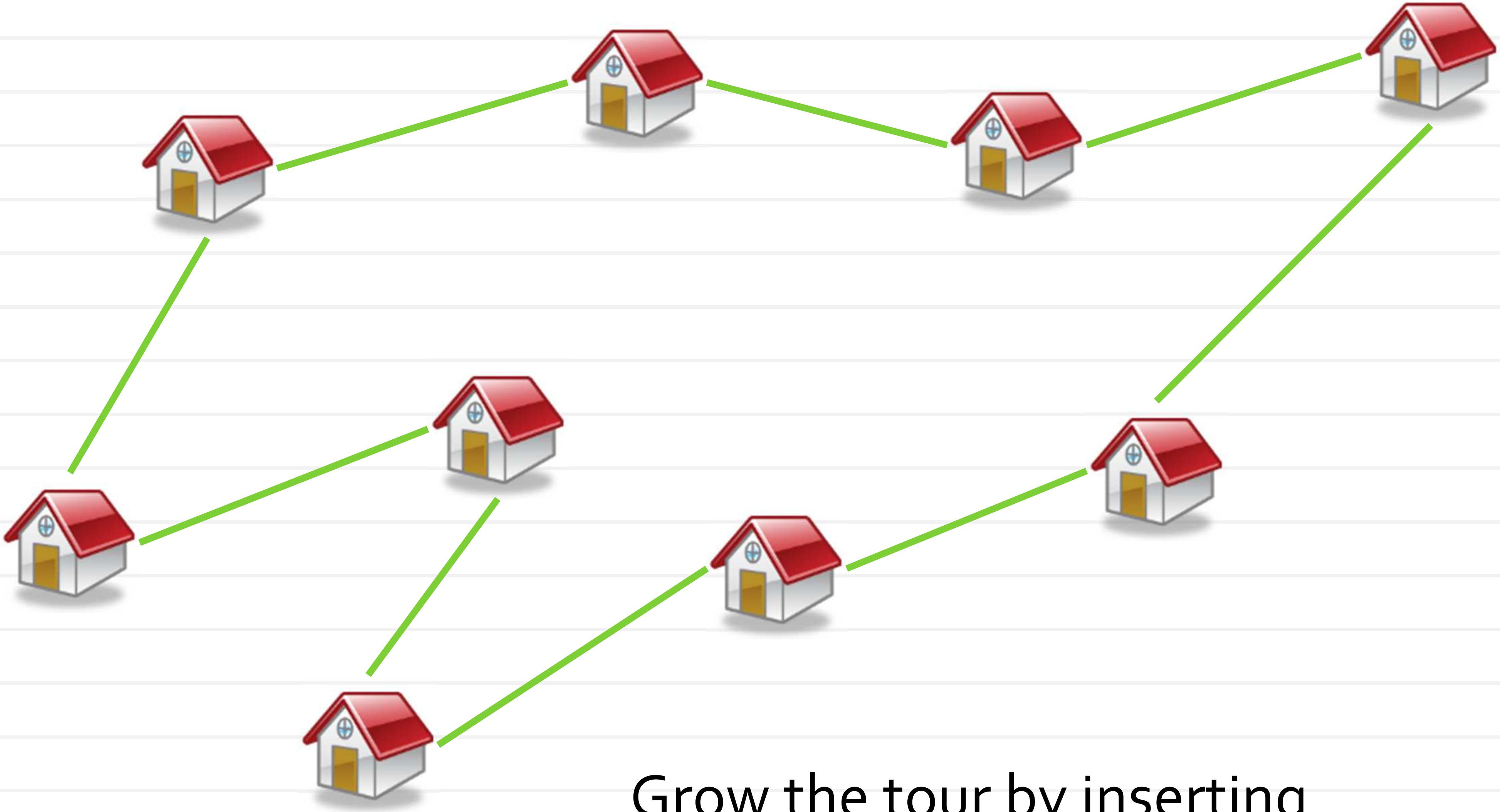
Grow the tour by inserting cheapest to include vertex

