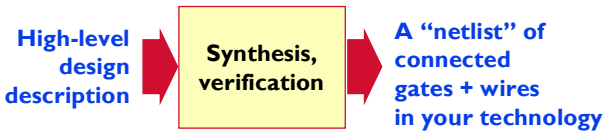
## (*Lec 12*) 1s & 0s -> Rectangles: Physical Design

- What you know...
  - Boolean, 1s and 0s stuff: synthesis, verification, representation
  - This is what happens in the "front end" of the ASIC design process




- What you don't know...
  - The "back end" problem -- turning these into IC masks
  - Basically a gates --> rectangles problem
  - Called "layout" or "physical design"



## Copyright Notice

## Where Are We?

- Physical design--how to geometrically *layout* gates in a netlist?

| | M | T | W | Th | F | |
|---|---|---|---|---|---|---|
| Jan | 11 | 12 | 13 | 14 | 15 | 1 |
| | 18 | 19 | 20 | 21 | 22 | 2 |
| | 25 | 26 | 27 | 28 | 29 | 3 |
| Feb | 1 | 2 | 3 | 4 | 5 | 4 |
| | 8 | 9 | 10 | 11 | 12 | 5 |
| | 15 | 16 | 17 | 18 | 19 | 6 |
| | 22 | 23 | 24 | 25 | 26 | 7 |
| Mar | 1 | 2 | 3 | 4 | 5 | 8 |
| | 8 | 9 | 10 | 11 | 12 | 9 |
| | 15 | 16 | 17 | 18 | 19 | 10 |
| Break | 22 | 23 | 24 | 25 | 26 | 11 |
| | 29 | 30 | 31 | 1 | 2 | 12 |
| Apr | 5 | 6 | 7 | 8 | 9 | 13 |
| | 12 | 13 | 14 | 15 | 16 | 14 |
| | 19 | 20 | 21 | 22 | 23 | 15 |
| | 26 | 27 | 28 | 29 | 30 | 16 |

Introduction
Advanced Boolean algebra
JAVA Review
BDDs
Formal verification
2-Level logic synthesis
Multi-level logic synthesis
Technology mapping
***Placement***
Routing
Partitioning
Static timing analysis
Geometric data structures
Test and ATPG



## Handouts

- Physical
  - Lecture 12 -- ASIC Layout: Placement
  - Kirkpatrick's simulated annealing paper
- Electronic
  - Nothing new...

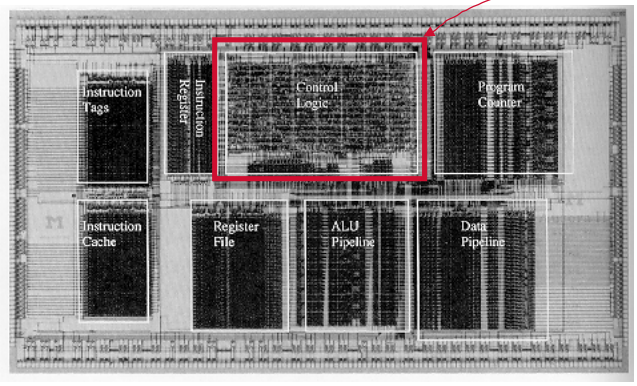
## Readings

- Sarrafzadeh & Wong
  - Chapter 1 on layout background: 1.1, 1.2, 1.4, 1.5
  - Chapter 2 on placement: 2.3.1-2.3.3
- Kirkpatrick paper
  - Original paper describing simulated annealing
  - A bit of a physics flavor, but still emminently readable



## Physical Design for ASICs

- We are focusing on a particular "niche" in layout
  - Row-based standard cells for semi-custom applications
  - Example: U Michigan GaAs MIPS CPU [ISSCC93]




This part is rows of so-called "standard cells" used for the control logic

## ASIC-related Terminology: Buzzword Lexicon

- Some terms
  - *Custom layout:* you do it all by hand. Think of this as design at the transistor level. Also called (derisively) "polygon pushing".
  - *Semi-custom layout:* you pick up some pre-laid-out pieces from some library and then arrange these to do your complete layout
  - *ASIC, Application-Specific Integrated Circit:* a semi-custom IC designed to do one thing (eg, an MPEG decoder) as opposed to something arbitrrily programmable, like a CPU
  - *ASSP, Application-Specific Standard Product:* an ASIC that works to a widely-established usage spec, so that many companies build cmpatible parts to this spec. Example: ethernet chip, many telecom parts.
  - *ASIP, Application-Specific Instruction Set Processor:* an ASIC that's programmable to a range of tasks, with a custom instruction set (ie, not like a Pentium or Sparc). Examples: many DSP tasks use ASIPs

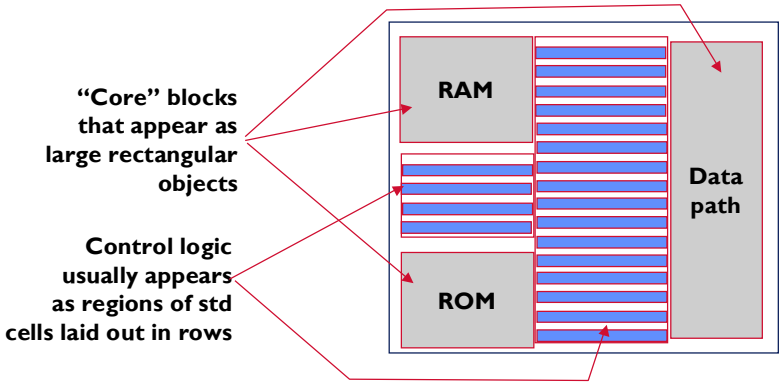


## ASIC-related Terminology: Buzzword Lexicon

- Some more terms: what ASICs are made out of
  - *Standard cell:* the smallest, most common thing in your library of pre-laid-out stuff. Basically, a gate or FF. All the stuff in your technology library for Tech Mapping has a layout in this library.
  - *Core-cell:* a huge thing in your library of pre-laid-out stuff, like an entire CPU core. Also sometimes called a *mega-cell*, but often just a *core*, or (modern buzzword) *intellectual property (IP)*
  - *Row-based layout:* the usual way you arrange standard cells to lay them out to make a final design
- Many modern designs are System-on-Chip (SoC) designs
  - Use standard cells for random logic, import cores for complex functions (like CPUs), and add memories to make single-chip systems

