Guest Lecture: Placement and Routing for FPGAs

Larry McMurchie
Synopsys, Inc.
Formerly with Depts. of EE and CS, UW

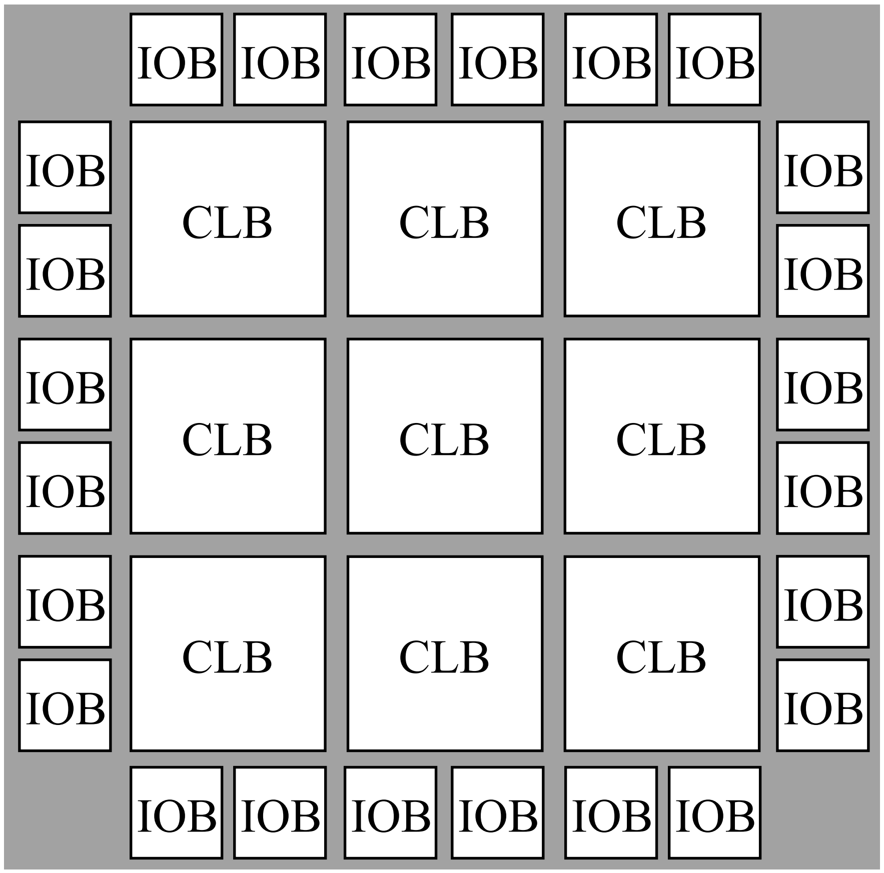
20 min. Placement
20 Min. Routing

Placement

Assign logic blocks to specific chip locations

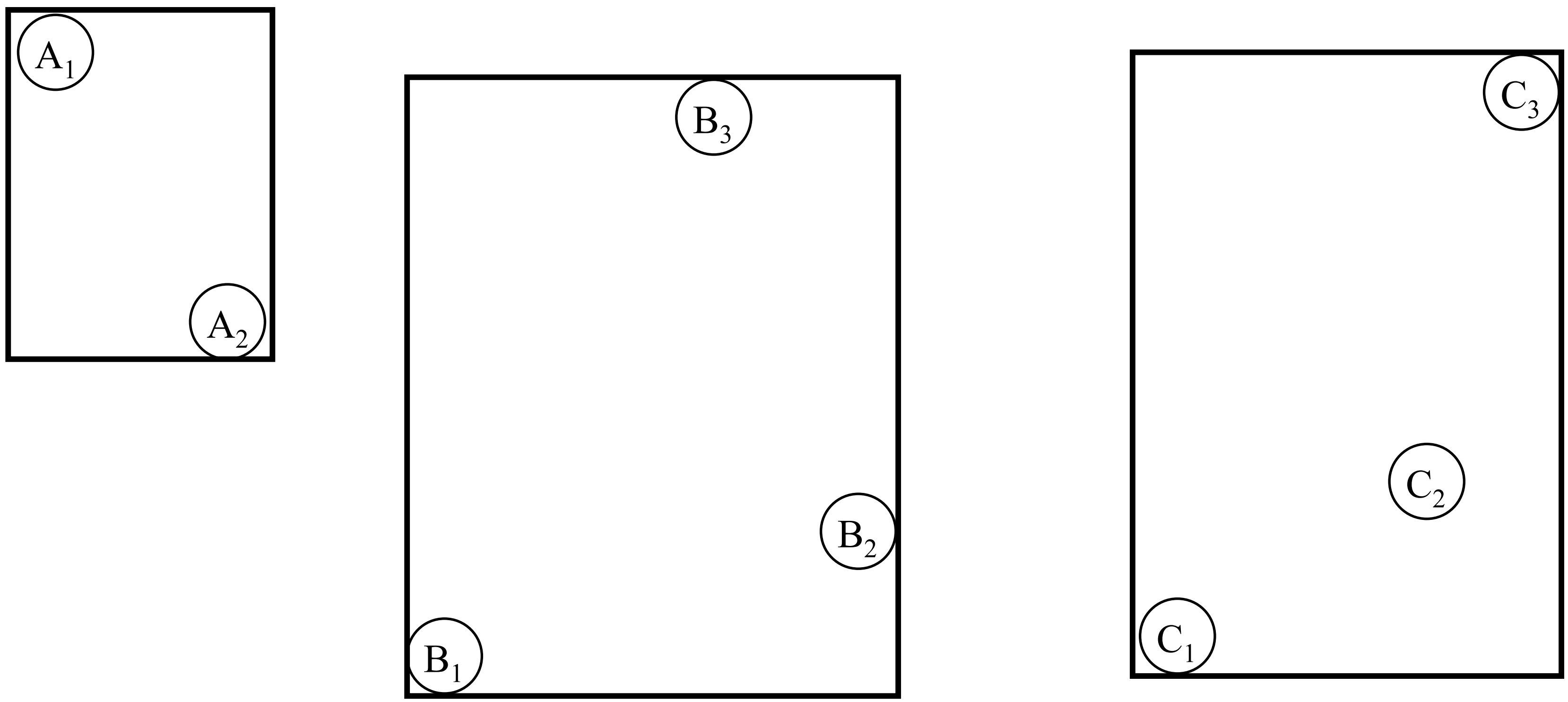
Goal: minimize routing distance and therefore allow successful routing