Heuristic 2: Greedy insertion



Grow the tour by inserting cheapest to include vertex

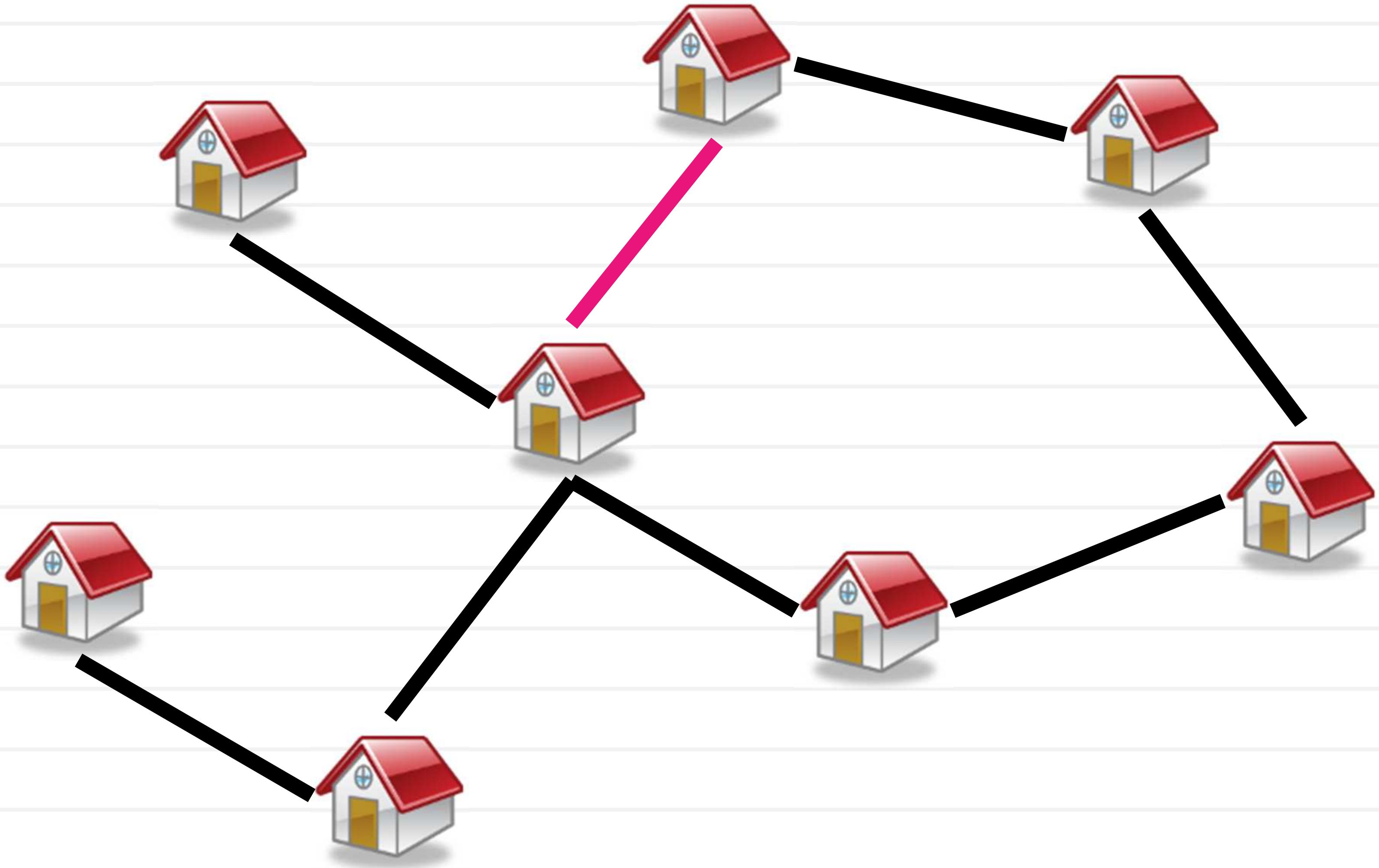
Heuristic 2: Greedy insertion



Grow the tour by inserting cheapest to include vertex

Kruskal's algorithm for MST

Some non-trivial problems can be solved greedily.



Kruskal's algorithm for MST

Some non-trivial problems can be solved greedily.

(pseudocode from Wikipedia)

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
    MAKE-SET(v)
foreach (u, v) ordered by weight(u, v), increasing:
    if FIND-SET(u) ≠ FIND-SET(v):
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

- **Finds a globally optimal tree**
- With proper data structures, runs "almost" linearly in the number of edges.

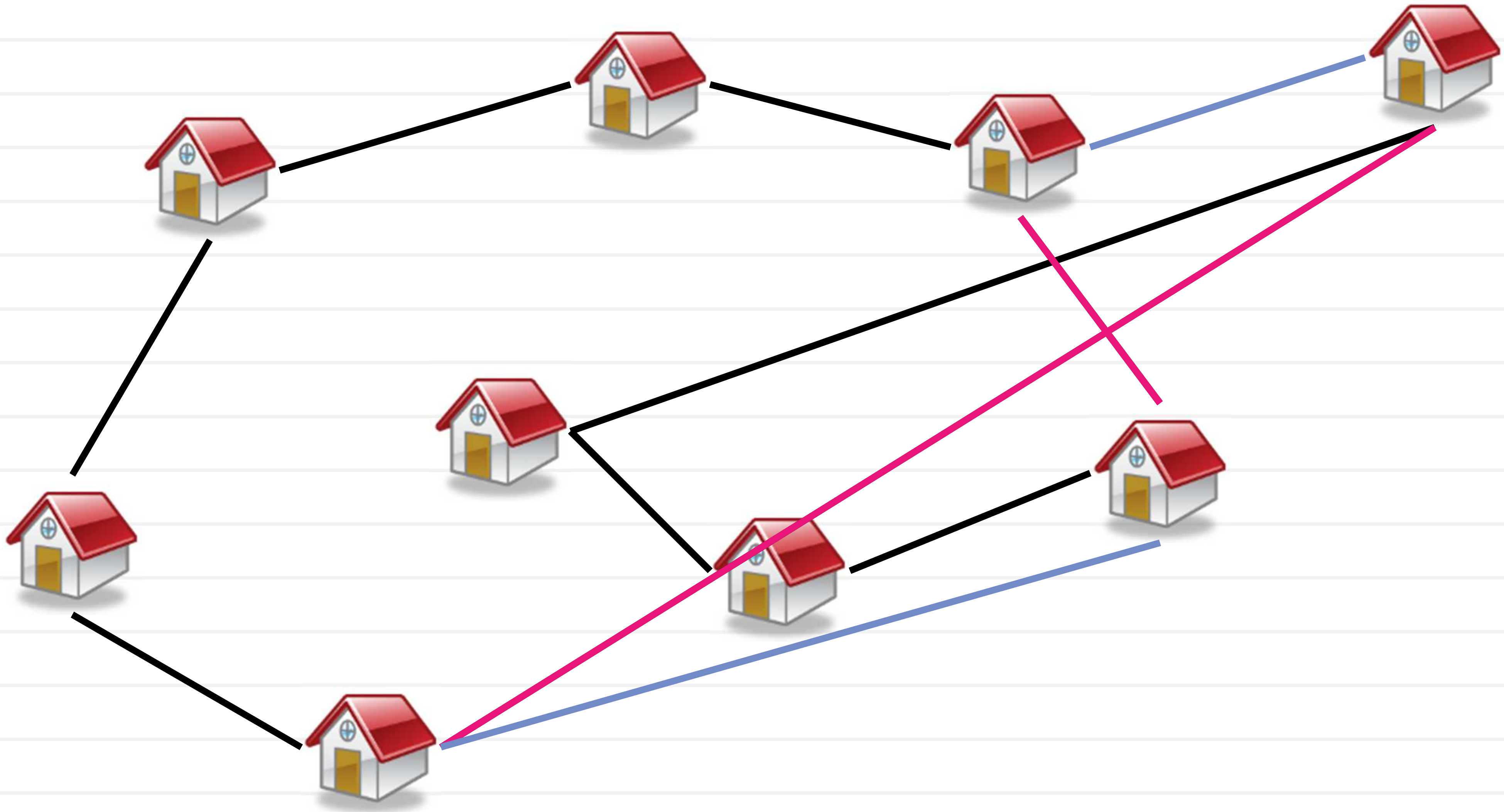
Local Search

```
x = starting position;
while nIter++ < MAX_ITER
        y = argmin( neighborhood(x) );
        if  f(y) < f(x)
                x = y;
        else    \\no improvement found
                break;
end while
```

