Lecture 5: Circuit Partitioning – III

Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Programming Assignment #1

- **Implement FM partitioning algorithm**
  - https://github.com/tsung-wei-huang/ece5960-physical-design/tree/main/PA1
- **Two checkpoint dues, 9/7 and 9/14 23:59 PM**
  - https://github.com/tsung-wei-huang/ece5960-physical-design/issues/2
- **Final Due on 9/21 (Wed) 23:59 PM**
  - Upload your solutions to twhuang-server-01.ece.utah.edu
    - Account: ece6960-fall22
  - Place your source code + README under PA1/your_uid/
  - README should contain instruction to compile & run your code

Programming Assignment #1 (cont'd)

- **In addition to source code + README, upload a report with:**
  - A table showing your results of each benchmark
  - A section discussing what challenges you encounter
  - A section discussing how you overcome those challenges
    - Also discuss unsolved challenges
- **The report needs to be just a one- or two-page pdf**
  - No need to be lengthy …
- **Upload your report to the class GitHub page**
  - https://github.com/tsung-wei-huang/ece5960-physical-design/issues/1
  - Due 9/21 23:59 PM

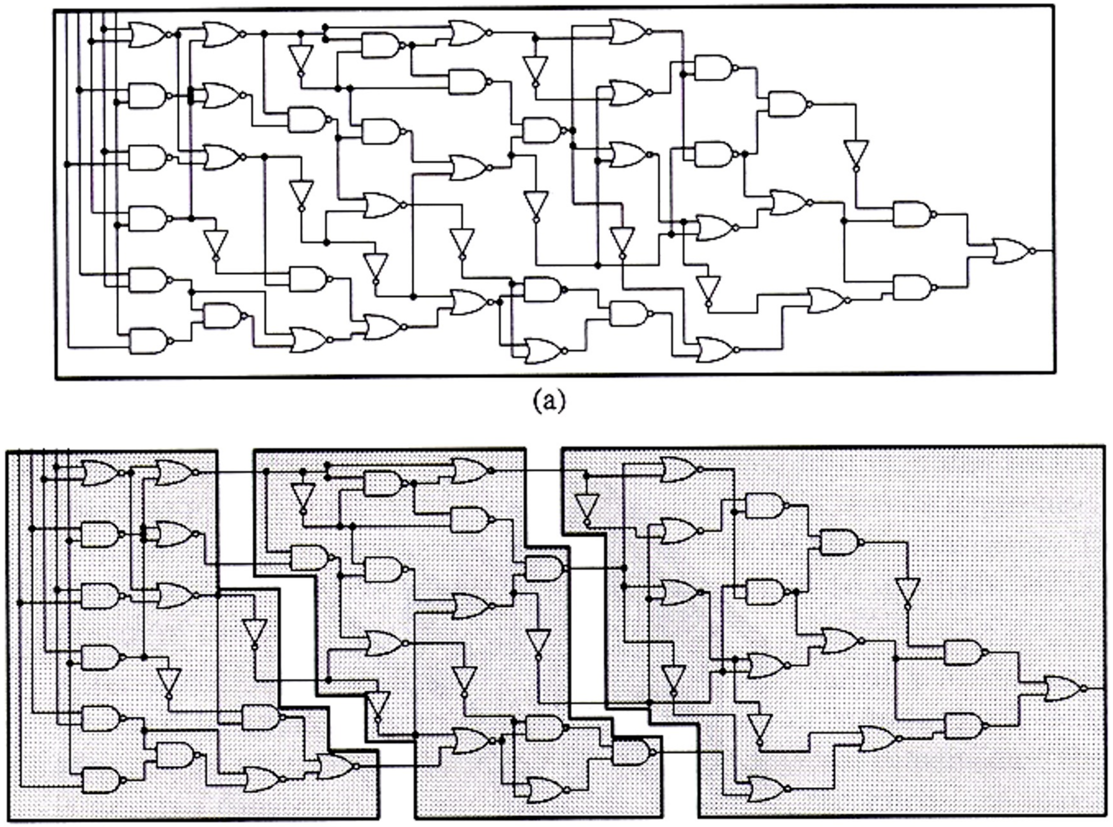
In-class Presentation: 9/14

- **Circuit partition research presentation on 9/14 (in class)**
  - George Karypis and Vipin Kumar, "Multilevel k-way Hypergraph Partitioning," *1999 ACM/IEEE DAC* – ***presented by W-L Lee***
  - Honghua Yang and Martin Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning," *1994 ACM/IEEE ICCAD* – ***presented by Randy***
  - Masahiro Tanaka, Kenjiro Taura, Toshihiro Hanawa, Kentaro Torisawa, "Automatic Graph Partitioning for Very Large-scale Deep Learning," *2021 IEEE IPDPS 2021* – ***presented by McKay***
- **Instructions**
  - Upload your pptx to https://github.com/tsung-wei-huang/ece5960-physical-design/issues/9
  - Template is available here: https://github.com/tsung-wei-huang/ece5960-physical-design/blob/main/Presentation/template.pptx