## System On Chip Style

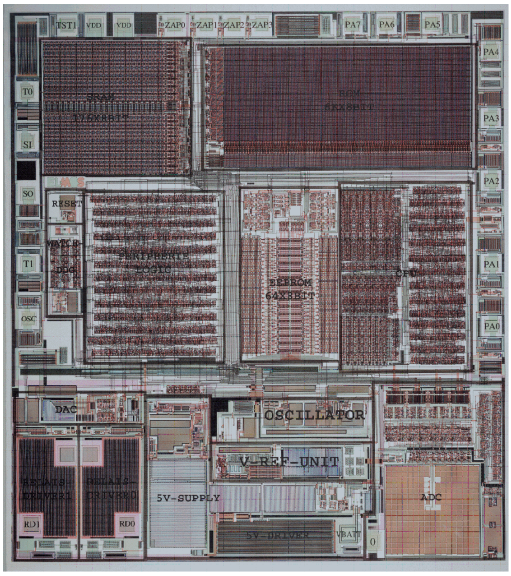
- Abstract example



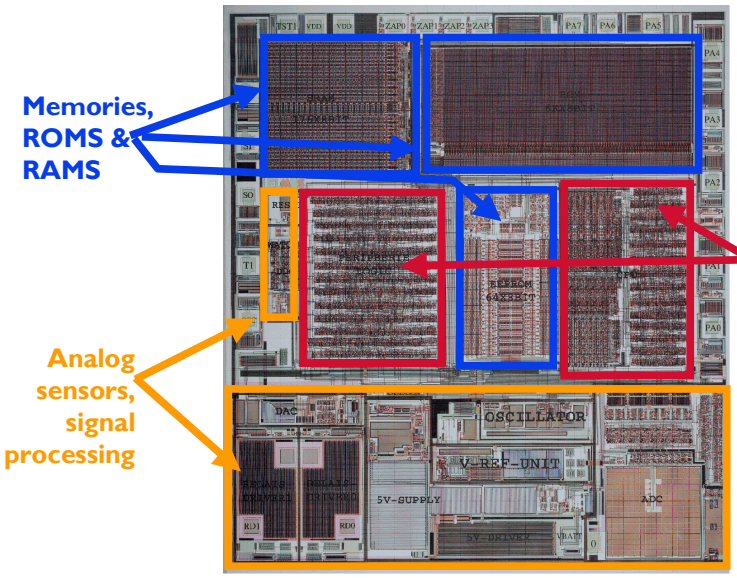



## Real Example: Mixed-Signal ASIC

- Mixed-signal == analog + digital; this is an automotive ASIC

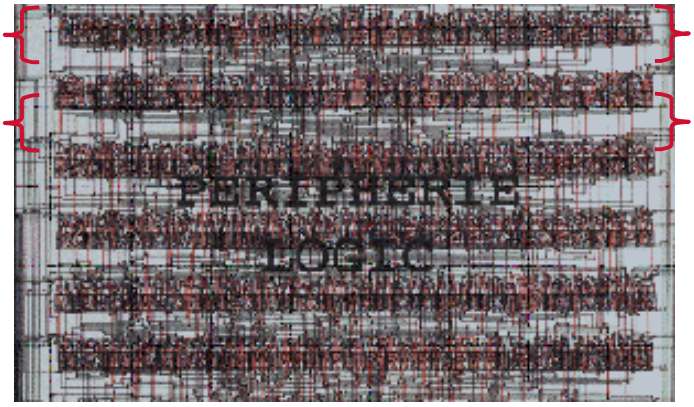


## Real Example: Mixed-Signal ASIC

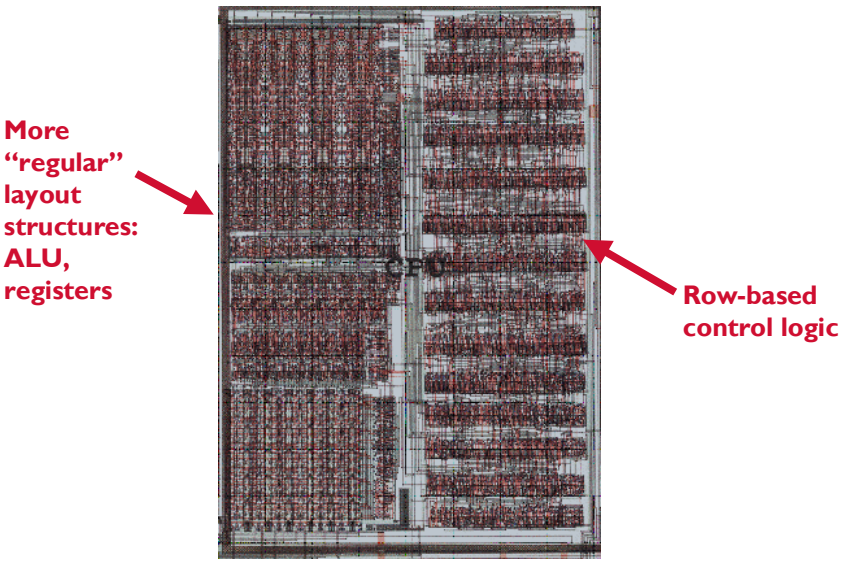





## Zoom: Row-Based Control Logic



## Zoom: CPU Is Itself a Hierarchy of Cells






## ASIC Design Methodology

- Methodology == ?
  - The sequence of CAD tools, design representations, design steps you use to go from an idea to silicon
  - Includes both synthesis steps, to make designs more detailed, and verification steps, to check correctness
- Modern methodology (simplified)
  - **Simulation /validation:** spec design as executable code (eg, Verilog), simulate to try test cases
  - **Synthesis:** high-level synthesis (eg, from Verilog) and logic synthesis, to go from high-level spec to gates
  - **Mapping:** to get the gates onto your implementable library
  - **Formal verification:** used where possible to prove equivalence
  - **Physical Design:** floorplan the chip, place the blocks/gates, route them
  - **Timing verification:** (actually happens at all steps), check to make sure you meet timing goals (ie, MHz)

## Aside: From Methodology to Intellectual Property

- Intellectual property (IP)
  - Something you can buy for $$$ that embodies some design expertise
  - Easier to buy it than to build it
- Historical forms of IP
  - **Chips**: You buy the hardware, plug it into a board, wire it up and go
  - **Software**: You buy it, load it, boot it, run it.
- Evolving forms of IP
  - **Soft IP:** buy the Verilog source code. Synthesize it yourself.
  - **Firm IP:** buy the gate/block level netlist. Do your own mapping, layout
  - **Hard IP:** buy the actual layout. May have to "adjust" to your fab
  - Ability to buy tools that do synthesis/layout is one big factor driving IP

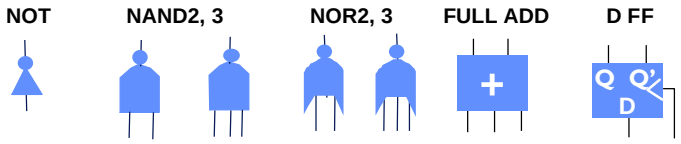


## What Are We Doing Now...?

- Big steps in physical design "backend" of ASIC methodology
- Synthesis steps
  - Partitioning
  - Placement
  - Routing
- Verification steps
  - Static timing analysis
  - Mask-level design rule checking

# Physical Design for ASICs

- Example of simple standard cell library

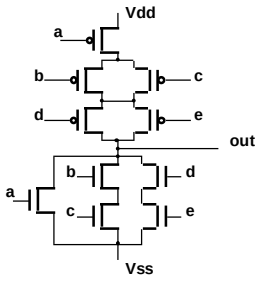


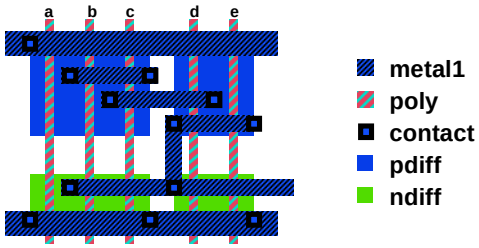

- As layouts...
  - Each a little rectangle of different mask layouts
  - Usually a common height (Y direction)
  - Varying widths, depending on number of IOs
  - In this style, the pins are available at the top and the bottom of each cell




# Physical Design for ASICs

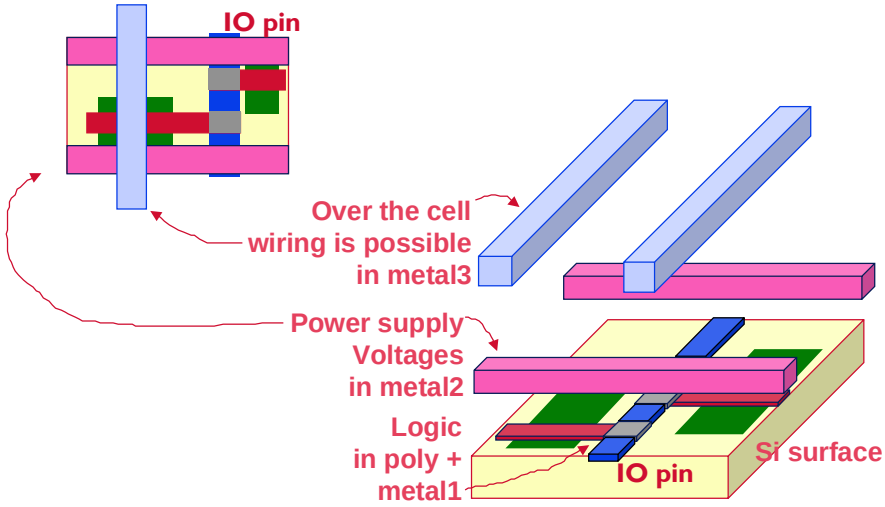
- A very simple cell
  - For static CMOS, inputs on polysilicon, P devices on top, N on bottom