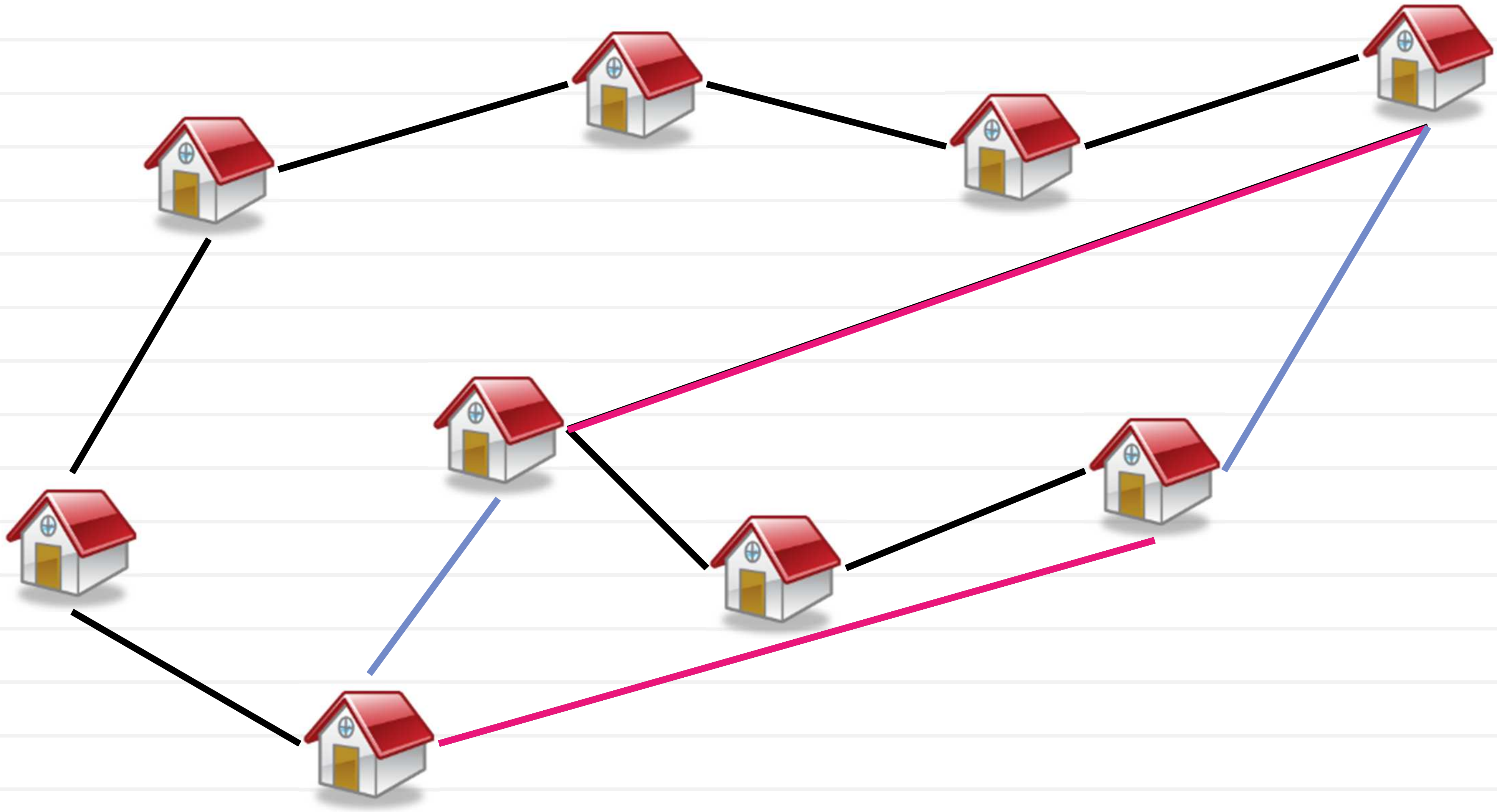
Random, null, greedy….