Mid-Autumn Festival

Recap: Circuit Partition

- **An essential step for reducing algorithm design complexity**
  - Divide and conquer (D&C)
- **Input**
  - A circuit graph
- **Output**
  - A set of partitioned subgraphs
- **Objective**
  - Minimize cross-connection



(a)

(b)

Recap: KL Algorithm

1. Pair-wise exchange of nodes to reduce cut size
2. Allow cut size to increase temporarily within a pass
3. Compute the gain of a swap

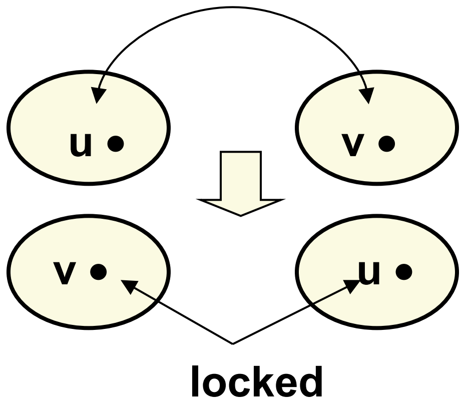
    Repeat

        Perform a feasible swap of max gain

        Mark swapped nodes "locked"

        Update swap gains

    Until no feasible swap



4. Find max prefix partial sum in gain sequence $g_1, g_2, \ldots, g_m$
5. Make corresponding swaps permanent
6. Start another pass if current pass reduces the cut size

Recap: FM Algorithm

- Moves are made based on object gain – extended from KL algorithms
- Object Gain: The amount of change in cut crossings that will occur if an object is moved from its current partition into the other partition
- A **pass** description

  While there is unlocked object

  1. Each object is assigned a gain
  2. Objects are put into a sorted gain list
  3. The object with the highest gain from the larger of the two sides is selected and moved
  4. The moved object is "locked"
  5. Gains of "touched" objects are recomputed
  6. Gain lists are resorted
- Repeat the pass until there is no improvement

State Space Search Problem

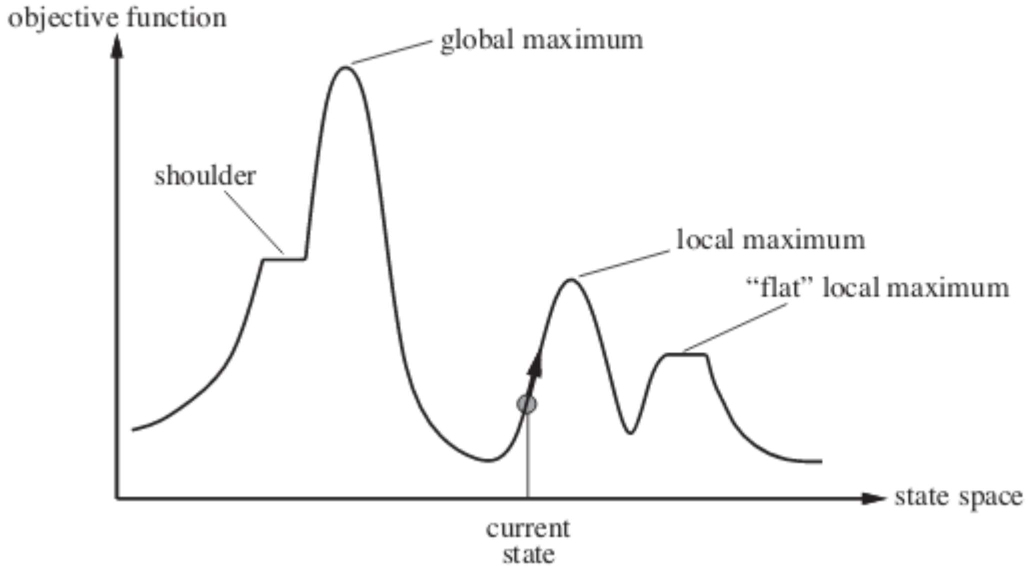
- Combinatorial optimization problems (like partitioning) can be thought as a State Space Search Problem
- A <u>State</u> is just a configuration of the combinatorial objects involved
- The <u>State Space</u> is the set of all possible states (configurations)
- A <u>Neighborhood Structure</u> is also defined (which states can one go in one step)
- There is a cost corresponding to each state (e.g., cut)
- Search for the min (or max) cost state (e.g., min-cut partition)

Greedy Algorithm

- A very simple technique for State Space Search Problem
- Start from any state to perform greedy, iterative improvement
- Always move to a neighbor with the min cost (assume minimization problem)
- Stop when all neighbors have a higher cost than the current state or no improvement can be made