Additional goal: Meet timing constraints on critical signals



Placement Cost Function - Wirelength

Most systems use “Manhattan” routing (North, South, East, West, no diagonals)

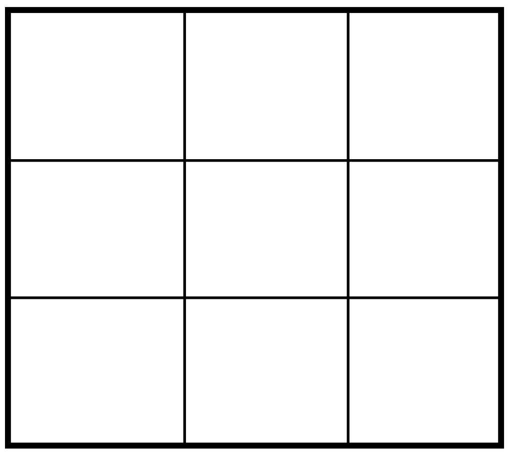


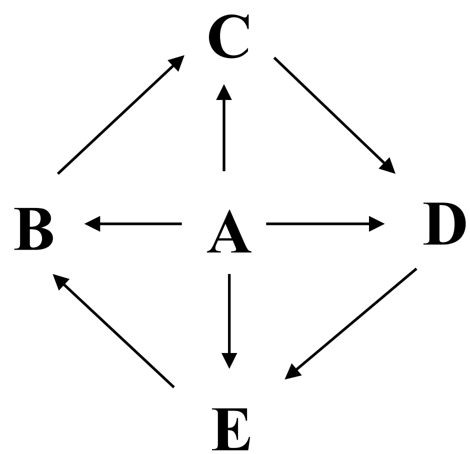
Wirelength estimate = 1/2*(perimeter of bounding box) = “Semi-perimeter”

Greedy Placement

```
Create initial placement randomly
old_cost = cost(placement);
for (iteration = 0; iteration < max_iteration; iteration++) {
   swap random pair of logic blocks;
   new_cost = cost(placement);
   if (old_cost < new_cost)
      undo_move();
   else old_cost = new_cost  /* Keep new placement */
}
```

Example of greedy placement problem





Example of greedy placement problem

A “clairvoyant” placement:

B	**C**	
A	**D**	
E		

Total cost = A->B + A->C + A->D + A->E + B->C + C->D + D->E + E->B
= 1 + 2 + 1 + 1 + 1 + 1 + 2 + 1.5
= 10.5

Example of greedy placement problem

An unfortunate starting point:

A	**B**	
D	**C**	
E		

Total cost = 11.5

```
A<->B gives 12.5
A<->C gives 12
A<->D gives 12.5
A<->E gives 11.5
```

All possible pairwise interchanges give higher cost
We are stuck with the above placement!

Greedy Placement Summary

Greedy placement algorithms (e.g. force-directed, recursive bipartitioning) can easily get stuck in local minima

Need a method that is less susceptible to local minima and can perform “hill climbing”

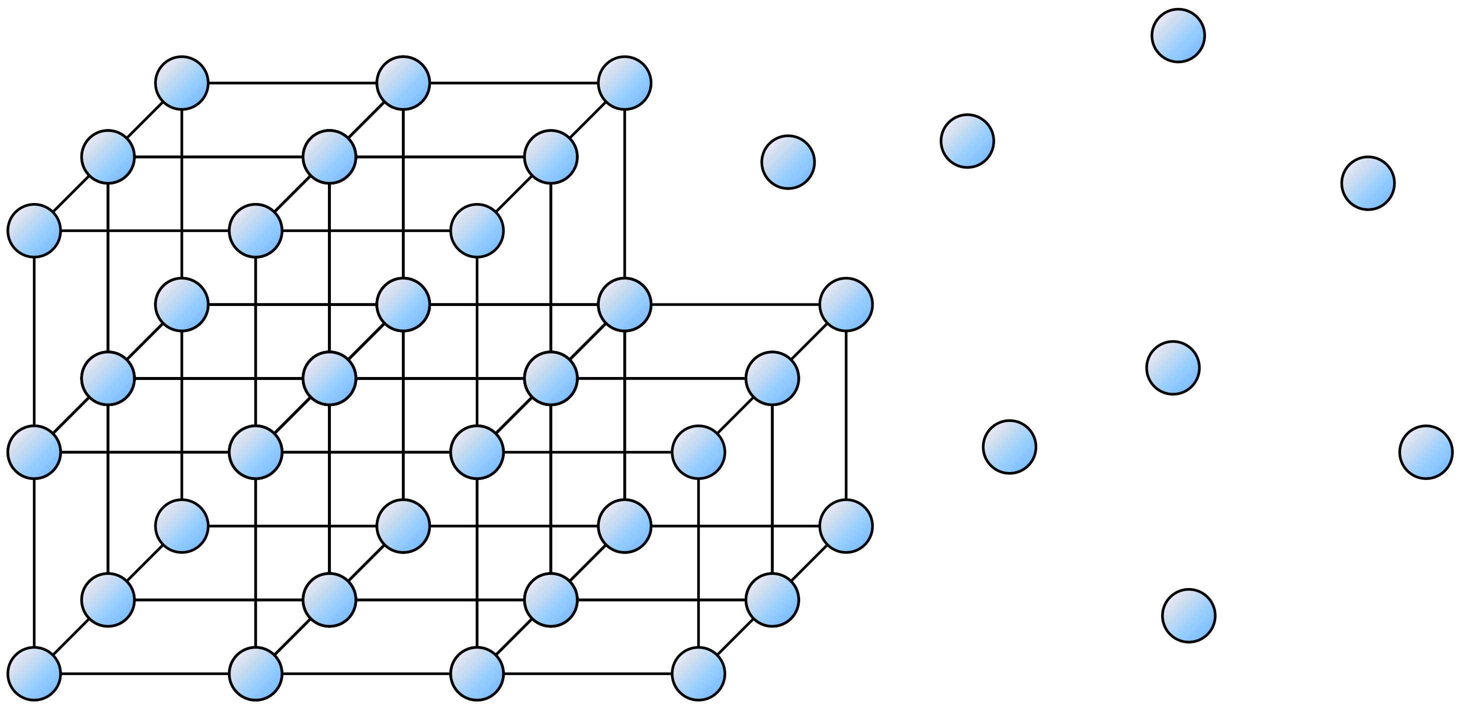
Lots of methods have been tried!