Exhaustive search
Random sampling

Heuristic: local search



Heuristic: local search

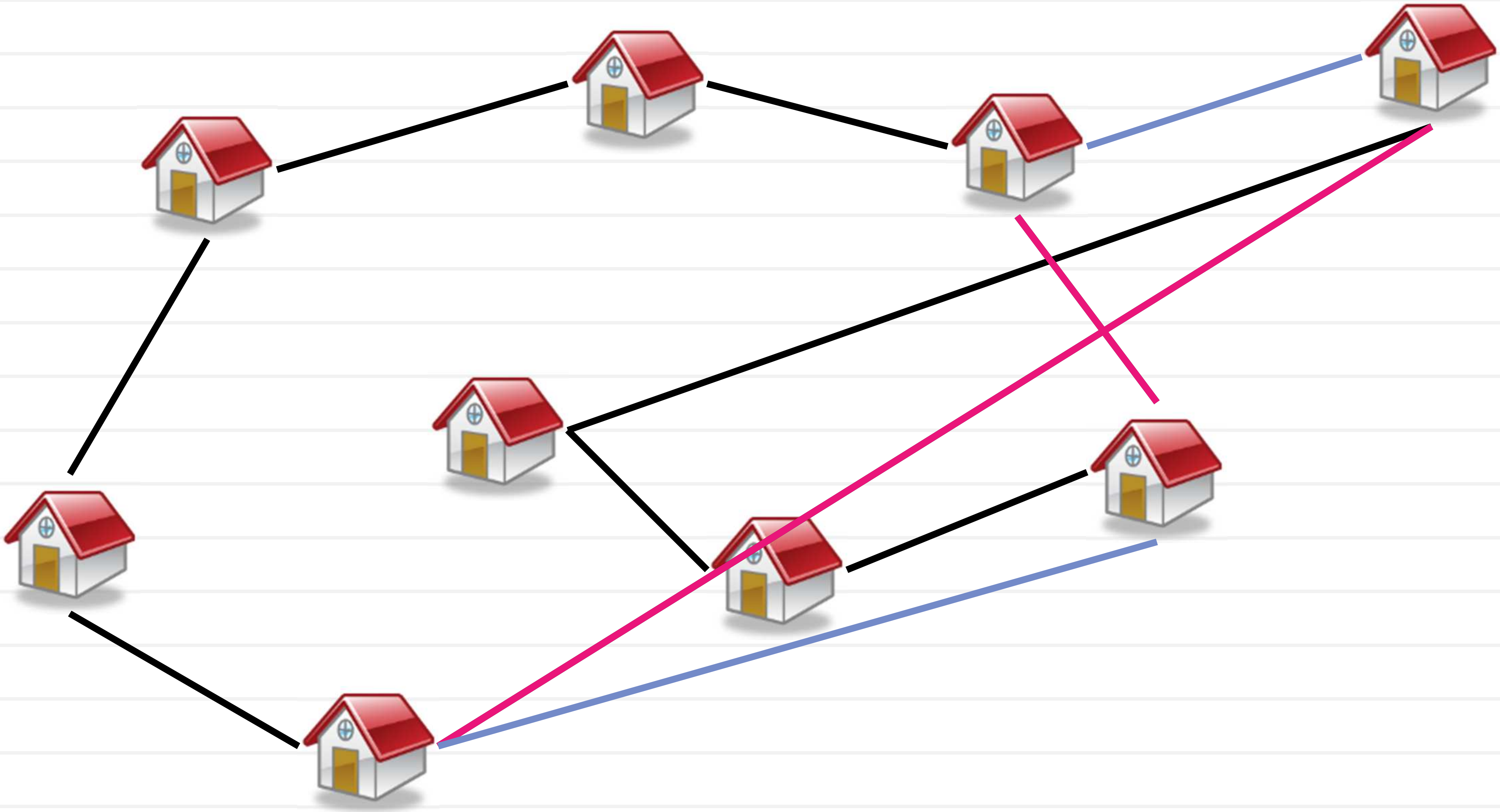


Local search and local optima

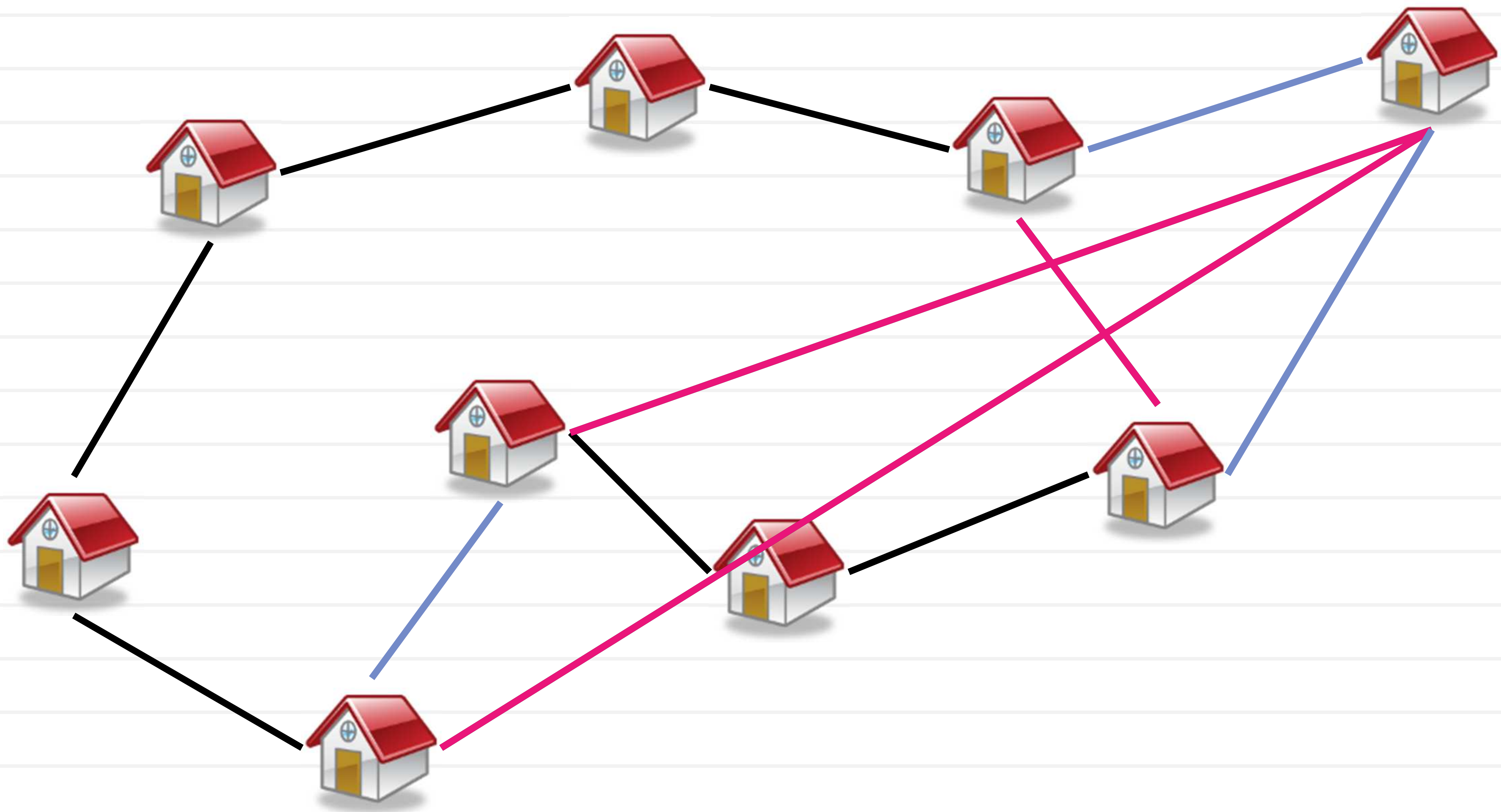
Definition: the *neighborhood* of X *w.r.t.* a local search algorithm A is the set of configurations within one step from X.

- The run of the algorithm will end up in a *local minimum* (w.r.t. A).
- Different local search algorithms have different neighborhoods and different sets of local minima.

2-Opt neighborhood for TSP



3-Opt neighborhood for TSP



Infeasible local search

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.:} \quad & x \in D \end{aligned}$$

$$\begin{aligned} \min \quad & f_\lambda(x) \\ \text{s.t.:} \quad & x \in D' \end{aligned}$$

$$\begin{aligned} \min_x \quad & \underbrace{f(x) + \lambda p(x, D)}^{f_\lambda(x)} \\ \text{s.t.:} \quad & x \in D' \supset D \end{aligned}$$

- Particularly useful for complex domains
- λ might be changing (increasing) through the process