Problem with Greedy Algorithm

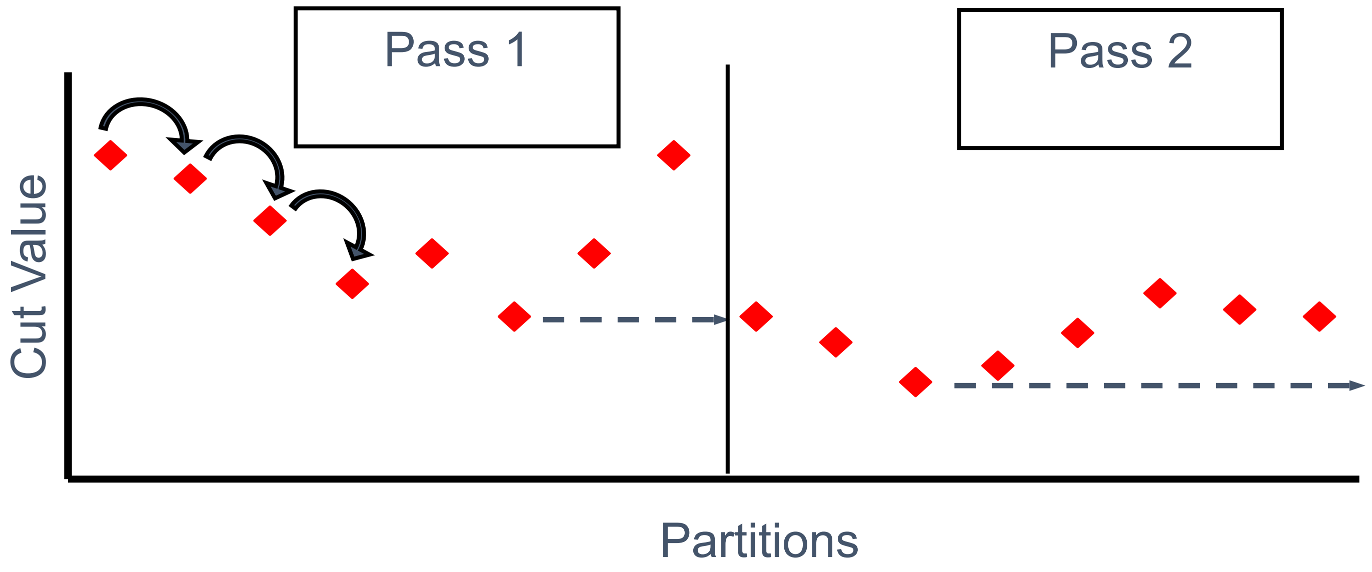
- Easily get stuck at local minimum with non-optimal solutions!
- Solution quality highly depends on the “initial solution”

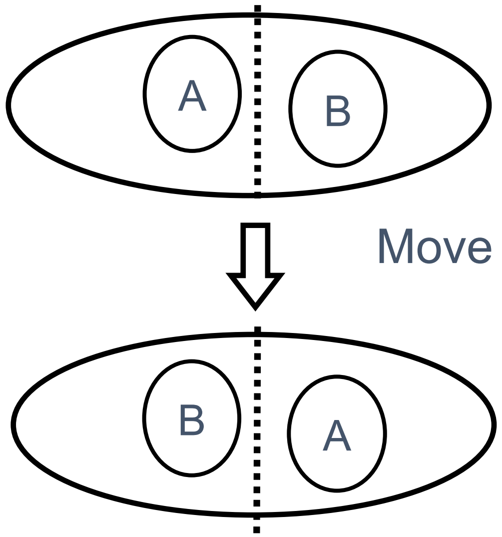


- Optimal only for convex (or concave for maximization) functions

Greedy Nature of KL and FM

- KL and FM are greedy algorithms





- Purely greedy if we consider a pass as a "move"

Simulated Annealing (SA) Algorithm

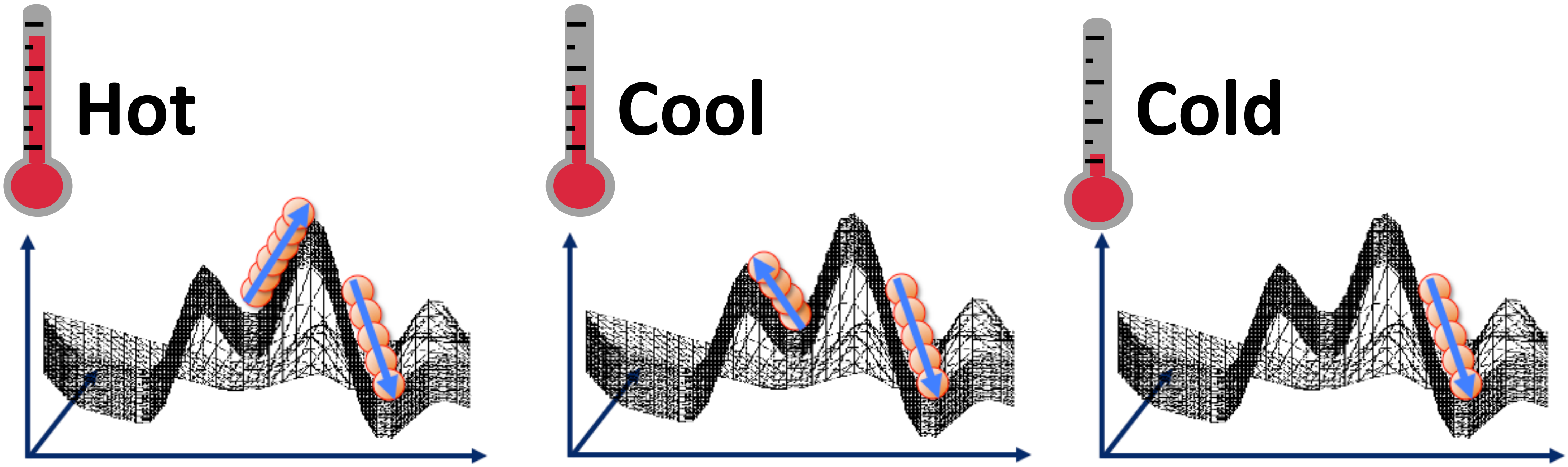
- **Very general search technique to avoid being trapped in local minimum by making probabilistic moves**
  - Kirkpatrick, Gelatt and Vecchi, "Optimization by Simulated Annealing", Science, 220(4598):498-516, May 1983
- **Almost same as old, greedy one, with two big changes**
  - Hill-climbing **T**, start **T=Hot=BIG**, slowly reduce **T** over many swaps
  - If a swap makes result become worse, **randomly accept it** with probability **P(ΔL, T)**