## Physical Design for ASICs

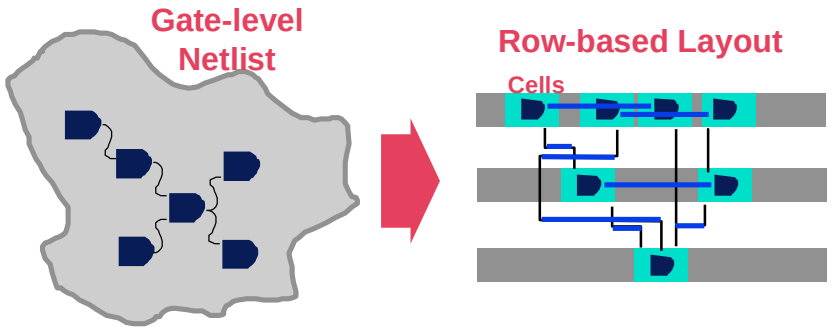
- Closer look at a bit more complicated cell style





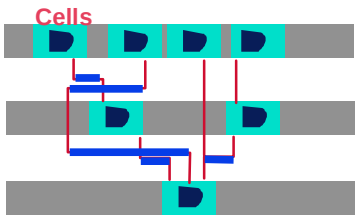

## Physical Design for ASICs

- How do these things get "laid out"?
  - In rows, with wiring in between the gates, and over the gates
  - Abutting the cells left-right connects the power and ground lines
  - Spacing between rows called *channels*; in older designs you put the wiring here; in more modern designs of the wiring goes *over* cell rows



## Aside: Wiring Channels Between Rows

- Basically, they are going away
  - When we had only 2 wiring layers (poly + M1) *all* wiring was in channel




  - When we got 2 metals--M1, M2--still had channels, we just stopped using poly to route outside of the gates them selves
  - Note that wiring layers usually have a *preferred* direction -- H or V; router can violate this if it helps to avoid some vias, but it tries not to do this too much



## Aside: About Channels...

- Cont.
  - When we got 3 metals--M1 M2 M3--started to see some wires go over the cells themselves, rather than reserve space inside cells or between cells. So, the channels got a lot smaller, eg


  - At 4+ metals, the channels basically go away, and you just route everything over the cells themselves. You pack the cells at minimumdesign-rule-correct distances.

## Physical Design for ASICs

- What exactly are we going to look at?

1. Placement
   - Once you know the gates on a particular IC, where do you put them?
2. Routing
   - Once you know where the gates are, how do you connect the wires?
3. Partitioning
   - Take a large gate netlist, divide into smaller pieces to fit on a set of Ics
4. Logical timing & electrical timing analysis
   - You have gates, you (maybe) have wires; how fast will it go?
5. Representation
   - How do you deal efficiently with 100,000,000 rectangles?
6. DRC / Extraction
   - OK, you got it laid out. Is it *right*? Does it implement what you want?



## 18-360 versus 18-760

- There is some overlap

| 18-360 | 18-760 |
|---|---|
| Kernighan-Lin partitioning | K&L + Fiduccia-Mattheyses algorithm |
| Annealing for placement | More annealing |
| Maze routing | More maze routing |
| Channel routing | Nope-- useful but no time |
| --- | Geometric data structures |
| --- | DRC & extraction |
| --- | Static timing analysis |
| --- | Electrical timing analysis |
| Test, fault simulation | Way less test: just D algorithm |

## ASIC Placement by Simulated Annealing

- What you know about layout
  - Probably not much, at this point...
- What you don't know about *placement*...
  - Placement: which gates go where on the chip
  - Annealing: powerful, general method of iterative improvement for combinatorial optimization problems like this



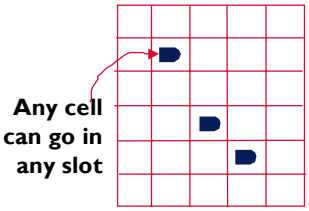
## ASIC Placement: Problem

- What are we trying to do with placement?
  - Input: a netlist of connected gates and nets
  - Output: exact location on the chip of each gate
  - Optimization: *make sure we can connect all the wires*
- Is this hard?
  - Yes. A bad placement can require *dramatically* more wiring.
  - More wiring is bad, since it consumes space we might have used for more gates...
  - ...and long wires have more delay, so affects overall speed too.
  - If your placement is *very* bad, the next tool in the layout flow--the router--may not even be able to find paths for all the wires.

## ASIC Placement: Issues

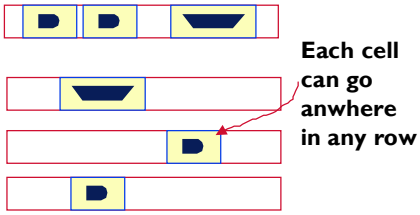
- Layout model
  - What do we know about geometric *shapes* of objects we are placing?
  - What constraints do we have on *where* they are allowed to go?

Simplest model:
all objects are "points"
placed in a simple grid




We use this one, just
because it is simplest to do...

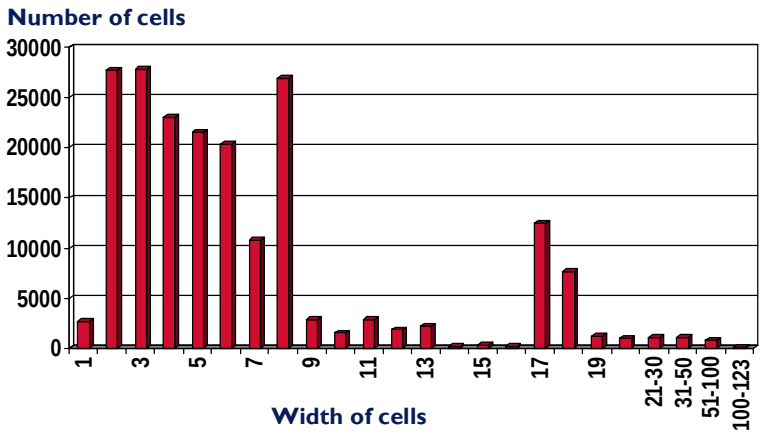
More realistic ASIC model:
all objects are rectangles of
varying width, same height,
placed in rows with variable separation





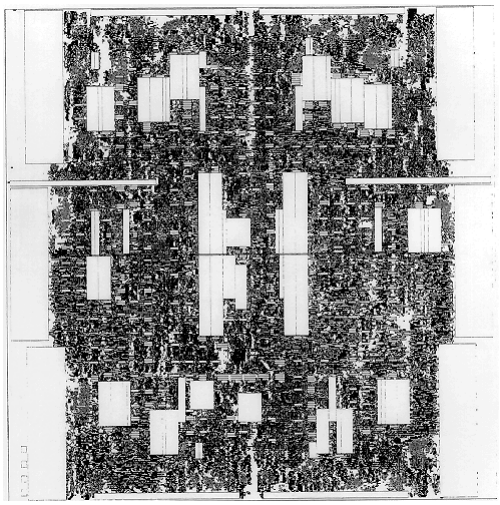

## Reality Check: Row-Based Layout

- The row-based objects *really* do come in different widths
  - "Width" you can think of as "how many IO pins wide"
  - Example: 200K gate IBM ASIC [J. Vygen, DAC98]



## Reality Check: Row-Based Layout

- ..and, you do still have to deal with random logic + big blocks
  - Blocks are "macros" like memories, registers, etc
  - From [Vygen DATE98], 200K gates + blocks

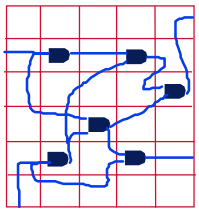




## ASIC Placement: Issues

- Wiring optimization
  - What do we mean "make it possible to route all the wires"?
  - We must translate this into a concrete goal for the placer.
  - Classical goal:

- New problem
  - How do we estimate the required total wirelength for a placement?
  - This is our estimate of of the "quality" of any candidate placement