“Simulated annealing” is perhaps the most widely used.

Annealing

Annealing: Cooling hot molecules to form good crystal structures

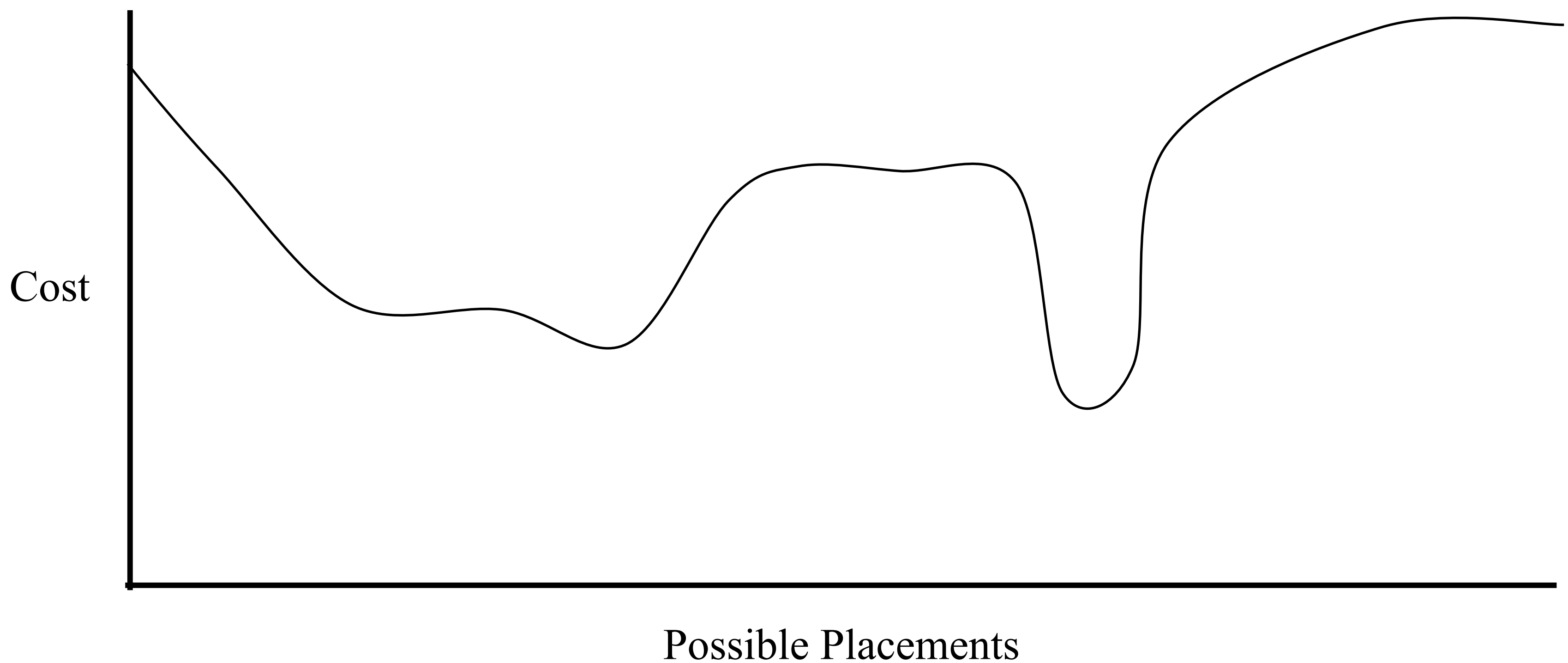
- Start at high temperatures - molecules move randomly about
- Cool at specific cooling schedule - leave enough time for molecules to form crystal lattice



Simulated Annealing

Move nodes randomly

- Initially “high temperature” - allow bad moves to happen
- Lower temperature, accepting less and less bad moves
- Slowly “cool” placement to allow good structure to form