Basic Ideas of SA

- **Inspired by the *Annealing Process***
  - Attaining a min cost state in simulated annealing is analogous to attaining a good crystal structure in annealing
  - The process of carefully cooling molten metals in order to obtain a good crystal structure
  - First, metal is heated to a very high temperature
  - Then slowly cooled
  - By cooling at a proper rate, atoms will have an increased chance to regain proper crystal structure

Basic Ideas of SA (cont’d)

- **Non-zero probability for “up-hill” moves**
  - Avoid being trapped in local optima
  - Controlled by annealing schedule and solution improvement



Basic Ideas of SA (cont'd)

- **Assuming minimization, such probability depends on**
  - magnitude of the "up-hill" movement
  - total search time (i.e., implicitly told by the current temperature)

$$Prob(S \rightarrow S') = \begin{cases} 1 & \text{if } \Delta C \leq 0 \quad /* \text{ ``}down - hill\text{'' } moves */ \\ e^{-\frac{\Delta C}{T}} & \text{if } \Delta C > 0 \quad /* \text{ ``}up - hill\text{'' } moves */ \end{cases}$$

where

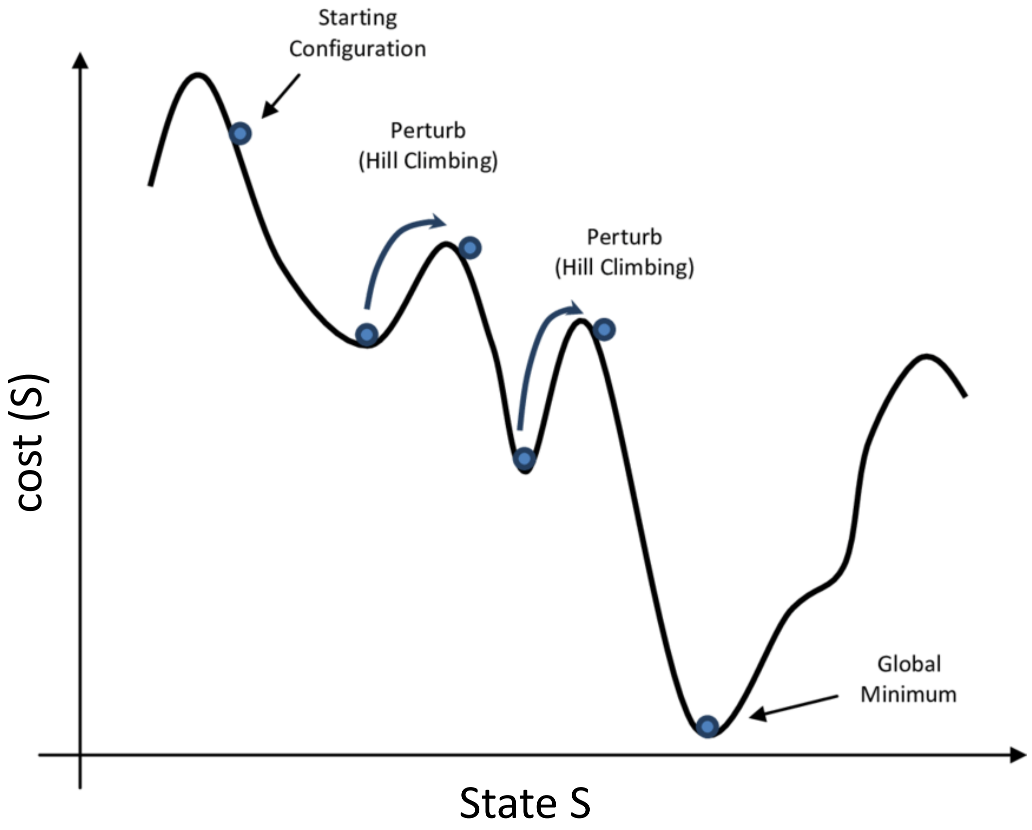
- $\triangle C = cost(S') - cost(S)$
- T: Current temperature
- Annealing schedule: $T = T_0, T_1, T_2, \ldots$, where $T_i = r^i T_0$, $r < 1$.