# ASIC Placement: Wirelength Estimation

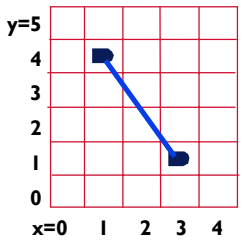
- Some facts
  - You have to estimate the total wirelength because its too expensive in CPU time (usually) to really call the routing tool for each wire
  - So, the "estimator" is supposed to give a reasonable guess for the wirelength, but be really quick to compute

- Wirelength estimators
  - Many many different types
  - Depend on what assumptions you can make about how the wires will actually get routed in the final ASIC layout
  - Also depends on how much CPU time you can afford
  - Let's look at a few classical strategies



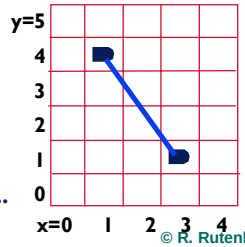
# Wirelength Estimation

- Euclidean estimation
  - For a 2-point net, just the hypotenuse of the triangle.
  - Problem: nobody really allows wires at arbitrary angles in most chips




Estimate is:

- Manhattan estimation
  - For a 2-point net, just the sum of the legs of triangle
  - (Name from pt-to-pt distance measured by NY cab drivers)
  - Perfectly OK for 2 point nets...

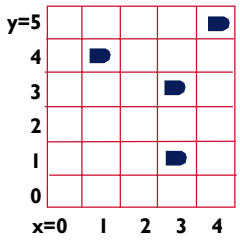



Estimate is:

## Wirelength Estimation

- What happens if >2 endpoints on the nets?

- Several options
  - Can use the simple trick of putting a 2-pt connection between *all* pairs of points...
  - ..but this dramatically overestimates the necessary wirelength

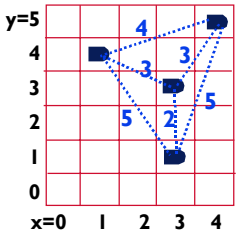



Estimate is:



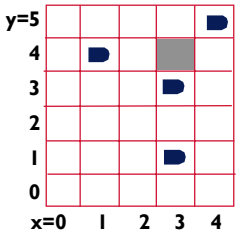
## Wirelength Estimation

- Better idea
  - Take the subset of those connections that has minimum overall length, but touches every point
  - Called "minimum spanning tree" -- $O(N^2)$ algs to get it for N points




Estimate is:

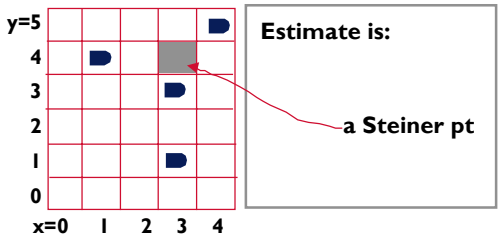
- Problems
  - It still overestimates the wire needed, since it assumes wire is made of of discrete point-to-point connections




Estimate is:

## Wirelength Estimation

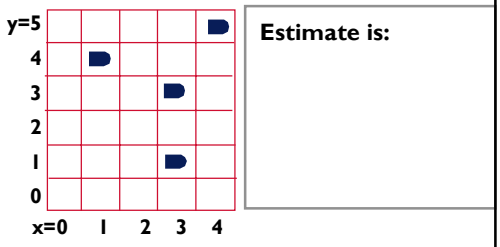
- OK, how would a real router tool do it?
- As a "Steiner tree"
  - Difference is the Steiner tree can have connection points at *arbitrary* places, not just at the spots where there are endpoints of net
- Problem
  - Getting an optimal Steiner tree is NP Hard, ie, exponentially hard in general case.
  - There are good heuristics, though, but its still expensive to do well.





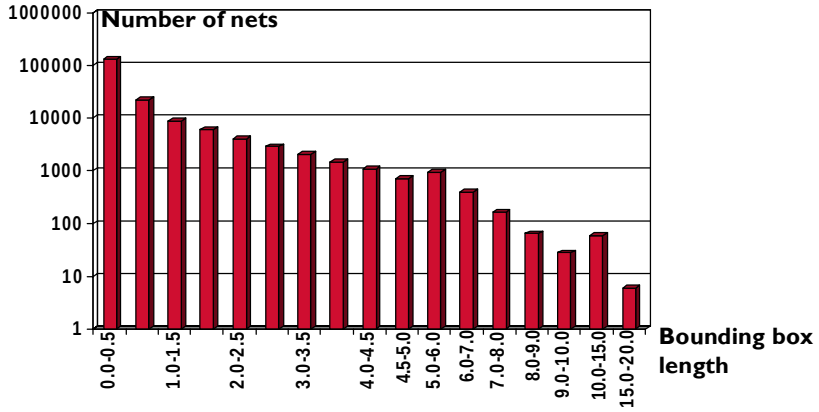

## Wirelength Estimation

- OK, what do we really use?
- Half-perimeter metric
  - Put a box around all the pins
  - Take 1/2 of perimeter, which is just length + width of box
  - This is a guaranteed lower bound on the amount of wire you need
  - (Why?)
  - This is really easy to compute, widely used.
  - Note, for 2-point nets this IS the Manhattan estimate!



## Reality Check: Wirelength Estimation

- Half-perimeter metric
  - Real distribution of bounding-box sizes for big IBM ASIC [Vygen DATE98]
  - 14.6mm$^2$, 181K nets, total wirelength: 106.34 meters



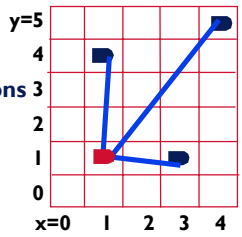



## Back to ASIC Placement

- Where are we?
  - If you have a placement (each gate located in a cell on grid), we can use this metric to compute $\Sigma_{nets}$ (estimated wirelength)
  - Can now tell if this placement is good ($\Sigma_{nets}$ = small) or bad ($\Sigma_{nets}$ = big)
- Now what?
  - How do we actually generate good placements?
  - Basic idea: *iterative improvement*
  - Start with a random placement
  - Perturb it (example: swap 2 gate's cell locations in grid)
  - Evaluate improvement = Δwirelength
- Questions
  - How do we know what to perturb, how much, when to quit, etc?

## Placement Strategies

- 1970s
  - "Optimal" perturbation schemes try to relocate gates their "best" new locations
  - Lots of variants



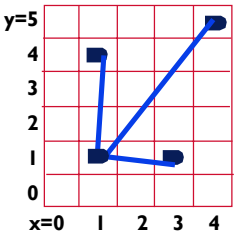

Example: treat other wires as springs, decide where they "pull" center gate to "want" to settle

- 1980s
  - Controlled random search techniques give us hill-climbing, more robust search
  - Most famous: *simulated annealing*
  - Vastly better than prior methods, but more CPU time



## Placement Strategies

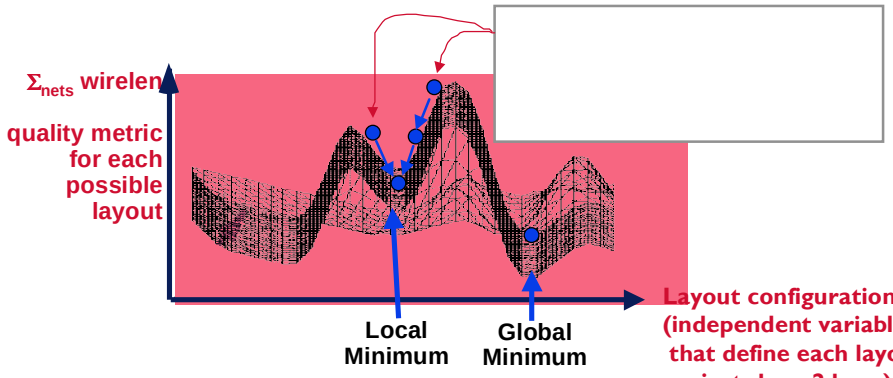
- 1990s
  - Mathematical programming
  - Pretend each wire is 2 points again, but that it's like a *spring*
  - Place to minimize the length of each spring, which is ~ Length$^2$
  - Turns out can do this with linear algebra, get a big sparse matrix Ax =b to solve
  - Several problems downstream now--technique does not respect grid locations or cell sizes (they all get treated like dimensionless pts)
  - But: fast, very good wirelength, works on enormous stuff.




