CS137: Electronic Design Automation

Day 12: February 13, 2002

Scheduling

Heuristics and Approximation

Today

- Scheduling
  - Force-Directed
  - List-Scheduling
  - Approximation Algorithms

Last Time

- Resources aren't free
- Share to reduce costs
- Schedule operations on resources
- Greedy not optimal
- Optimal solution with Branch-and-Bound
- Lower Bound Pruning

This Time

- Heuristic/non-optimal approaches
- Use to solve quickly
- Use to generate upper bounds
  - help pruning in alpha-beta search
- Bounds on "badness" of heuristic

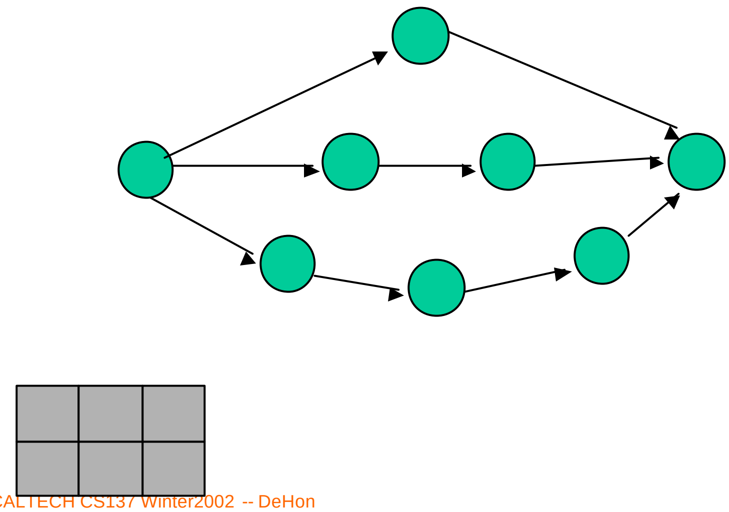
Force-Directed

- **Problem**: how exploit schedule freedom (slack) to minimize instantaneous resources
  - (time constrained)

Force-Directed

- Given a node, can schedule schedule anywhere between ASAP and ALAP schedule time
  - latest schedule predecessor and ALAP
  - ASAP and already scheduled successors
- Scheduling node will limit freedom of nodes in path

Example



CALTECH CS137 Winter2002 -- DeHon

Force-Directed

- If everything where scheduled, except for the target node:
  - examine resource usage in all timeslots allowed by precedence
  - place in timeslot which has least increase maximum resources

CALTECH CS137 Winter2002 -- DeHon

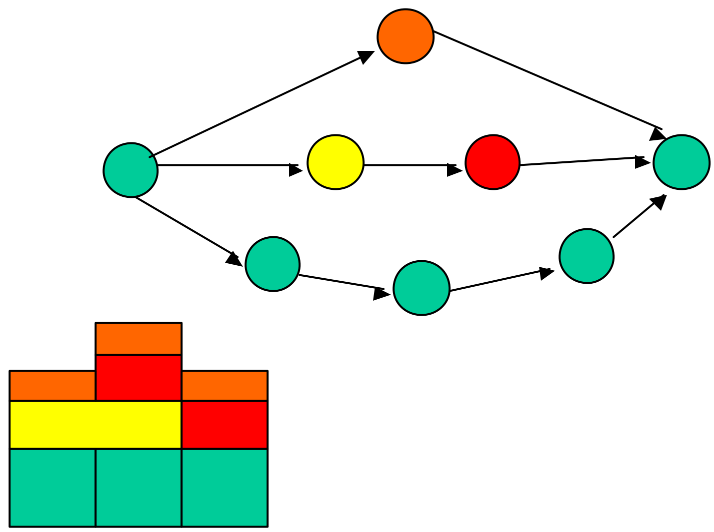
Force-Directed

- **Problem:** don't know resource utilization during scheduling
- Strategy: estimate resource utilization

Force-Directed Estimate

- Assume a node is uniformly distributed within slack region
  - between earliest and latest possible schedule time
- Use this estimate to identify most used timeslots

Example



Force-Directed

- Scheduling a node will shift distribution
  - all of scheduled node's cost goes into one timeslot
  - predecessor/successors may have freedom limited so shift their contributions
- Want to shift distribution to minimize maximum resource utilization

Force-Directed Algorithm

- ASAP/ALAP schedule to determine range of times for each node
- Compute estimated resource usage
- Pick most constrained node
  - Evaluate effects of placing in feasible time slots (compute forces)
  - Place in minimum cost slot and update estimates
  - Repeat until done

Time

- Evaluate force of putting in timeslot O(NT)
  - Potentially perturbing slack on net prefix/postfix for this node → N
  - Each node potentially in T slots
- Evaluate all timeslots can put in $O(NT^2)$
- N nodes to place
- $O(N^2T^2)$

List Scheduling

List Scheduling (basic algorithm flow)

- Keep a ready list of “available” nodes
  - (one whose predecessors have already been scheduled)
- Pick an unscheduled task and schedule on next available resource
- Put any tasks enabled by this one on ready list