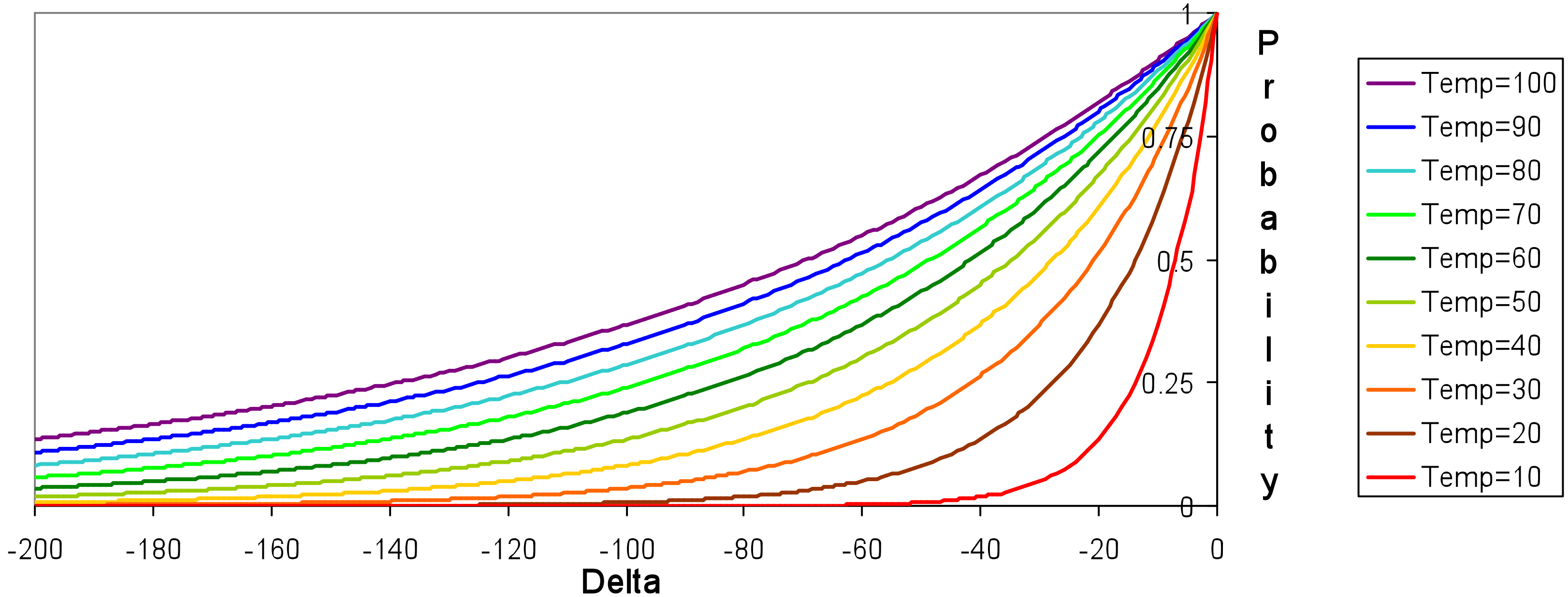


Simulated Annealing Acceptance Criteria & Cooling Schedule

Compute delta = cost(old_placement) - cost(new_placement)

if (delta>=0) accept

else if ($random < e^{delta/Temp}$) accept, else reject /* 0<=random<=1 */



Initially temperature is very high (most bad moves accepted)

Temp slowly goes to 0, with multiple moves attempted at each temperature

Final runs with temp=0 (always reject bad moves) greedily “quench” the system

Simulated Annealing Algorithm

```
Create initial placement randomly
old_cost = cost(placement);
for (temp = max_temp; temp >= min_temp; temp = next_temp) {
   for (iteration = 0; iteration < max_iteration; iteration++) {
      Swap random pair of logic blocks;
      new_cost = cost(placement);
      if (old_cost < new_cost)
         if (random >= Func((old_cost - new_cost)/temperature))
            undo_move();
      else old_cost = new_cost   /* Keep new placement */
   }
}
```

Simulated Annealing Details

Cooling schedule is important!

- Cooling too fast will force a greedy solution

Slow, but amenable to parallelism

- Nature does this very efficiently…

Simulated Annealing cost function is extensible!

- Multiple goals captured in one metric, for example:

$$cost(placement) = c_1\left(\sum_{i \in nets} semi - perimeter(i)\right) + c_2 * criticality$$

- Criticality is determined by length of path, clock cycle, placement, etc.

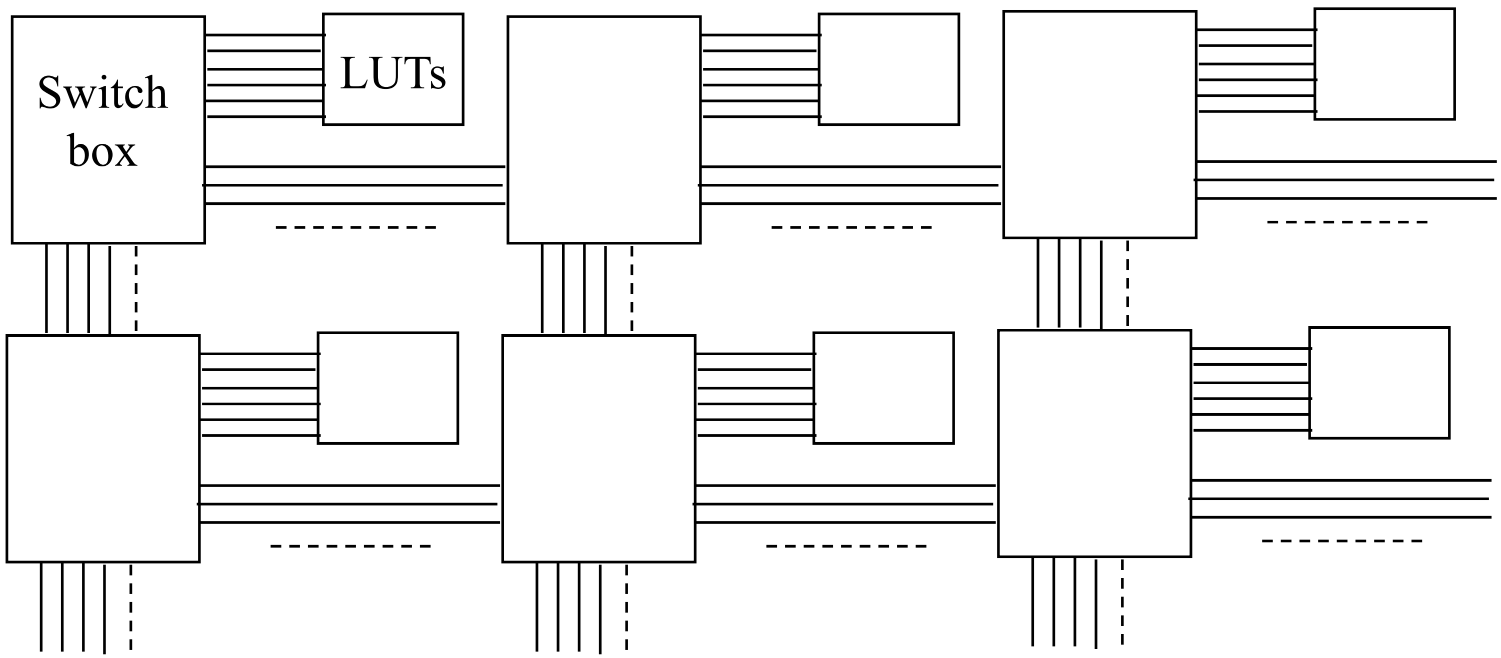
In practice simulated annealing is easy to program, very versatile (though slow), and widely used.