Now, instead of perturbing 1 gate at a time, we solve for all gate locations in 1 big solve

## Placement Strategies

- What's wrong with the 70s "optimal" strategies?
  - They pick "good" perturbations that most improve wirelength...
  - ...and continue until they can't make any more progress
  - Problem: local minima in the cost surface for the placement task

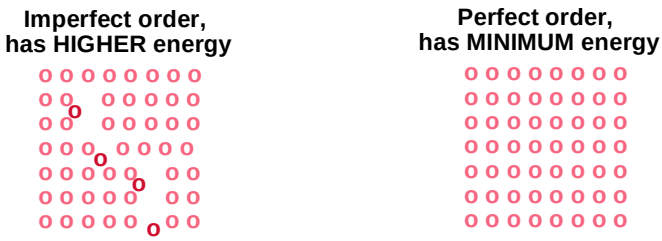





## Solution Technique: Simulated Annealing

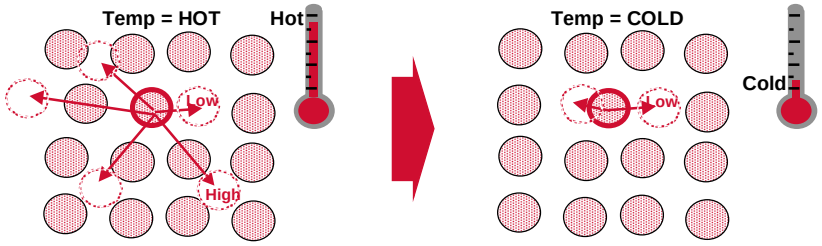
- Let's go waaay off to the side here and develop an idea
  - How far off to the side? Let's go look at some *statistical mechanics* from our friends in computational physics
  - Idea originally developed by Scott Kirkpatrick et al, physicist from IBM

- Suppose you want to make a *perfect crystal*
  - To do it: coerce the atoms to occupy their lowest energy configuration



## Simulated Annealing

- How do you do this physically?
  - You "anneal" the material
  - Get it very hot: gives atoms energy to move around
  - Cool it very slowly: gently restrict range of motion till everything freezes into (you hope) a low energy configuration




When it's hot...

When it's cold...



## Simulated Annealing

- Now what?
  - That was a real physical system: real atoms, energy, heat, etc.
  - Think about attacking this problem *computationally*
  - How do you *compute* this low energy state, from first principles.
- Back up a bit...
  - Suppose the temperature is constant
  - How do you *simulate* what these atoms are doing as they hop around?

## Annealing: Basics

- Phrase this question more exactly
  - How do you compute the low-energy configurations of a physical system in thermal equilibrium (ie, at a constant temperature)?
- Answer
  - *Metropolis* algorithm

Start with the system in a known configuration, at known energy E

Perturb system slightly (eg, move an atom to new location)

Compute $\Delta E$ , change in energy due to this perturbation

if ($\Delta E < 0$ )

then

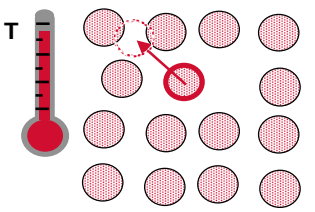
else

go back to start



## Aside: Metropolis Criterion

- That *if-then* in algorithm is "the Metropolis criterion"
  - After you perturb an atom and compute $\Delta$energy, it tells you if you keep this new perturbation as new configuration or throw it away
  - If the energy goes down, $\Delta E<0$, this is a "better" state: keep it
  - If energy goes up, $\Delta E >0$, this is a "worse state": *maybe keep it, depends on temperature*




Compute $e^{-\Delta E/KT}$

= a number in [0,1]

= Probability that you will accept this perturbation

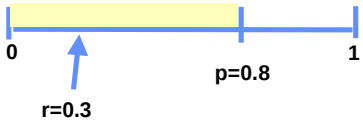
Generate r
a random number in [0, 1]

Compare r and $e^{-\Delta E/KT}$

if (r is smaller)
keep this perturb
else reject it

## Aside: Metropolis Criterion

- Example
  - Suppose $\Delta E > 0$
  - Suppose $p = e^{-\Delta E / KT} = 0.8$
  - Suppose you generated $r$ = uniform random number in $[0,1] = 0.3$
- What is really going on?




- What is the probability that $0 < r < 0.8$?



## Simulated Annealing

- Question
  - Metropolis algorithm iteratively visits configurations with "reasonably probable" energies at the given fixed temperature
  - What if I want to *find* a minimum energy state, now what do I do?
- Answer
  - *Simulated* annealing
  - Add outer loop that starts with a high temperature, and slowly cools it
  - Do enough perturbations at each temperature in the sequence of cooling steps to get to thermal equilibrium (ie, do the Metropolis procedure)
  - Do enough temperatures so that the problem actually freezes into a low energy state, and further cooling does not further lower energy

## Simulated Annealing

Start with the system in a known configuration, at known energy E

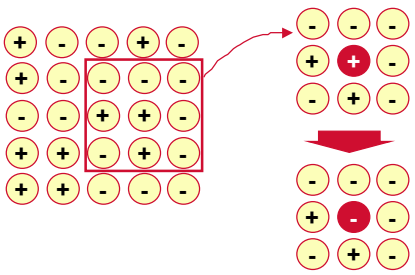
```
T = temperature = hot;    frozen = false;
while ( ! frozen ) {
    repeat {
        Perturb system slightly (eg, move a particle)
        Compute ΔE , change in energy due to perturbation
        if (ΔE < 0 )
        then   accept this perturbation,  this is the new system config
        else   accept maybe, with probability = e -ΔE/T
    } until (the system is in thermal equilibrium at this T)

    If (E still decreasing over the last few temperatures)
    then T = 0.9 T   /* cool the temperature; do more perturbations*/
    else  frozen = true
}
```

return (final configuration as low-energy solution)



## Toy Example

- Pretty easy to code a little example
  - Problem: 2D lattice of atoms, each in one of 2 states: + / -
  - Energy of the system:
    - Only in the bonds between neighbor atoms
    - Contribution is +1 if atom states different, else 0
  - To anneal:
    - Moves are just: pick an atom, flip the state, compute ΔE



Suppose we flip center atom

Old contribution to energy:

New contribution to energy:

ΔE is:

## Annealing Pseudo-Code

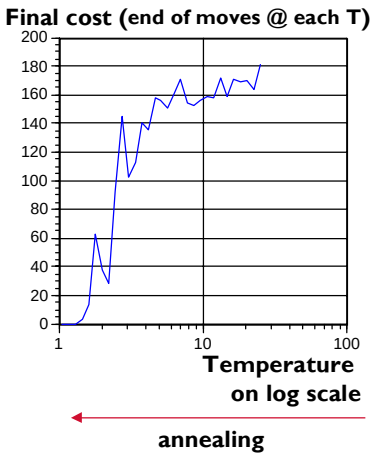
Pseudo-code

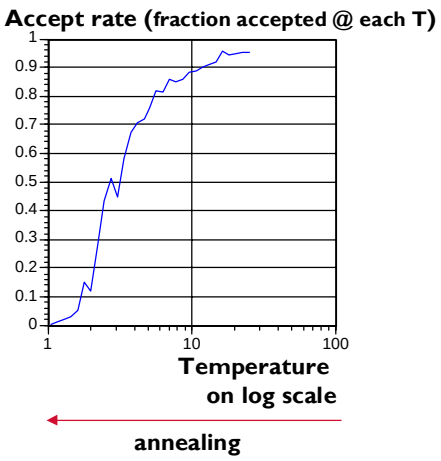
```
T = 100
Loop: for ( i = 1 to 10 * number of atoms ) {
      pick a random atom, flip it, compute ΔE
      accept = metropolis(ΔE, T)
}
if  (total cost is still improving, ie, changed > 1% over last 3 temps)
      T = 0.9 * T
      goto Loop;
else quit
```