Greedy as infeasible local search

Greedy can be interpreted as an infeasible local search

$$\min_x f(x) + \lambda p(x, D)$$

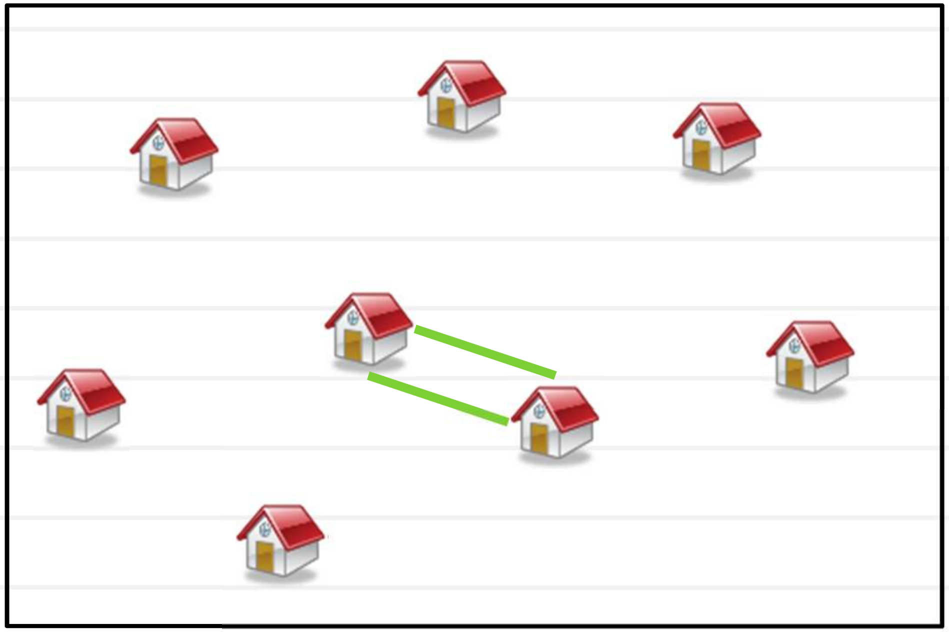
$$\text{s.t.: } x \in D'$$

$\lambda \to 1$

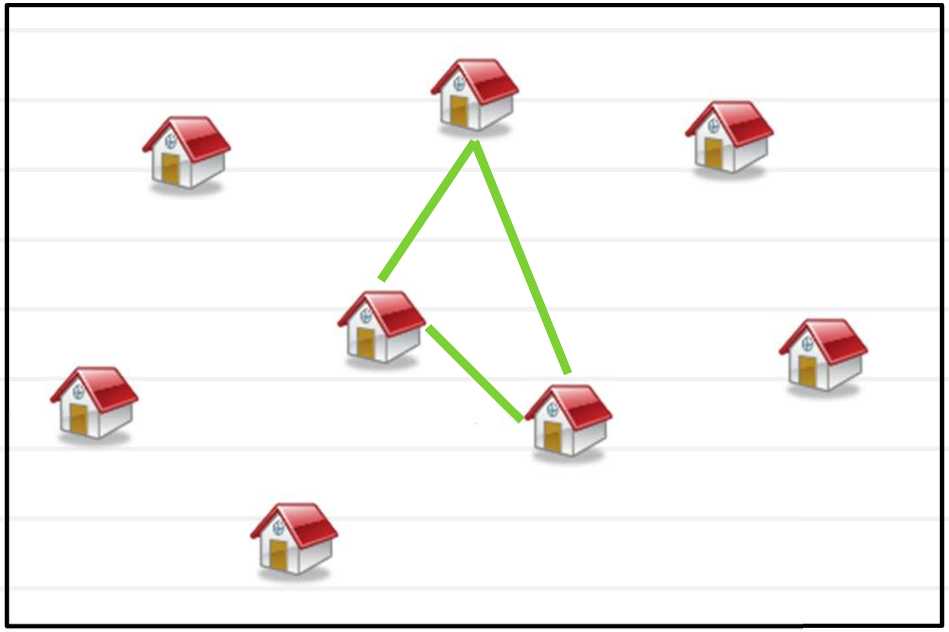
D = Tour(V)

$D' = \{Tour(V') | V' \subset V\}$

$p(x, D) = N - N(x)$



$p(x, D) = 6$



$p(x, D) = 5$

Greedy/local search

- Often easy to implement
- Often fast
- Sometimes competitive

- In most cases, only a local minimum is achievable (sometimes, bad ones)
- In many cases, no optimality guarantees

Local Search with restart

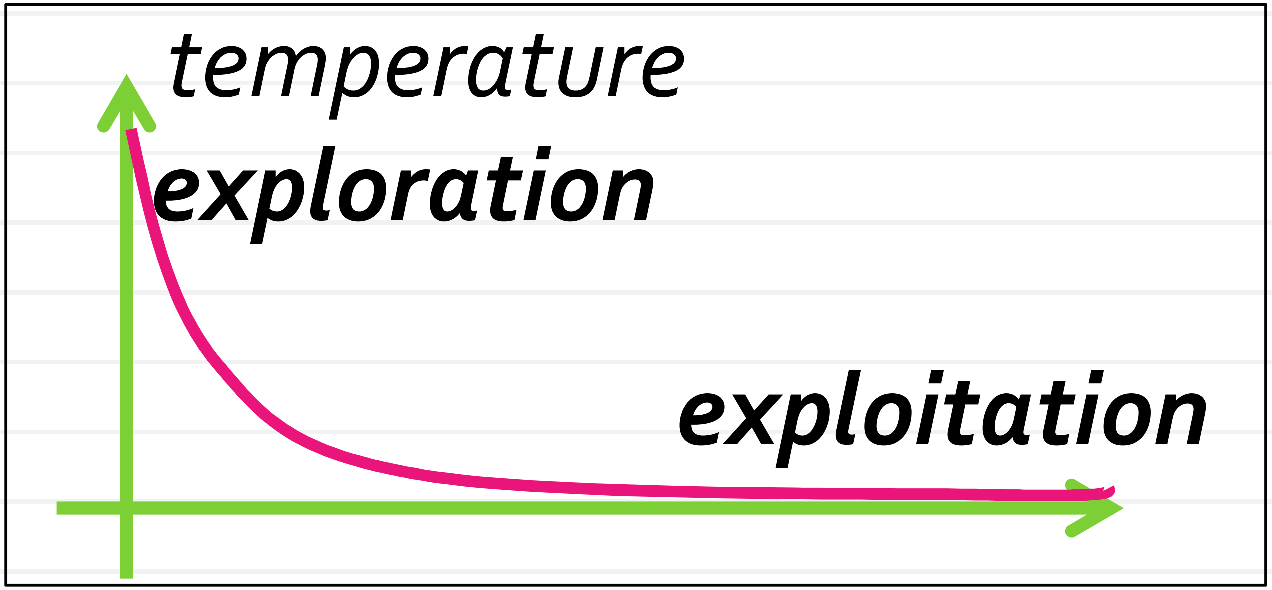
```
x = starting position; best = x;
while nIter++ < MAX_ITER
        for y in neighborhood(x)
                if  f(y) < f(x)
                        x = y;
                if f(y) < f(best)
                        best = y
        end for
        if something (e.g. local minimum, M steps done)
                x = random position;
end while
```