Results of annealing too quickly

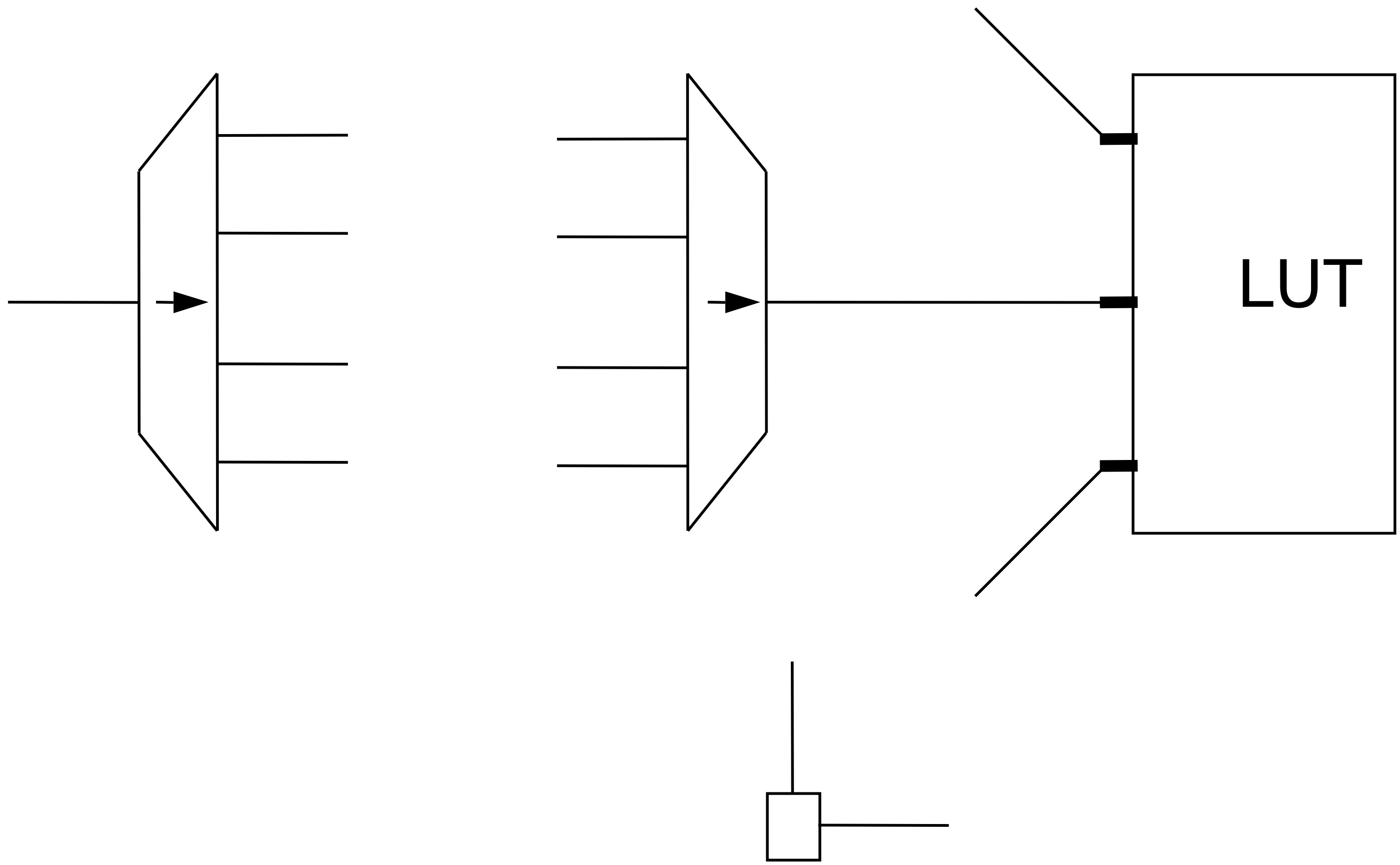
Results of annealing slowly

FPGA Routing Architecture

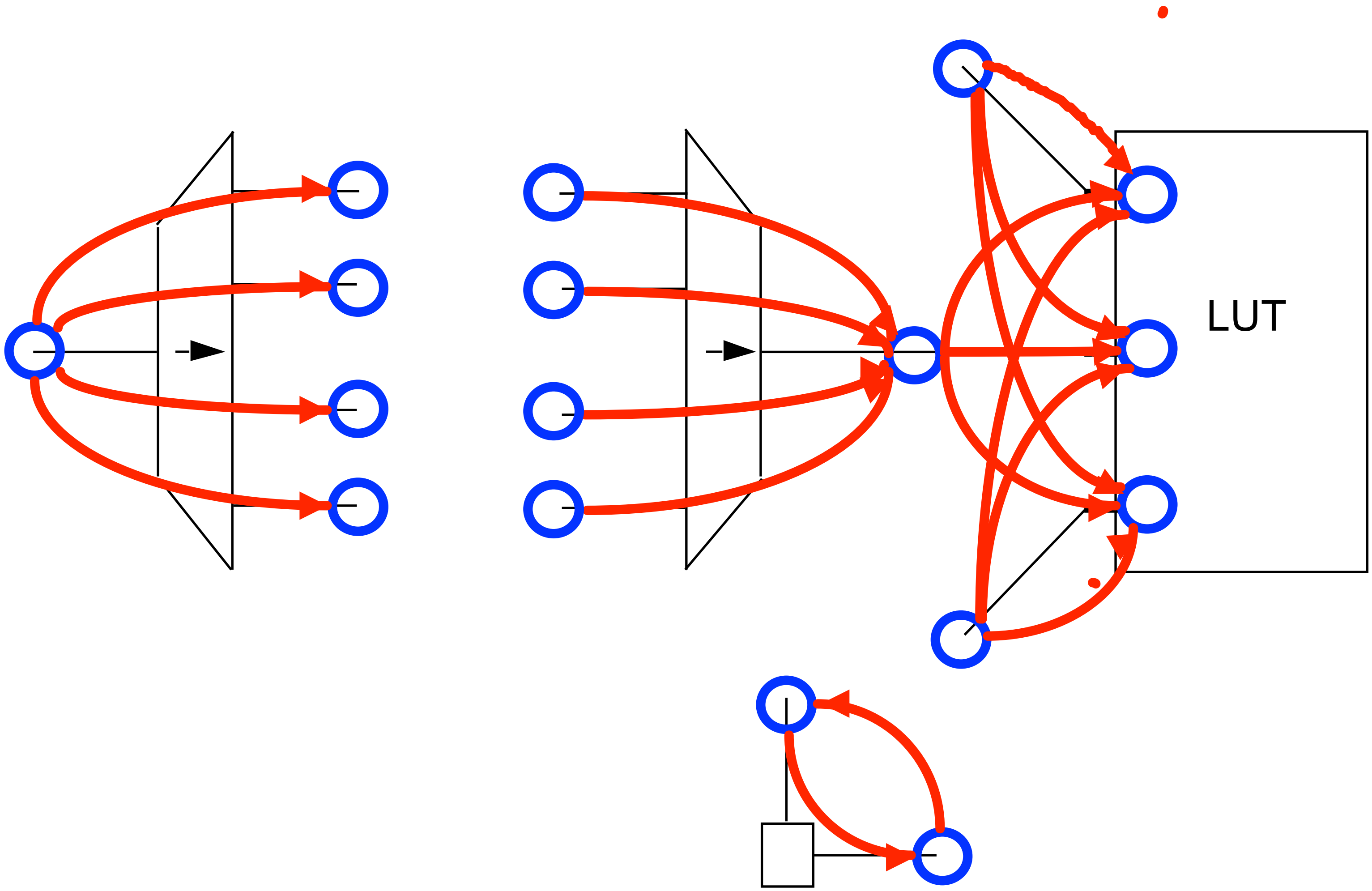
"Island" style architecture



Routing Elements



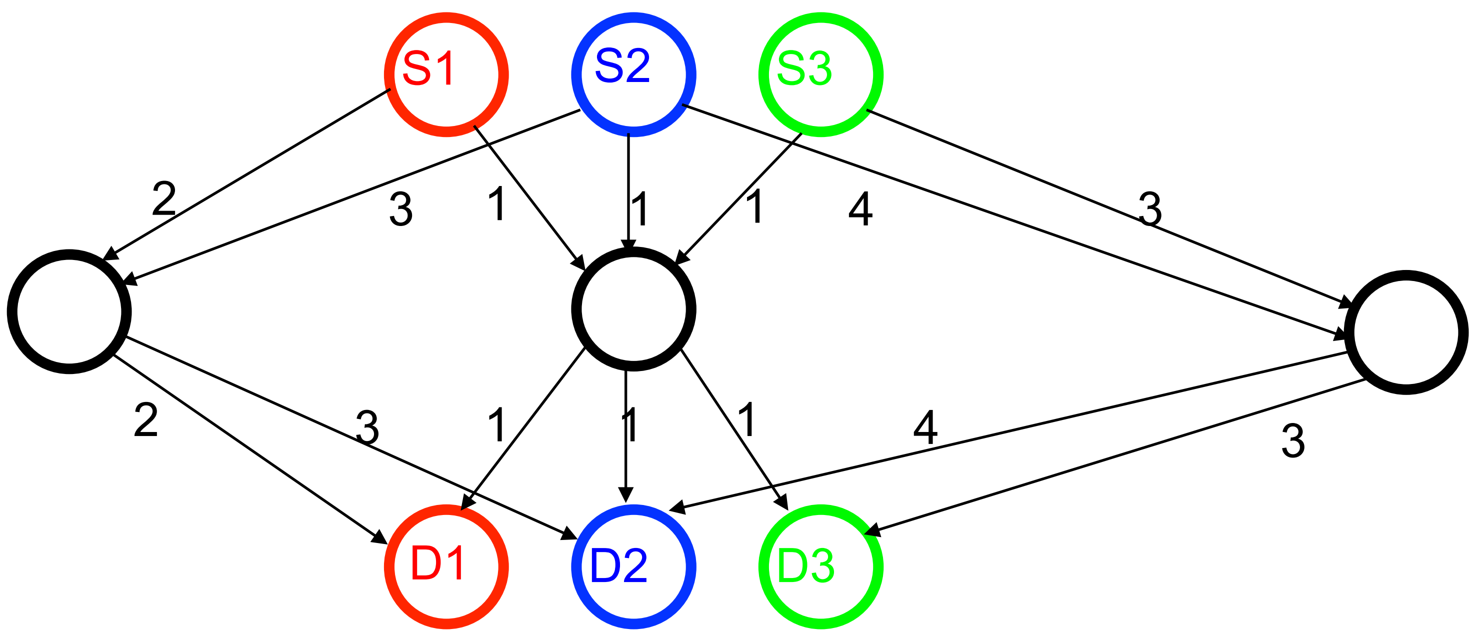
Abstraction to a Directed Graph



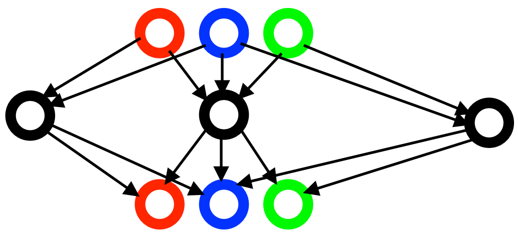
Routing goals

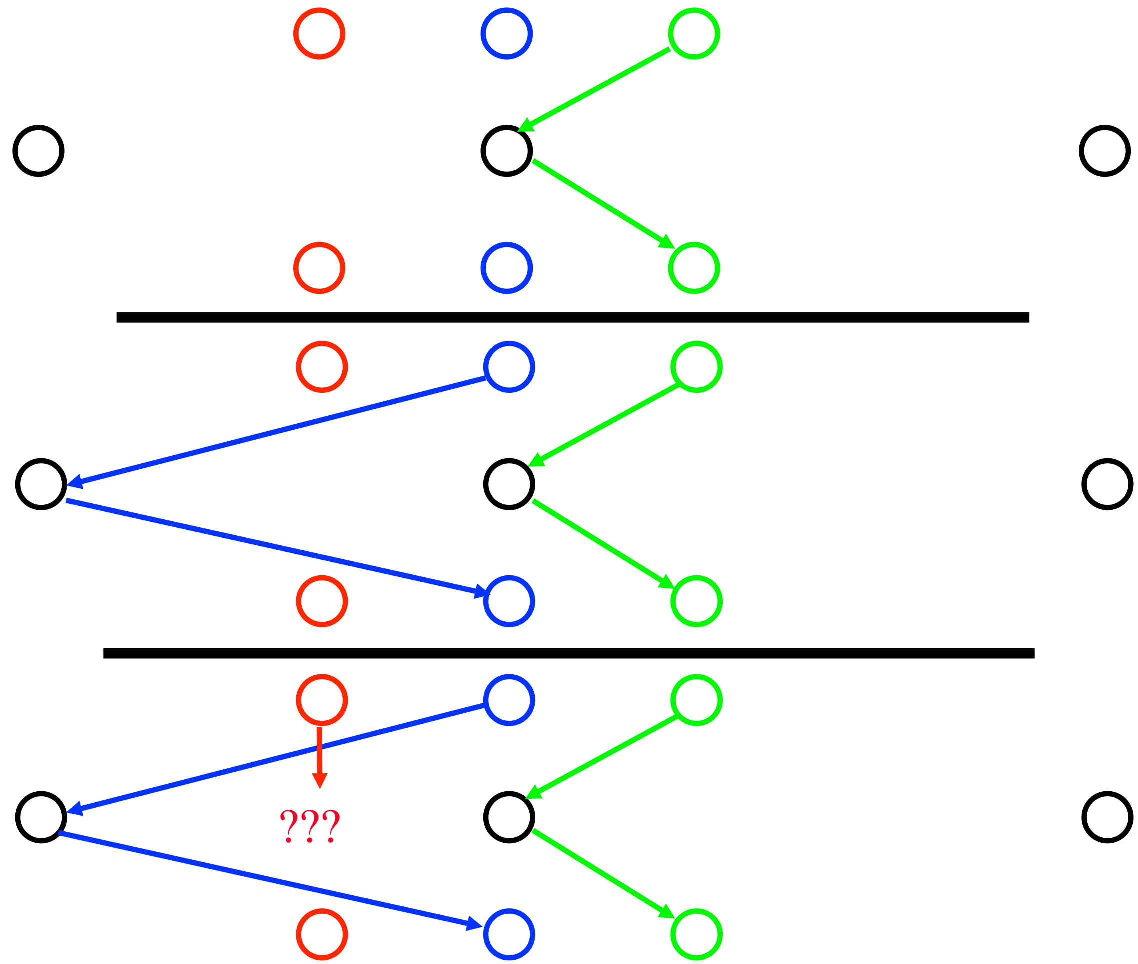
1. Find a feasible routing for all signals (nets) using routing network
   - Feasible means:
     - Different signals cannot share same nodes in routing network
     - “Sharing” == “Congestion”