Things to Consider in SA

- **When solving a combinatorial problem, we have to decide**
  - The state space
  - The neighborhood structure
  - The cost function
  - The initial state
  - The initial temperature
  - The cooling schedule (how to change temperature)
  - The freezing point

SA Algorithm Pseudocode

```
begin
Get an initial solution S;
Get an initial temperature T > 0;
while not yet "frozen" do
    for i=1 to P do
        Pick a random neighbor S' of S;
        △← cost(S') – cost(S);
      /* down hill move */
        if △  0 then S←S'
      /* uphill move */
        if △ > 0 then S←S' with probability;
  T←rT ; /* reduce temperature */
return S
end
```



Common Cooling Schedules

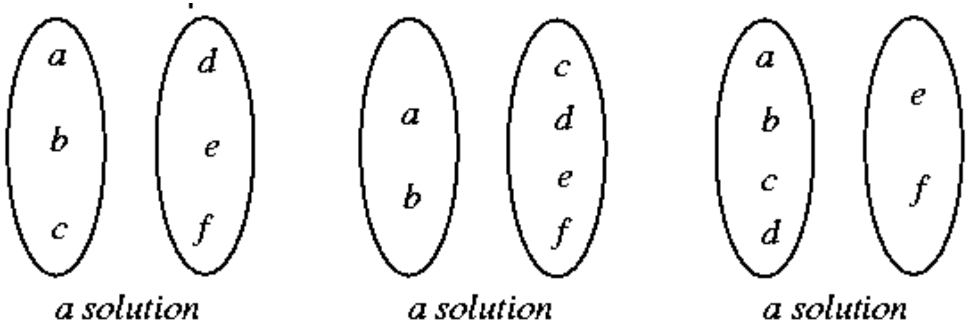
- **Initial temperature, cooling schedule, and freezing point are usually experimentally determined**
  - Parameter tuning is very critical in getting a good SA result
- **In practice, we can do the following cooling schedules**
  - $t = \alpha t$, where α is typically around 0.95
  - $t = e^{-\beta t}$, where β is typically around 0.7
  - …

Basic SA Algorithm Structure

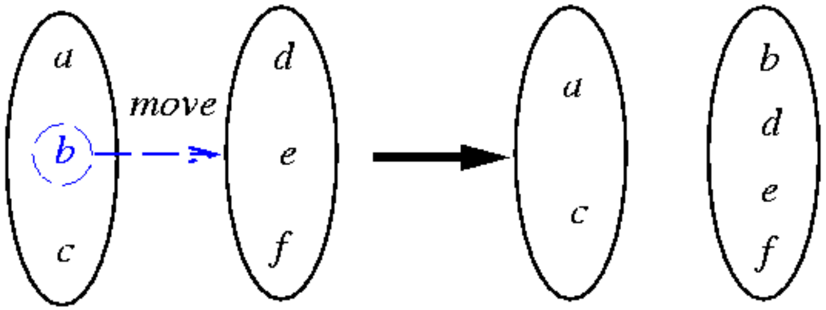
- **Solution space**
  - Represent the state size (e.g., possible solutions) we define for the optimization problem
- **Neighborhood structure**
  - Represent the state change after applying a local move
  - We don't want radical or sharp change! (why?)
- **Cost function**
  - Represent the objective of the optimization problem
- **Annealing schedule**
  - Represent the temperature decreasing process

SA-based Partitioning Algorithm

- **Solution space**: set of all partitioning solutions



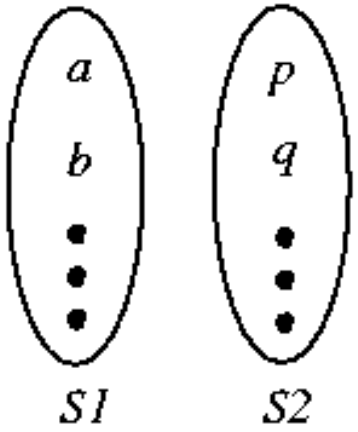
- **Neighborhood structure:** move one cell to another partition



Randomly move one cell to the other side

SA-based Partitioning Algorithm (cont'd)

- **Cost function**: $f = C + \lambda B$
  - C: the solution cost as used before
  - B: a measure of balance
  - λ: a constant



$$B = (|S1| - |S2|)^2$$

- **Annealing schedule**
  - $T_n = r^n T_0$, $r = 0.9$
  - At each temperature, either
    1. There are 10 accepted moves/cell on the average, or
    2. # of attempts = 100 * total # of cells.
  - The system is "frozen" if very low acceptances at 3 consecutive temperatures