Exploitation

Exploration

Simulated Annealing

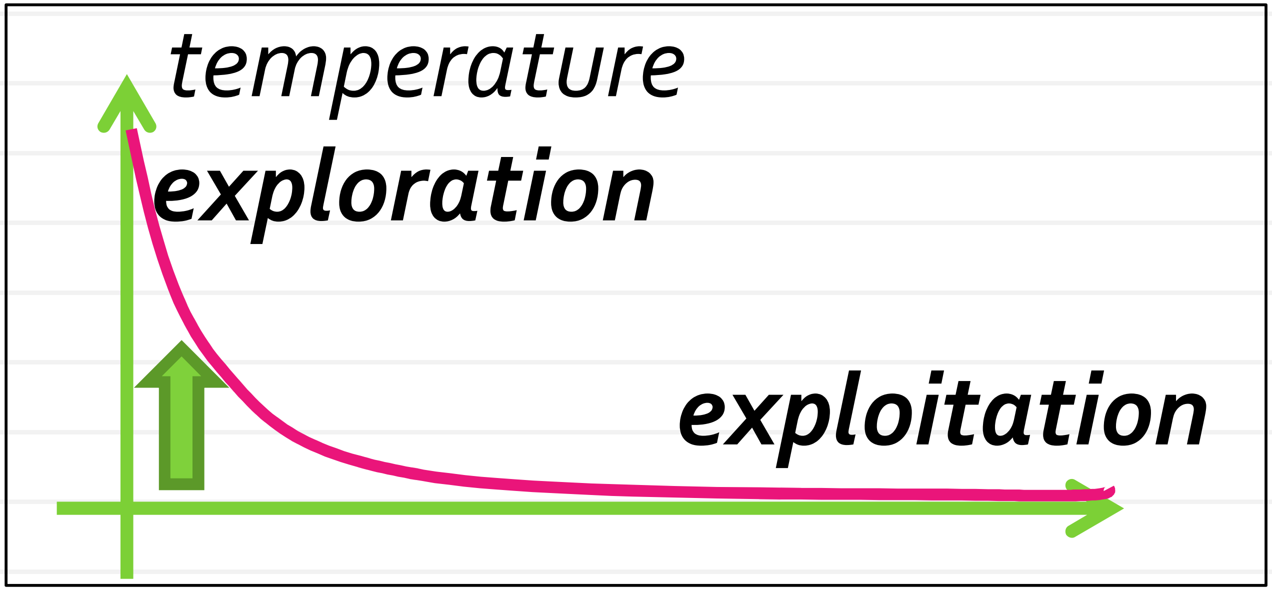
```
x = starting position, bestx = 0, bestf = +∞, T = T0, γ=0.999;
while nIter++ < MAX_ITER
        y = random in neighborhood(x)
        dE = f(y) - f(x)
        if exp(-dE/T ) > rand(0,1)
                x = y;
                if f(x) < bestf
                        bestx = x, bestf = f(x)
                end
        end
        T = T*γ;
end while
```



Simulated Annealing: high T

```
x = starting position, bestx = 0, bestf = +∞, T = T0, γ=0.999;
while nIter++ < MAX_ITER
        y = random in neighborhood(x)
        dE = f(y) - f(x)
        if exp(-dE/T ) > rand(0,1)
                x = y;
                if f(x) < bestf
                        bestx = x, bestf = f(x)
                end
        end
        T = T*γ;
end while
```

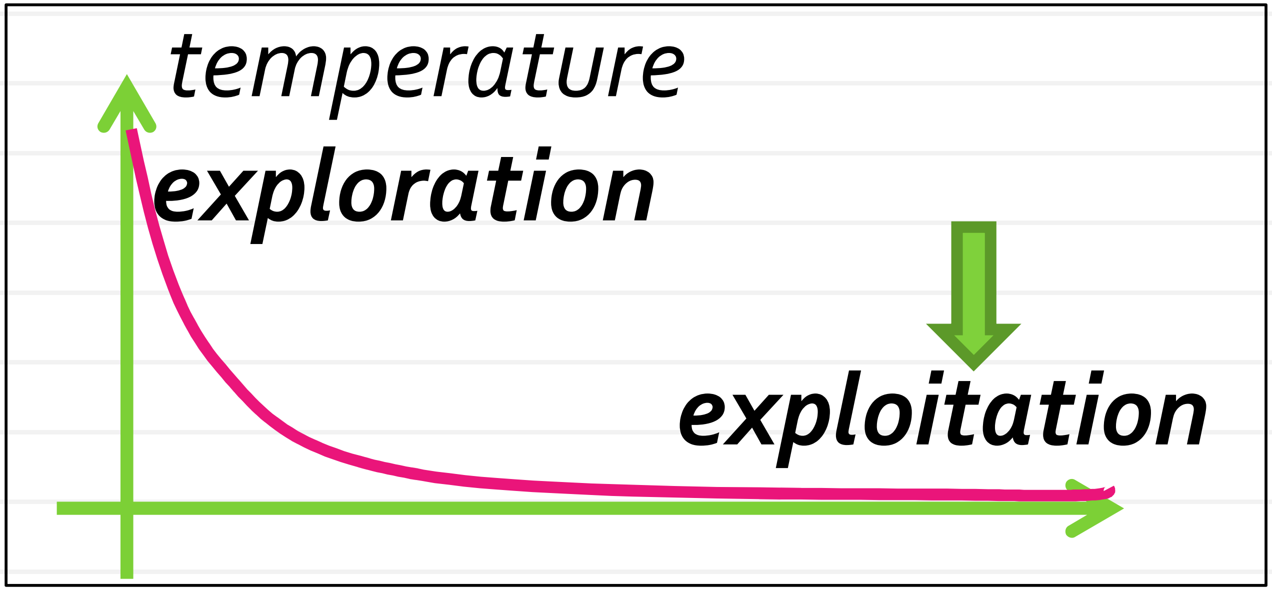
close to 1



Simulated Annealing: low T

```
x = starting position, bestx = 0, bestf = +∞, T = T0, γ=0.999;
while nIter++ < MAX_ITER
        y = random in neighborhood(x)
        dE = f(y) - f(x)
        if exp(-dE/T ) > rand(0,1)
                x = y;
                if f(x) < bestf
                        bestx = x, bestf = f(x)
                end
        end
        T = T*γ;
end while
```

either large or close to zero



Simulated Annealing

```
x = starting position, bestx = 0, bestf = +∞, T = T0, γ=0.999;
while nIter++ < MAX_ITER
        y = random in neighborhood(x)
        dE = f(y) - f(x)
        if exp(-dE/T ) > rand(0,1)
                x = y;
                if f(x) < bestf
                        bestx = x, bestf = f(x)
                end
        end
        T = T*γ;
end while
```

Dennis Rapaport toolbox:

http://www.ph.biu.ac.il/~rapaport/java-apps/travel.html

Tabu search

```
x = starting position, bestx = 0, bestf = +∞
while nIter++ < MAX_ITER
        y = argmin( neighborhood(x) \ TabuSet);
        x = y;
        if f(x) < bestf
                bestx = x, bestf = f(x)
        end
        AddToTabuSet(x);
        Expire(TabuSet);
end while
```

Max Nagl toolbox:
http://siebn.de/other/tabusearch/

Tabu set

Tabu set:

- Just a list of points of a certain length
- Set of sets of points (e.g. all tours containing the edge 4-3)

- It is possible to tabu moves rather then configurations
- We can allow tabu moves if they are improving the best solution we have seen

Block-coordinate descent

$\min\limits_{x,y} f(x,y)$ s.t. $(x,y) \in D$	$\min\limits_{x} f(x,y)$ s.t. $(x,y) \in D$	$\min\limits_{y} f(x,y)$ s.t. $(x,y) \in D$
"*hard*"	"*doable*"	"*doable*"