3. Optimize delay of critical nets
   - Take more direct routes for critical nets
   - Non-critical nets can take longer routes
   - Tradeoff between optimizing delay for critical nets and finding feasible routing for all nets.

1st Order Congestion Example



Simple Dykstra Routing Algorithm with Obstacle Avoidance





How To Fix Simple Obstacle Avoidance Routing

1) Results depend upon the order in which signals are routed
2) Easy for signals to be blocked and prevented from being successfully routed.
3) Try different orderings

   What algorithm should be used to guide the ordering?
4) Use simulated annealing to guide routing in a manner similar to placement.
   Use a random choice of routes guided by a cost function and cooling schedule

   This has been tried and shown to work, but is computationally expensive.
5) Try the Pathfinder algorithm

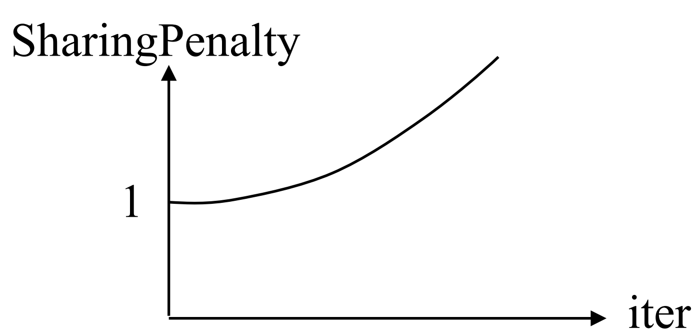
PathFinder Algorithm: Approach

1) Iterative approach – route every signal every iteration

2) Start with base costs for every arc

3) During subsequent iterations, GRADUALLY increase the cost for arcs that go to nodes that are already occupied with another net.

Cost = BaseCost * SharingPenalty(iter)