Evaluation of SA-based Partitioning

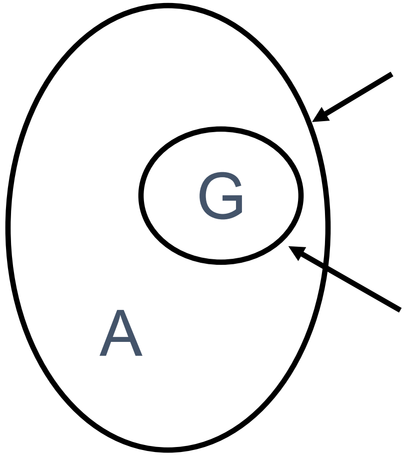
- An extensive empirical study of Simulated Annealing versus Iterative Improvement Approaches have been studied over the past decades
- Conclusion: SA is a *competitive* approach, getting better solutions than KL/FM for random graphs
- Remarks about SA:
  - Netlists are not random graphs, but sparse graphs with local structure
  - Yet, SA is often too slow—so KL/FM variants are still most popular
  - Multiple runs of KL/FM variants with random initial solutions may be preferable to SA

Common Questions about SA

- Does annealing always find the perfect, **best global** optimum?
  - **NO**. It is just good at avoiding a lot of local minimums
- Does annealing work on **every type** of optimization problem?
  - **NO**. But it does work on many optimization problems–it is not always the most efficient option
- Is annealing always **slow**– doing all those many swaps over many temperatures?
  - **NO.** Lots of engineering tricks to speed it up
- Do I have to set all the parameters by **trial and error**?
  - **YES/NO**. There are fancy adaptive techniques to determine these automatically but they could also be problem-dependent

The Use of Randomness

- For any partitioning problem:



All solutions (state space)

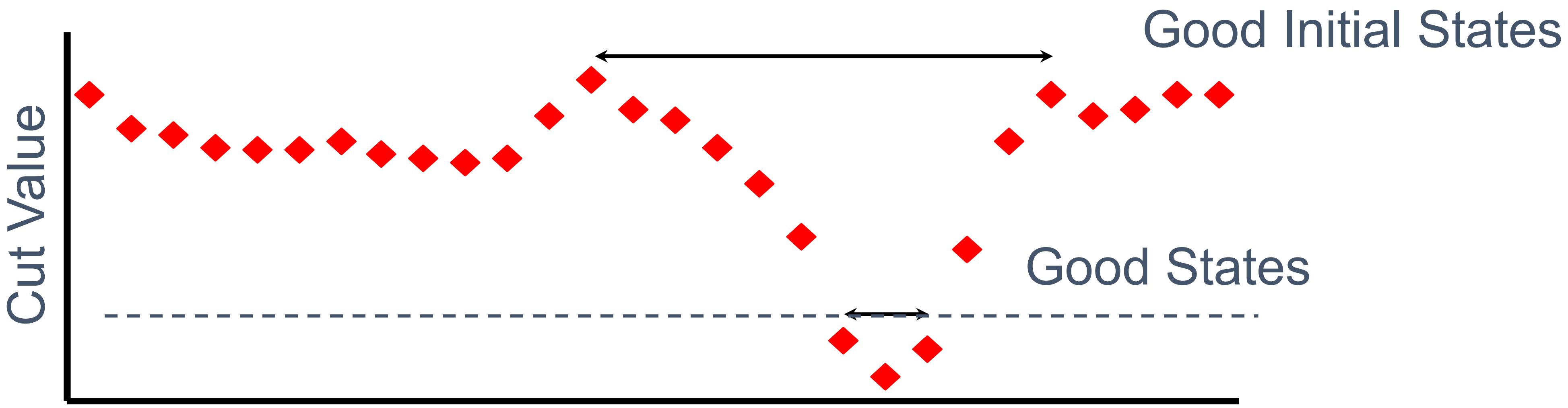
Good solutions

This is actually NOT bad!

- Suppose solutions are picked randomly
- If $|G|/|A| = r$, Pr(at least 1 good in 5 trials) = $1-(1-r)^5$
- If $|G|/|A| = 0.001$, Pr(at least 1 good in 5000 trials) = $1-(1-0.001)^{5000} = 0.9933$