$x = x_0$, $y = \arg\min\limits_{y} f(x,y)$ s.t. $(x,y) \in D$

Loop

$x' = \arg\min\limits_{x} f(x,y)$ s.t. $(x,y) \in D$

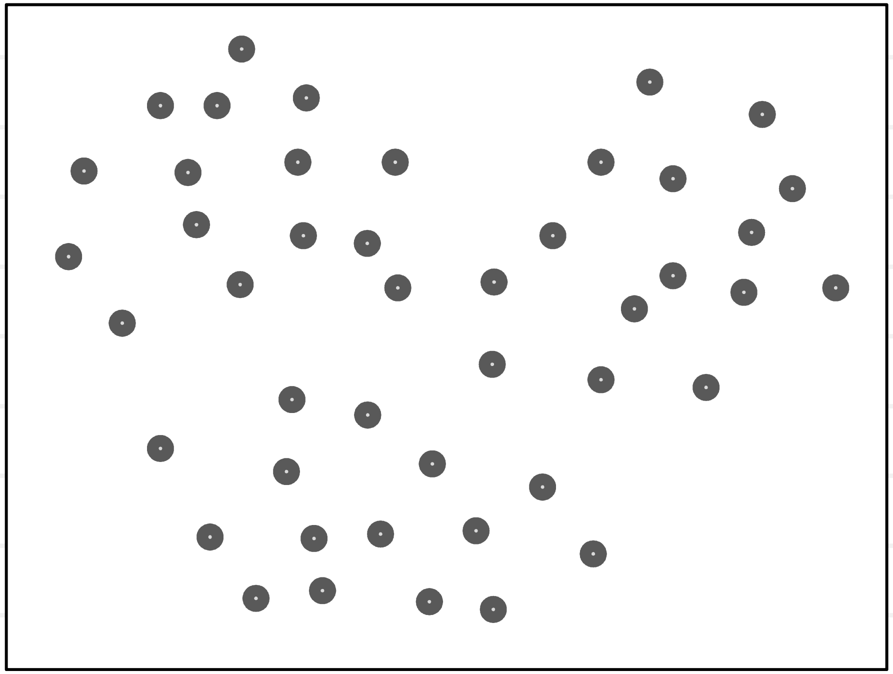
if $x == x'$ ***return;*** *else* $x = x'$

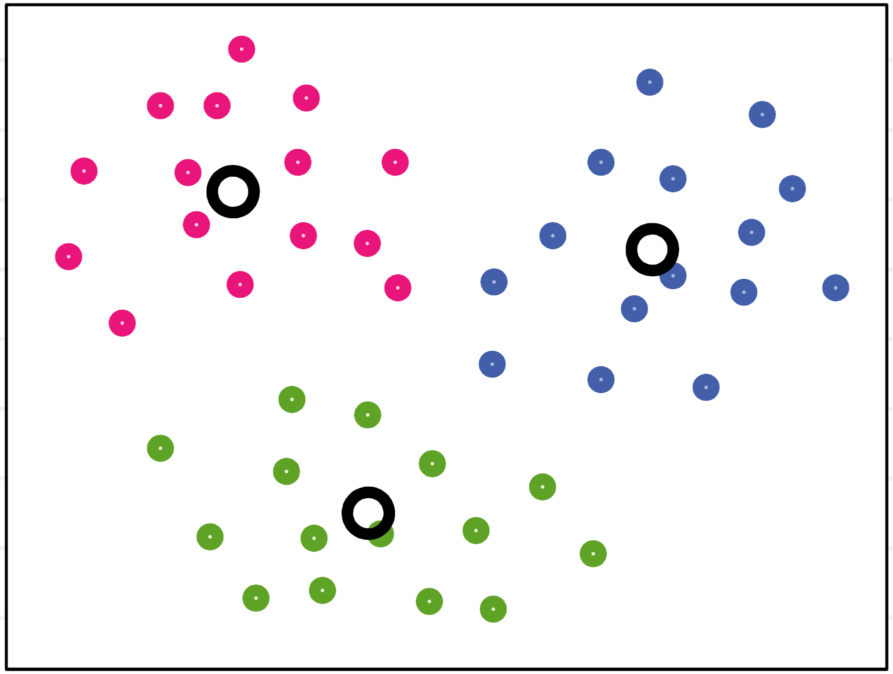
$y' = \arg\min\limits_{y} f(x,y)$ s.t. $(x,y) \in D$

if $y == y'$ ***return;*** *else* $y = y'$

End

k-means clustering





Task: split the points *{x}* into k clusters

Input points:

$$x_1, x_2 \dots x_M \in \mathbb{R}^n$$

$$\min_{c,n} \sum_{i=1}^{M} \|x_i - c_{n_i}\|_2^2$$

Cluster centers:

$$c_1, c_2 \dots c_K \in \mathbb{R}^n$$

Point assignments:

$$n_i \in \{1, 2, \dots K\}$$

Solving the k-means problem

$$\min_{c,n} \sum_{i=1}^{M} \| x_i - c_{n_i} \|_2^2$$ - "hard" problem

$$\min_{n} \sum_{i=1}^{M} \| x_i - c_{n_i} \|_2^2$$

Exact solution:

$$n_i = \arg\min_t \| x_i - c_t \|_2^2$$

(for each point pick the closest center)