## Toy Annealer: Results

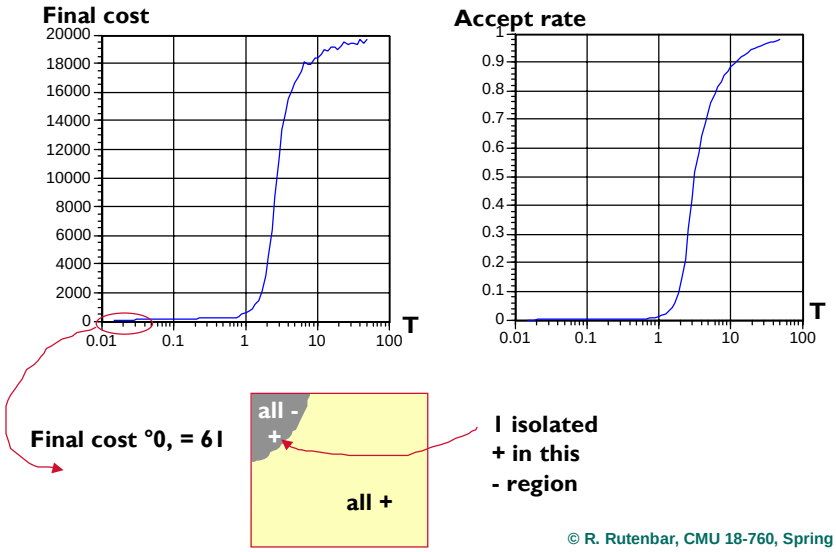
- 10x10 lattice, 1000 moves per temperature



## Toy Annealer: Results

- 100x100 lattice, 250,000 moves per temperature






## What Has This To Do With VLSI CAD?

- Combinatorial optimization problems are like these physical systems being coerced into low-E states

| Physical System | Mathematical System |
|---|---|
| System with atoms in various states | |
| Energy | |
| ΔE perturbation | |
| Lowest energy "groundstate" | |
| Temperature | |
| Quenching | |
| Annealing | |

## Annealing Algorithm: Essential Pieces

- What are the components of any annealing solutions to a combinatorial problem?

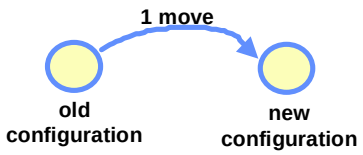
- State representation
  - Exactly what are the configurations of solutions to your problem that you will visit as you iteratively perturb things?

- Cost function
  - How will you measure how good each visited configuration is during iterative perturbations?
  - This plays the role of "energy" in simulated annealing



## Annealing Algorithm: Essential Pieces

- Move set
  - In annealing-speak, perturbations are always called "moves"
  - The *move set* is the set of "types" of perturbations that you with do to evolve from one solution configuration to the next




  - Examples:
    - Move that atom from (x,y,z) to (x',y',z')
    - Rotate that block in the floorplan for the chip
    - *Swap the position of those 2 gates in the placement*

## Annealing: Essential Pieces

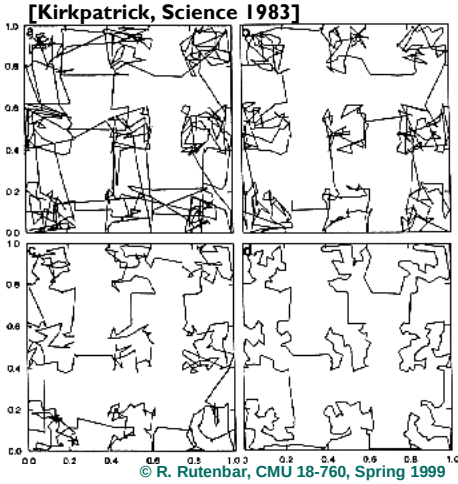
- Cooling Schedule
  - Starting temperature
    - How hot is not enough at the start of annealing?
    - Usually want it hot enough that any move you try is accepted
    - When it's hot, you basically randomize the solution
  - Equilibrium criterion
    - How do you know you have done enough moves at the current temperature to stop, and exit to see if you should cool T?
    - Usually just do a *lot* of moves at each temperature (~100*objects)
  - Cooling rate
    - How fast to cool? Tnew = 0.9•Told ? Tnew = 0.8 • Told ?
    - Slower cooling (0.9) gives better answers, but takes longer
  - Frozen criterion
    - When is overall solution as good as it will get, so it's time to quit?
    - Usually wait a few temps and see if cost stops changing much



## Simple Combinatorial Optimization Example

- Travelling Salesman Problem
  - Visit a set of cities in order, one visit per city, first city = last city
  - Minimize total length of travel
- To anneal
  - *State* = list of cities in order, called a *tour*
  - Ex: (Detroit, Paris, Lisbon, London, Detroit)
  - *Move* = swap 2 cities in tour
  - Ex: (Detroit, London, Lisbon, Paris, Detroit)
  - Cost = sum of lengths of travel, city to city, on tour
  - Cooling stuff -- you know

[Kirkpatrick, Science 1983]



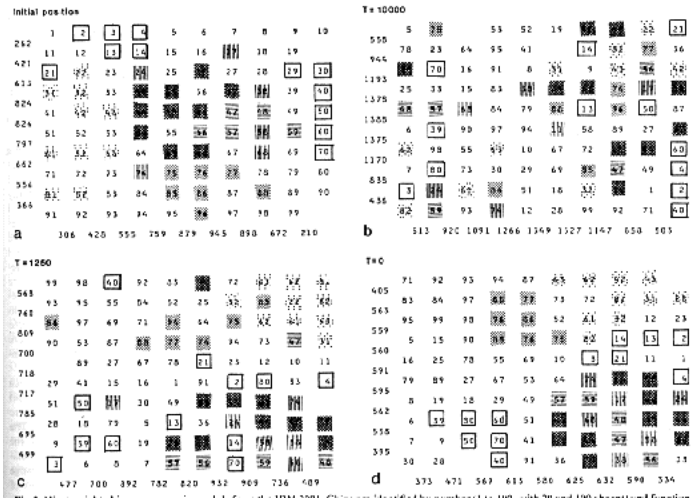
## OK, How to Do ASIC Placement?

- Surprisingly easy to do a "simple" placer
- State
  - Just the (x,y) location of each placeable object in our grid
- Cost
  - Just total estimated half-perimeter wirelength
- Moves
  - Easiest is pick to 2 random gates and swap their locations on the grid
- Cooling
  - T init = hot; Tnew = 0.9*Told; do a lot of moves at each temperature to ensure you are in equilibrium (eg, 100*number of gates moves/temp)
  - Quit when the cost curve versus temperature is flat enough



## Example: [Kirkpatrick, Science 1983]

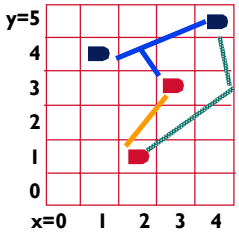
- Actually placing chips on a package, but same idea

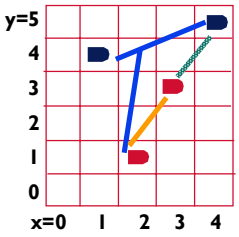


## Optimizations

- Incremental cost calculation
  - You cannot afford to go recompute the cost of each net in the entire placement after you do one measly little swap
  - For one thing, it's *stupid*: most lengths didn't change!
  - You have do this incrementally--just look at the wires that *could* change

Δwirelen =



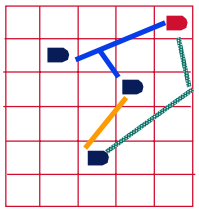





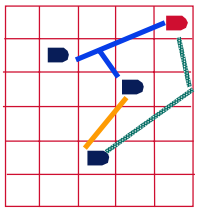

## Optimizations

- Range limiting
  - You don't get any rewards for proposing moves that have a very high probability of being rejected -- rejected moves don't advance solution
  - Sometimes you can tell in advance which are more likely to succeed
  - Range = amount by which the cost is likely to change if you do this move
  - T = HOT, moves with large range are OK; T=COLD, not