Adding Randomness to KL/FM

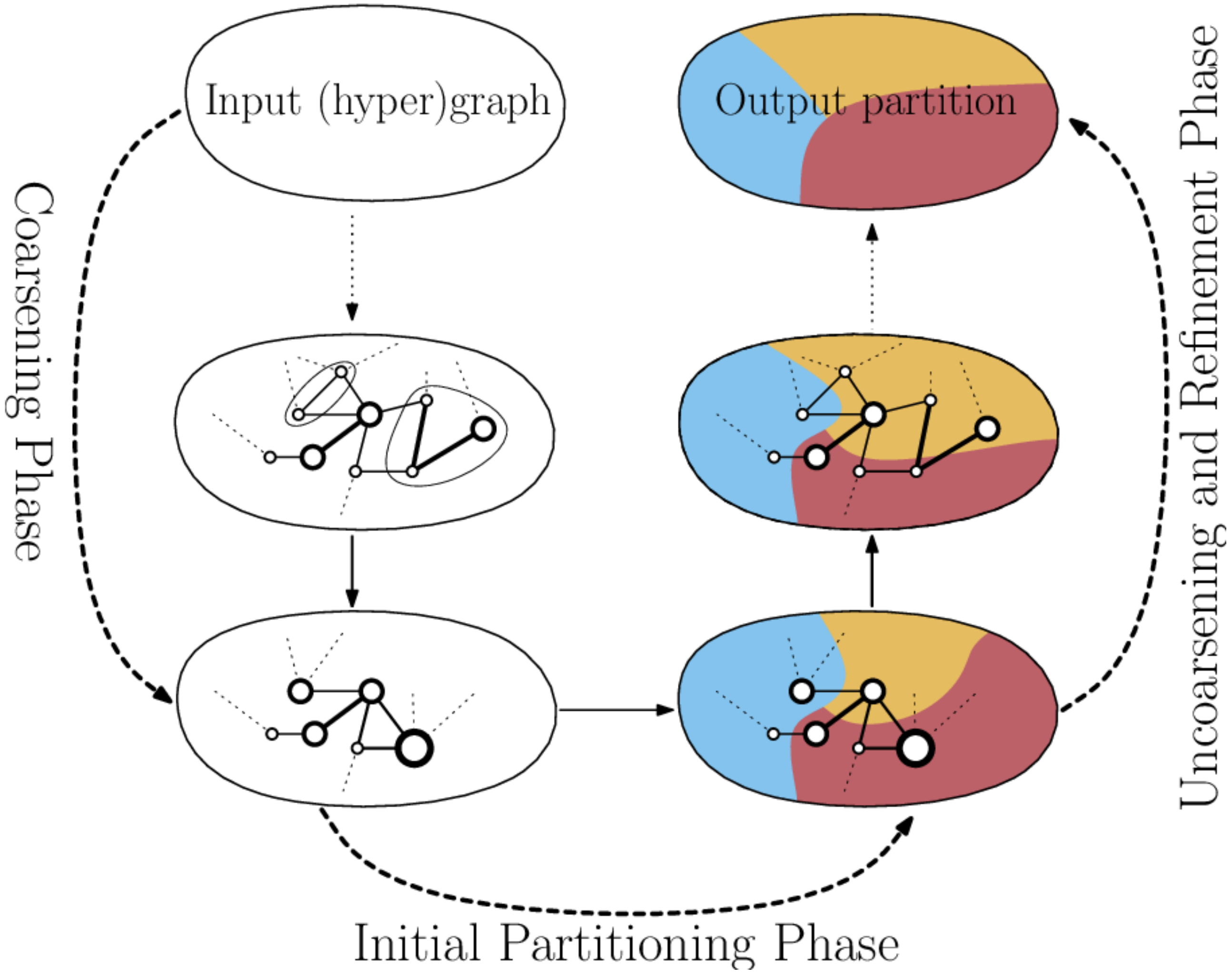
- **In fact, # of good states are extremely few**
  - Therefore, r is extremely small
- **Need extremely long time if just picking states randomly**
  - Therefore, we can running KL/FM variants several times with random initial solutions



Other Partitioning Approaches

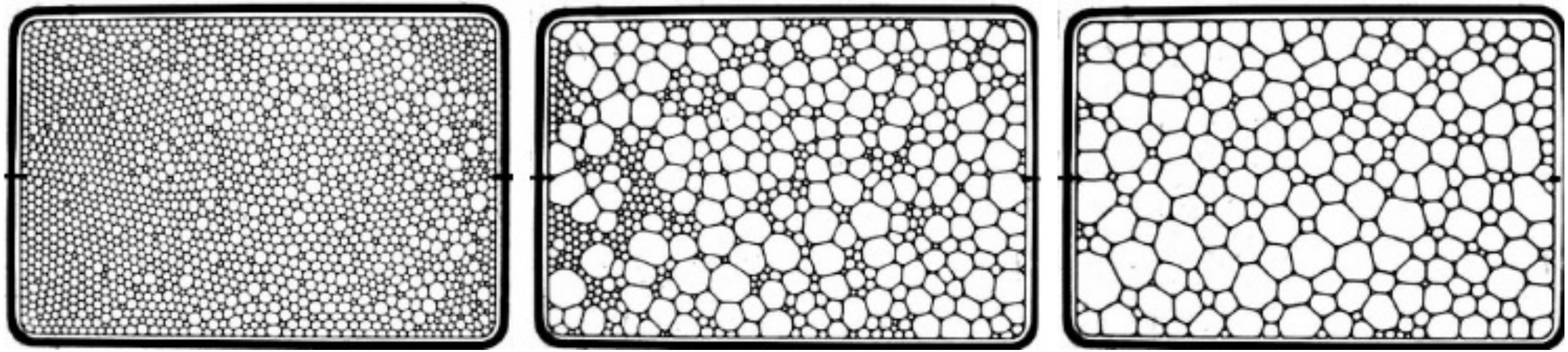
- KL/FM-SA Hybrid: Use KL/FM variant to find a good initial solution for SA, then improve that solution by SA at low temperature
- Tabu Search
- Genetic Algorithm
- Spectral Methods (finding Eigenvectors)
- Network Flows
- Quadratic Programming
- …