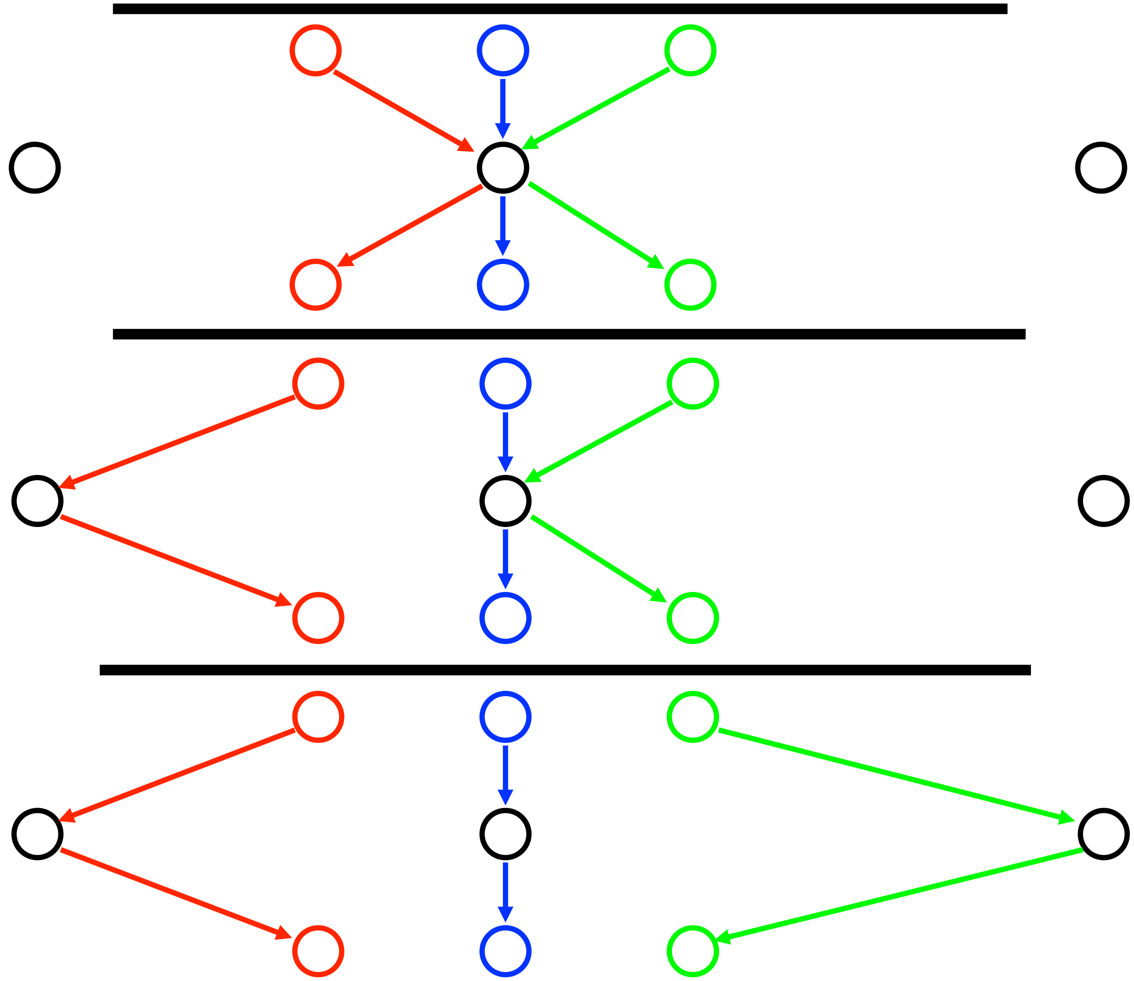
PathFinder Algorithm: Example

Cost = BaseCost * SharingPenalty(iter)

Iteration 0

Iteration i

Iteration i++



Pathfinder Algorithm Enhancements

1) Cost function can include criticality as well as sharing:

Cost = BaseCost *[A(j) + (SharingPenalty(iter)) *(1-A(j))]
where A(j) is the criticality of signal j and
0 <= A(j) < 1

Critical signals see only A(j) term
Noncritical signals see (1-A(j)) term

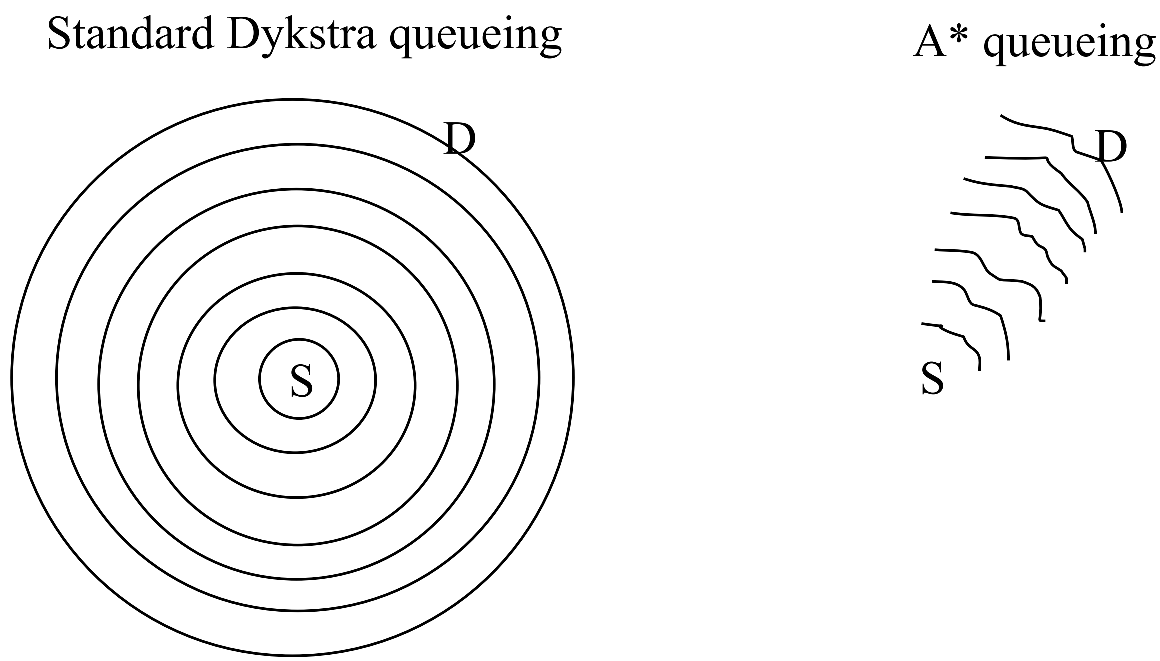
As a result, critical signals will take more direct routes and less critical signals will move out of their way

Pathfinder Algorithm Enhancements (cont.)

2) Runtime improvement is obtained by rerouting only signals which are involved in sharings

3) Additional runtime improvement is obtained by using A* queueing.

   A* queueing requires a knowledge of the minimum delay from each node visited in the routing graph to the destination



What conclusions can we make?

Both simulated annealing and the pathfinder algorithm are heuristics.

Neither are guaranteed to produce high quality results.

- Dependencies include cooling schedule and form of sharing penalty function.

Both algorithms have “good” intuition about finding high quality results.

Both are reasonably easy to implement.

Both work quite well in practice and have been used almost as long as FPGAs have been around.