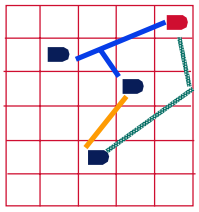
HOT



WARM



COLD



## Why Does Annealing Work?

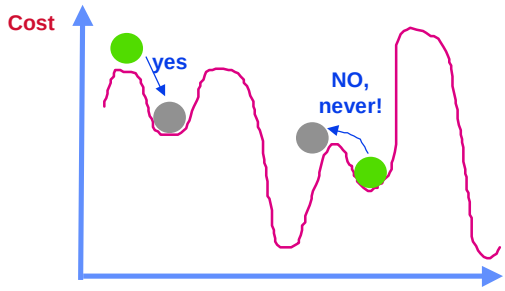
- Helpful mental model #1: *Balls & Hills*
  - Look at a simple representation of a combinatorial optimization task
  - Can model as a cost surface (also called a "landscape" or "space")
  - The configuration we are visiting now is the "ball" on the "hill"


All possible configurations of system
that is being optimized.
Note we only draw for 1 variable

## Balls & Hills

- Consider classical "greedy" iterative improvement
  - Only take moves that improve the cost
  - Physical analogy: like a quench, cooling too fast, you get lousy crystal
  - Can get easily trapped in local minima

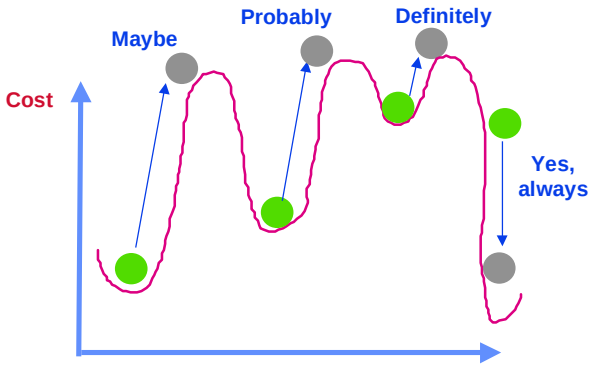



All possible configurations of system
that is being optimized.
Note we only draw for 1 variable

## Balls & Hills

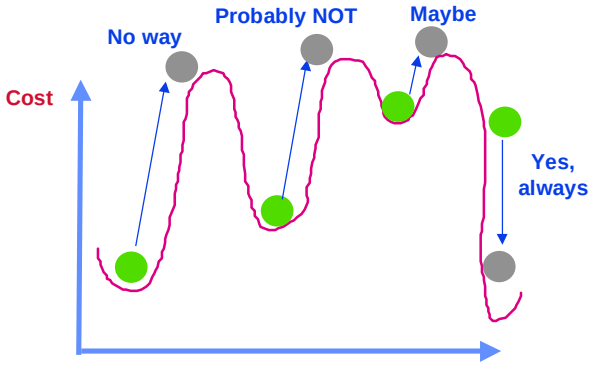
- Simulated annealing allows probabilistic hill climbing
  - Suppose temperature T = HOT, remember Pr[accept] = $e^{-\Delta C/T}$



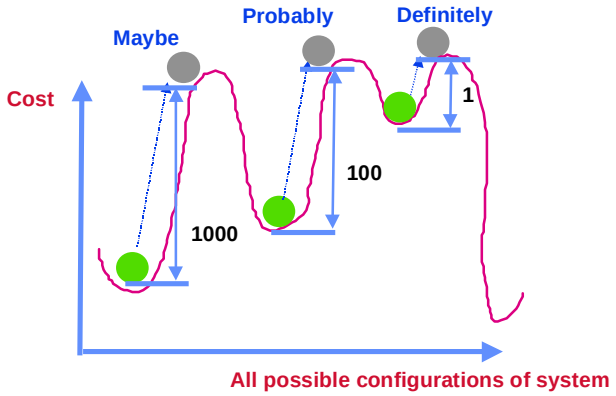



## Balls & Hills

- Simulated annealing allows probabilistic hill climbing
  - Suppose temperature T = COLD, remember Pr[accept] = $e^{-\Delta C/T}$
  - As temperature cools, fewer uphill moves acceptable



## Balls & Hills: Some Numbers




| Uphill ΔC | *Probability we will accept this move* Hot T=1000 | Warm T=100 | Cold T=1 |
|---|---|---|---|
| 1 | 0.999 | 0.99 | 0.37 |
| 100 | 0.900 | 0.37 | ~0 |
| 1000 | 0.37 | 0.00004 | ~0 |

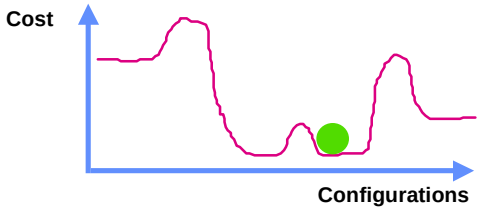


## Why Does Annealing Work?

- Helpful mental model #2: *Landscape flattening*
  - The annealing temperature idea smooths out the bumps on the cost surface, making more of the surface reachable
  - As the temperature cools, the bumps "reappear" and force search process to a good local minimum
  - In contrast, downhill-only style algorithms get stuck at the very first local minimum, ie, they cannot get over even the first uphill bump they encounter
  - Idea is sometimes referred to as "adaptive smoothing" of the cost surface

## Landscape Flattening

- Consider this bumpy cost surface (ball & hills)

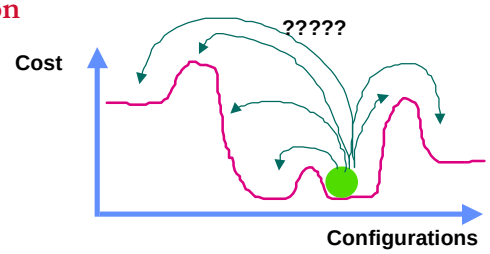



- Question
  - As a function of temperature, how much of this cost surface is reachable if we start from where the ball is in this figure?
  - Remember: T= hot => big uphill jumps
    T = cold => small or no uphill moves

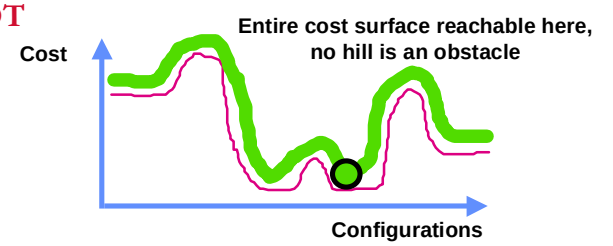


## Landscape Flattening

- Question

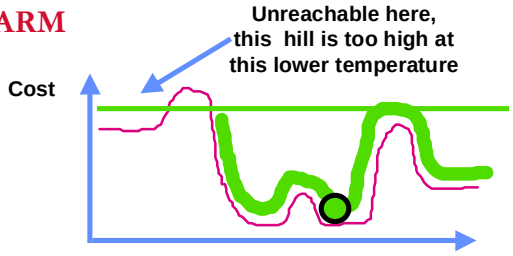



- T = HOT

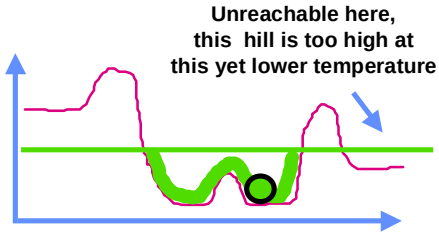


## Landscape Flattening

- T = WARM




- T = COLD






## Landscape Flattening

- T = FROZEN


- Idea
  - Temperature T "hides" the obstacles when hot
  - Cooling restricts us to smaller and smaller "reasonable" areas

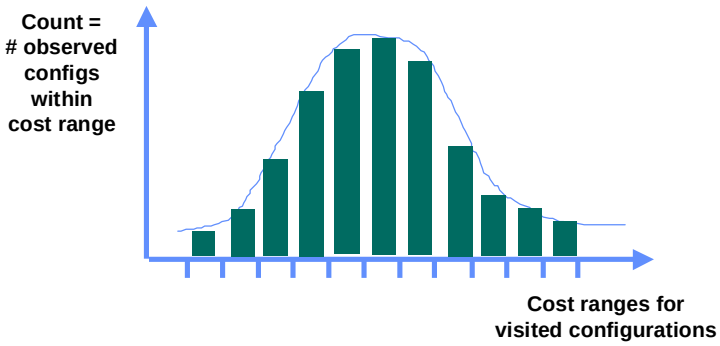
## Annealing Dynamics

- Question
  - When my annealer is running, what do I actually see happening at each temperature, and across sequences of decreasing temperatures?
- Answer
  - At each temperature, you visit solution configurations in your "neighborhood" of the cost surface
  - Those solution configurations will all have different costs
  - You will see a "distribution" of costs at any fixed T
  - What does that distribution look like?