Multilevel Partitioning Techniques



Motivation of Coarsening

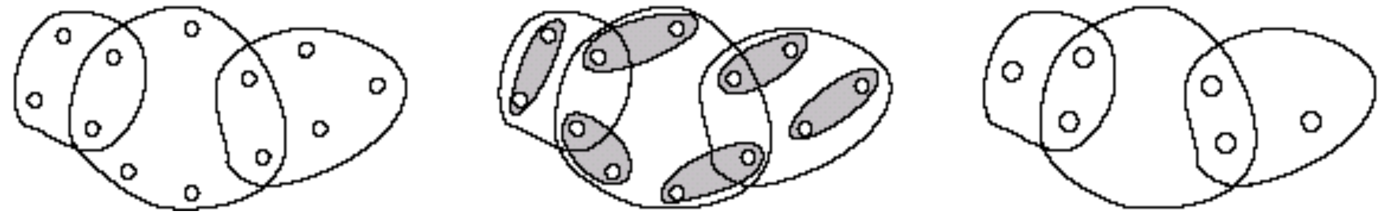
- **Find a small, representable graph of the original one**
  - Preserve the structure of the original graph
  - Preserve the similarity to the original graph



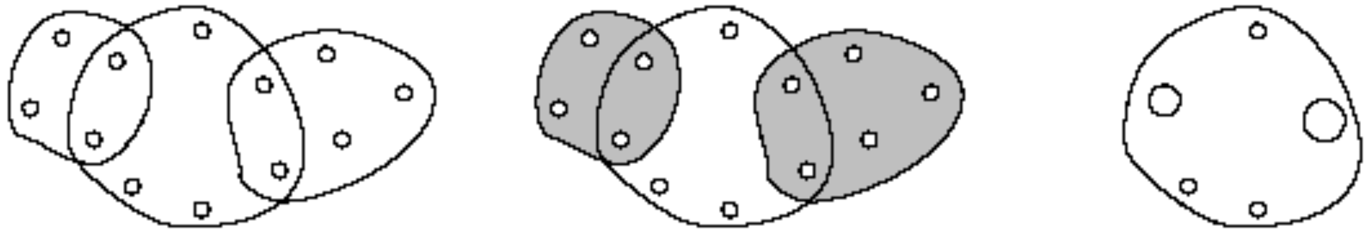
- **Reduce complexity by running on the coarsened graph**

Coarsening Phase

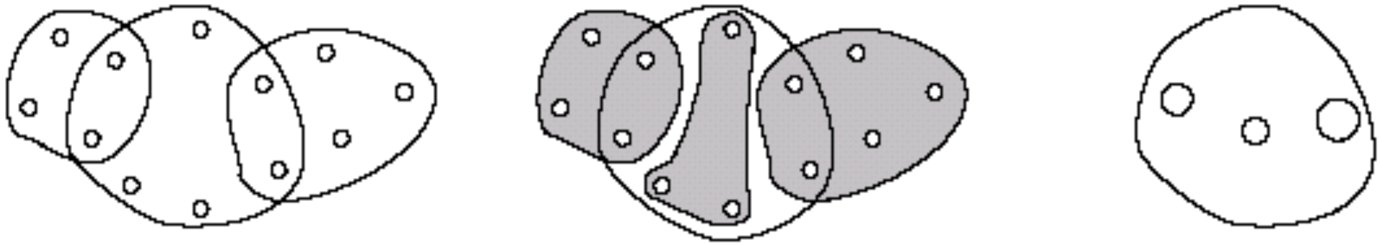
- Edge coarsening



- Hyperedge coarsening (HEC)



- Modified hyperedge coarsening (MHEC)



Uncoarsening Phase

- **Based on FM with two simplifications**
  1. Limit number of passes to 2
  2. Early-Exit FM (FM-EE), stop each pass if *k* vertex moves do not improve the cut
- **Hyperedge Refinement**
  - Move a group of vertices between partitions so that an entire hyperedge is removed from the cut

hMETIS Algorithm: A Mixed Strategy

- George Karypis and Vipin Kumar, "Multilevel k-way Hypergraph Partitioning," *1999 ACM/IEEE DAC* (will be one our in-class research paper presentation)
  - https://course.ece.cmu.edu/~ee760/760docs/hMetisManual.pdf
- **hMETIS-EE$_{20}$**
  - 20 random initial partitions
  - with 10 runs using HEC for coarsening
  - with 10 runs using MHEC for coarsening
  - FM-EE for refinement

Summary

- We have discussed problems of greedy partitioning algorithms
- We have discussed simulated annealing (SA) algorithm
- We have discussed SA-based partitioning algorithm
- We have discussed multi-level partitioning algorithms