$$\min_{c} \sum_{i=1}^{M} \| x_i - c_{n_i} \|_2^2$$

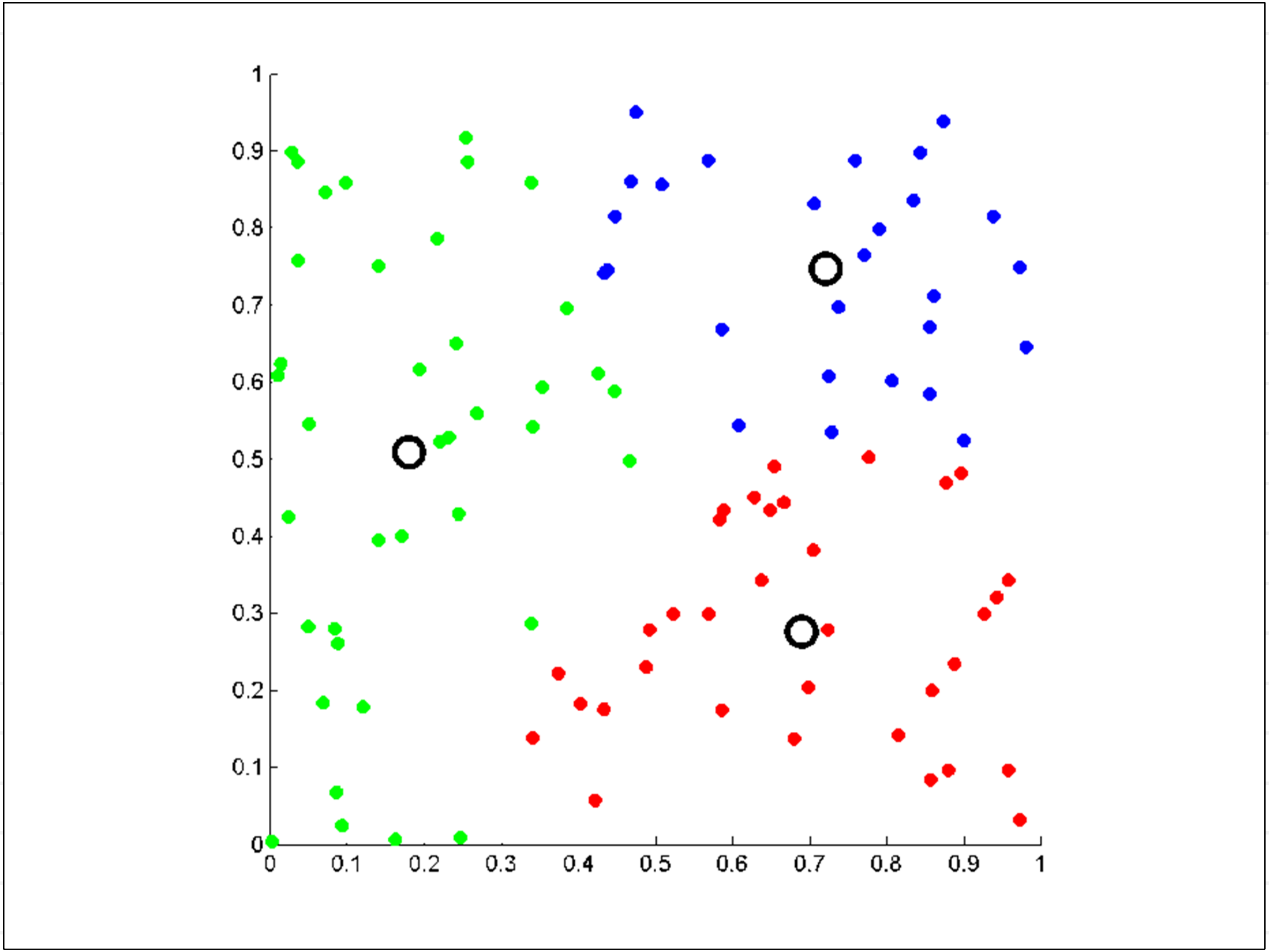
Exact solution:

$$N_t = \{ i \mid n_i = t \}$$

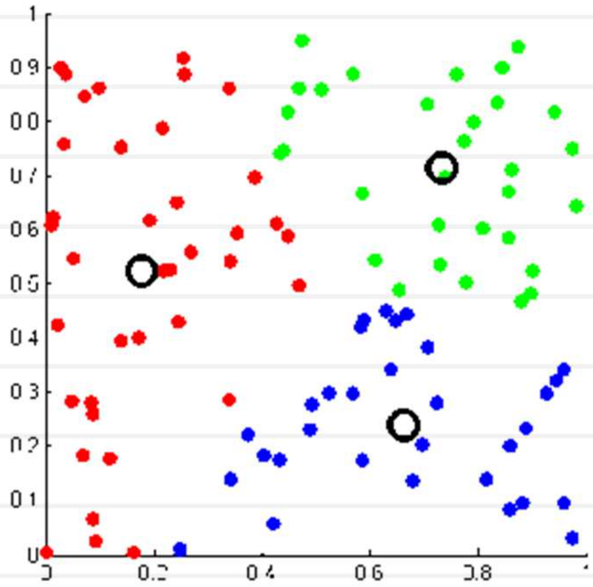
$$c_t = \frac{1}{|N_t|} \sum_{i \in N_t} x_i$$

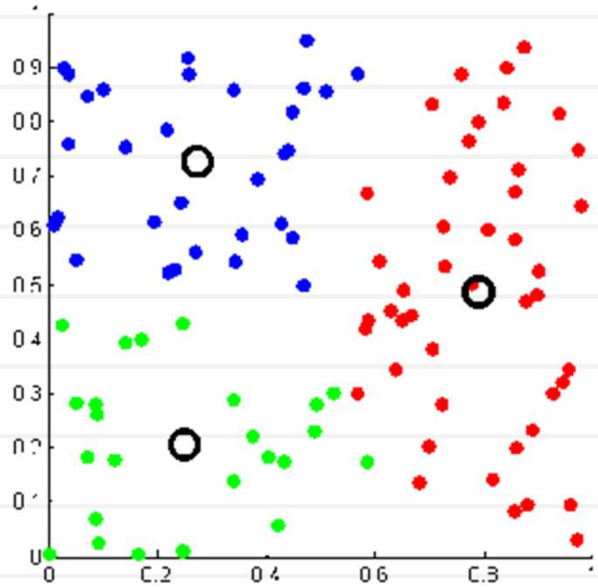
(average points that belong to each cluster)

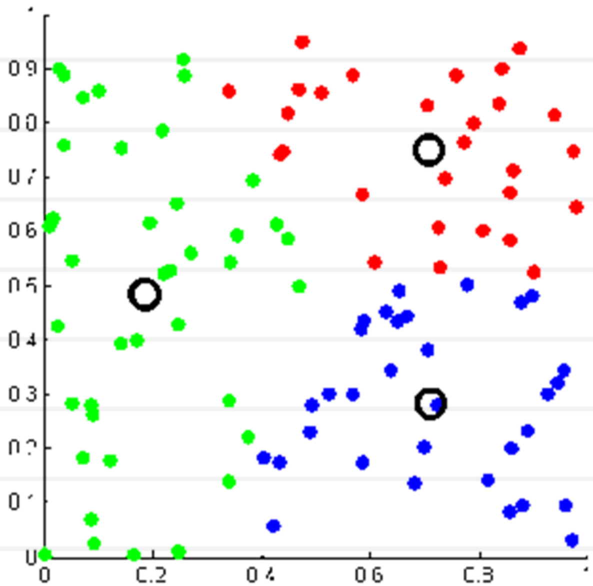
K-means example



k-means clustering







- Same input points
- Three different local minima
- Multiple restarts would help

Local optimization: summary

- Greedy/local search is typically the first thing to try
- Sometimes approximation guarantees are possible
- In rare cases, optimum is obtained
- Most complex systems with heterogeneous groups of variables are optimized using block-coordinate descent