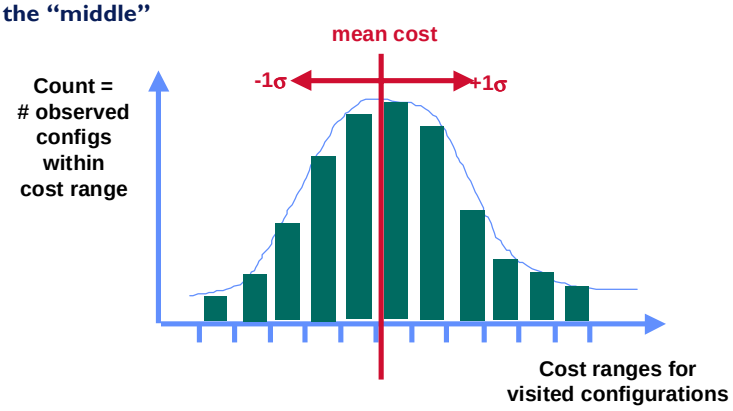
## Annealing Dynamics

- Distribution of configurations at temperature
  - Can make a histogram, with ranges for cost of solutions seen
  - Vertical axis counts how many configurations visited that fall into each cost "bucket"
  - Get a bell-shaped distribution



## Annealing Dynamics

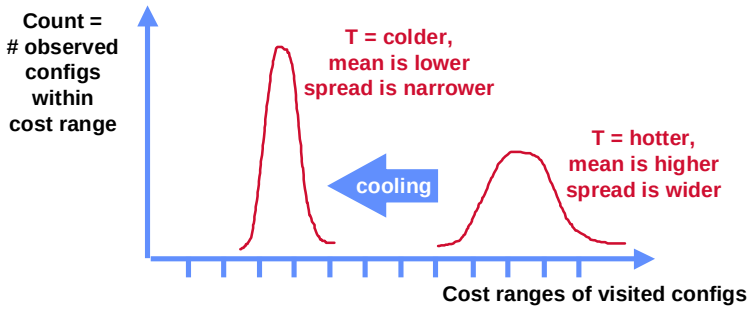
- Typically...
  - You visit some really good (low cost solutions), but temperature is high enough you keep jumping out
  - You visit some really lousy configurations (uphill) but keep falling back to the "middle"





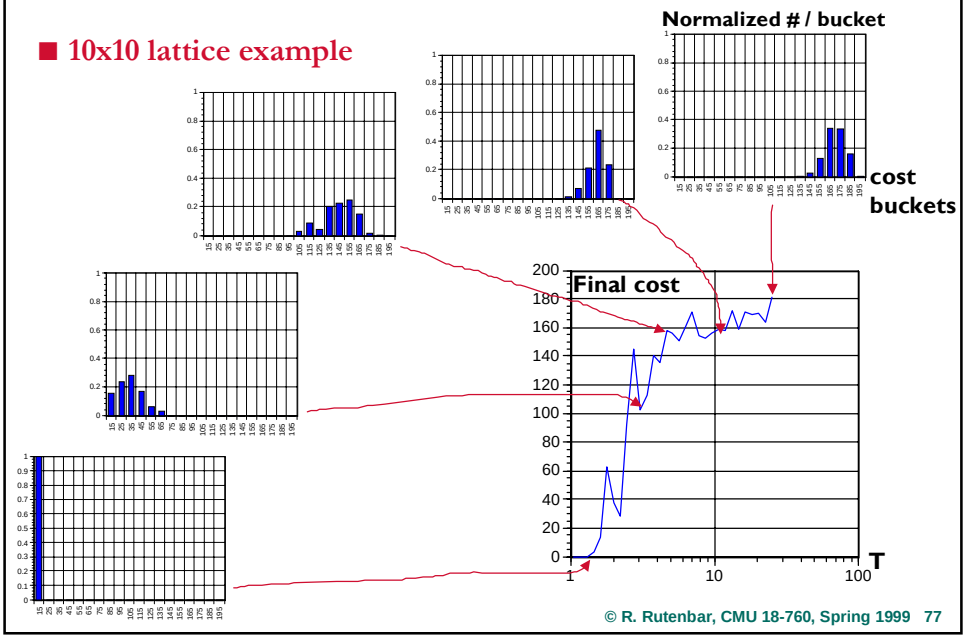

## Annealing Dynamics

- What happens to distribution as cooling proceeds?
  - Histograms get narrower: unwilling to visit so many bad configs in the neighborhood, and there are fewer "better" configs around
  - Histograms get taller: more of the solutions you find are near the mean, temp is too low to jump uphill to really worse ones, and again there are fewer better ones around to fall down into



## Toy Example: How Cost Distribution Evolves

- 10x10 lattice example






## Some Annealing Facts

- Does annealing always get global minimum?
  - No. It's just good at avoiding *many* local minima
- Is annealing always slow?
  - Well, you do have to visit a lot of candidate solutions...
  - ...but it's competitive in a lot of situations
  - Good for ASIC cell placement (upto about 100k gates or so). Great answers, OK times.
- Does it work for any combinatorial problem?
  - No, or at least, not ideally
  - Good for things with messy, mutually interacting constraints, nasty cost functions, etc. In other words, *engineering* problems

## Some More Annealing Facts

- Repeatability of results
  - If I run an annealer on the same problem 10 times, using 10 different random number sequences for the moves, what will I get?
  - 10 different solutions. (Hey, it's random search, after all!)
  - Cost for each will be close, but even then these will probably be distributed just like the standard distributions at any T during annealing.
- What if I store the very best config I see over all T's?
  - Still probably won't matter, still probably get 10 solutions, at least for a big interesting problem.
- How do people actually deal with this?
  - Run it a couple of times, take the answer you like best.
  - Don't use it on things where you have to have the optim*um* answer, instead of just a very good answer



## Yet More Annealing Facts

- Controllability
  - Do people really play with all the magic numbers for each new problem?
  - T-hot, cooling rate, equilib. criterion, weights in cost function, etc
- No.
  - There are *automatic* techniques that set these things adaptively, based on how the problem "wants" to cool; ie, algorithm designer plays with these, NOT the tool user
  - Automatic techniques for:
    - Initial temperature
    - Cooling schedule
    - Equilibrium
    - Weights in cost function
    - Which moves to pick to minimize the probability they will be rejected (ie, don't try moves so stupid that they have high likelihood of getting bounced, these just waste CPU time)

## Summary

- Annealing is
  - A way of constructing algorithms for combinatorial optimiz. problems
  - Iterative improvement with hill climbing
  - Composed of a few essential pieces
    - State representation, cost function, move set, cooling schedule
  - Good at not getting stuck in some local minima
- ASIC placement
  - Again, an iterative improvement process
  - Quality metric is total estimated wirelength; half-perimeter is common
  - Annealing is extremely successful for placement problems
  - Avoids local minima in cost surface, does a great job minimizing wirelen