List Scheduling

- Greedy heuristic
- How prioritize ready list?
  - Dominant constraint
    - least slack (worst critical path)
    - enables work
    - utilize most precious resource
- Last time:
  - saw that no single priority scheme would be optimal

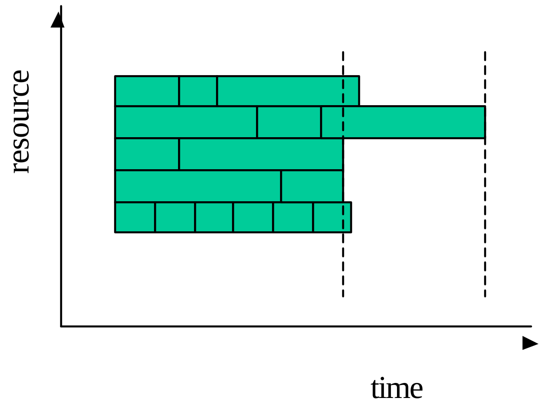
List Scheduling

- Use for
  - resource constrained
  - time-constrained
    - give resource target and search for minimum resource set
- Fast: $O(N) \rightarrow O(N\log(N))$ depending on prioritization
- Simple, general
- How good?

Approximation

- Can we say how close an algorithm comes to achieving the optimal result?
- Technically:
  - If can show
    - Heuristic(Prob)/Optimal(Prob)$\leq\alpha$ $\forall$ prob
    - Heuristic is an α-approximation

Scheduled Example Without Precedence



How bad is this schedule?

Observe

- Optimal length L
- No idle time up to start of last job to finish
- start time of last job <= L
- last job length <=L
- Total LS length <=2L
- Algorithm is within factor of 2 of optimum

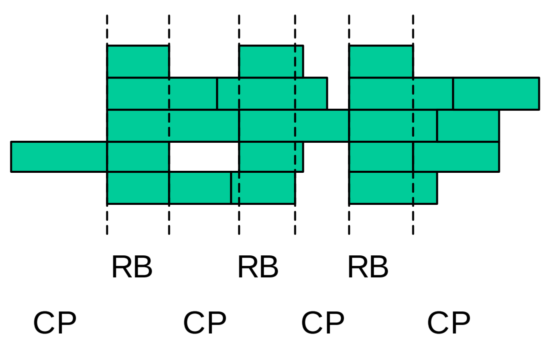
Results

- Scheduling of identical parallel machines has a 2-approximation
  - *I.e.* we have a polynomial time algorithm which is guaranteed to achieve a result within a factor of two of the optimal solution.
- In fact, for precedence unconstrained there is a 4/3-approximation
  - (schedule Longest Processing Time first)

Recover Precedence

- With precedence we may have idle times, so need to generalize
- Work back from last completed job
  - two cases:
    - entire machine busy
    - some predecessor in critical path is running
- Divide into two sets
  - whole machine busy times
  - critical path chain for this operator

Precedence



Precedence Constrained

- All busy times < Optimal Length
  - Optimal Length > Resource Bound
  - Resource Bound > All busy
- This path < Optimal Length
  - Optimal Length > Critical Path
  - Critical Path >= This Path
- List Schedule = This path + All busy times
- List Schedule <= 2 *(Optimal Length)

Tighten

- LS schedule <= Critical Path+Resource Bound
- LS schedule <= Min(CP,RB)+Max(CP,RB)
- Optimal schedule >=Max(CP,RB)
- LS/Opt <= 1+Min(CP,RB)/Max(CP,RB)

- The more one constraint dominates
  - the closer the approximate solution to optimal

Tightening

- Example of
  - More information about problem
  - More internal variables
  - …allow us to state a tighter result
- 2-approx for any graph
  - Since CP may = RB
- Tighter approx as CP and RB diverge

Multiple Resource

- Previous result for homogeneous functional units
- For heterogeneous resources:
  - also a 2-approximation
    - Lenstra+Shmoys+Tardos, Math. Programming v46p259
    - (not online, no precedence constraints)

Bounds

- Precedence case, Identical machines
  - no polynomial approximation algorithm can achieve better than 4/3 bound
    - (unless P=NP)
- Heterogeneous machines (no precedence)
  - no polynomial approximation algorithm can achieve better than 3/2 bound

Other Approaches

- Graph Coloring
  - Viable with pruning
- ILP
- Annealing

Summary

- Heuristic approaches to schedule
  - Force-directed
  - List Scheduling
- We can, sometimes, bound the “badness” of a heuristic
  - get a tighter result based on gross properties of the problem
  - approximation algorithms often a viable alternative to finding optimum
  - play role in knowing “goodness” of solution

Today’s Big Ideas:

- Exploit freedom in problem to reduce costs
  - (slack in schedules)
- Technique: estimation/refinement
- Use dominating effects
  - (constrained resources)
  - the more an effect dominates, the “easier” the problem
- Technique: Approximation