Scheduling

- Scheduling is the "problem of allocating scarce resources to activities over time." [Baker 1974]
- Typically, planning is deciding *what* to do, and scheduling is deciding *when* to do it.
- Generally, scheduling is determining how to accomplish some set of tasks while:
  - obeying task constraints (e.g., due dates, task interactions) [feasibility criteria]
  - minimizing resource usage (e.g., time, personnel) [optimization criteria]

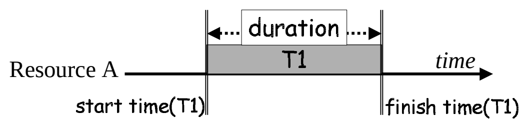
Feasibility

- Overconstrained:
  - Sometimes not all constraints can be honored (e.g., *soft* constraints)
  - decide which constraints to *relax* then decide when to do the tasks
- Oversubscribed:
  - Sometimes not all tasks can be accommodated given fixed resources.
  - decide *which* tasks to discard then decide when to do them

CS 540, Artificial Intelligence
© Adele Howe, Spring 2004

Task Characteristics

- Non-preemptive:
  - once started, must be allowed to finish.

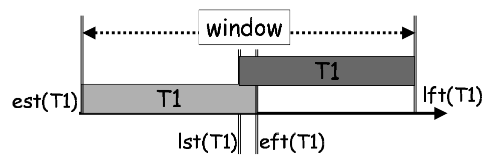


- Preemptive:
  - can be interrupted and re-started later.

CS 540, Artificial Intelligence
© Adele Howe, Spring 2004

Task Characteristics (cont.)

- Time windows:
  - must be scheduled within some interval

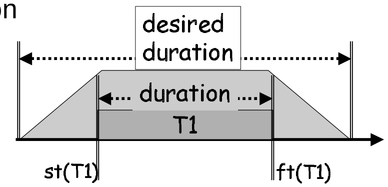


  - release dates: beginning of window
  - deadlines: end of window
  - slack: LST - EST

CS 540, Artificial Intelligence
© Adele Howe, Spring 2004

Task Characteristics (cont.)

- Processing time:
  - fixed duration
  - flexible: penalty function for less than desired duration



Task Characteristics (cont.)

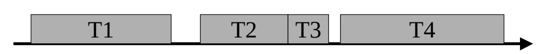
- Precedence Constraints:
  - Task A before Task B
  - start(B)-finish(A)>=0
- Resource Constraints:
  - must be on a particular resource or particular type of resource
- Alternatives
  - resources
  - times

Task Characteristics (cont.)

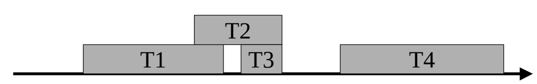
- Set-up Times
  - task dependent interaction effects
  - depending on what preceded task in schedule, it may take extra time to prepare resource to execute the task
  - e.g., making different colored plastics in a vat, extruding different viscosity materials, re-orienting antenna to receive signal

Resource Characteristics: Capacity

- Unit capacity: one task at a time



- Multi-capacity:
  - *n* tasks at a time
  - certain sum of task size requirements at a time



Resource Characteristics: Capacity Types

- consumable:
  - monotonically decreasing capacity
- renewable:
  - variable (up and down) capacity
- reusable:
  - fixed, unchangable capacity

Optimization Metrics

- Resource based
  - minimize number of resources used/cost of resources allocated
  - maximize resource utilization: percentage of time that resources are not idle
  - minimum peak resource usage
- Task based
  - cumulative value of tasks scheduled
  - cumulative value of time slots for tasks
  - maximum weighted # of tasks

Optimization Metrics (cont.)

- Time based

  Makespan: duration between start of first task and completion of last task

  Total Weighted Flow Time: sum of weighted finish times

  Maximum Tardiness: largest delay from desired finish time to actual end time for any task

  Total Weighted Tardiness: Sum of the delays between desired and actual finish times

  Total Weighted Number of Late/Discarded Tasks

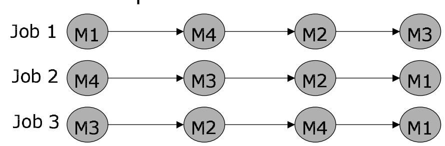
Scheduling Problem

- Given:
  - set of resources
  - set of tasks with constraints, temporal descriptions and resource requirements
  - an objective function
- find a solution (mapping of tasks to times on resources) that
  - satisfies constraints
  - minimizes objective function

Benchmark Application: Manufacturing Scheduling

Job Shop (JSP):

- n jobs on m machines
- each job must be processed on each machine exactly once for a fixed duration in a pre-defined order (job routing order).
- unit capacity, non-preemptive, no time windows or setups
- minimize makespan



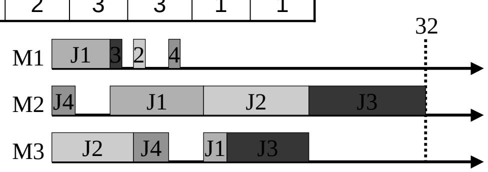
JSP (cont.)

- Solution:
  - is processing order for all jobs on each machine.
  - specifies est for each job/machine s.t. precedence and resource constraints are satisfied.
- (n!)^m possible solutions of which a subset are feasible.

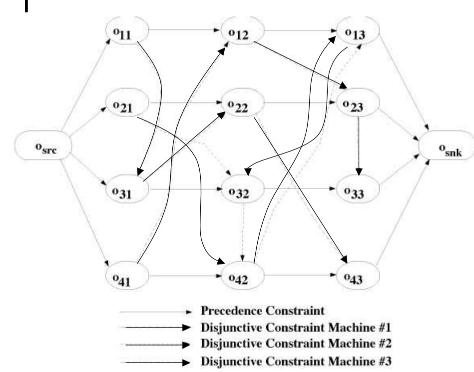
Example JSP

	Oi,1		Oi,2		Oi,3	
Job	Dur	Rout	Dur	Rout	Dur	Rout
1	5	1	8	2	2	3
2	7	3	3	1	9	2
3	1	1	7	3	10	2
4	2	2	3	3	1	1

Solution: 1,2,3,4



Example JSP (cont.)



JSP Taxonomy of Schedules

Feasible Solution: a job processing order on each machine without any cyclic dependencies

Feasible Schedule: specifies a start time s.t. all constraints are satisfied

Inadmissible: operations start later than their est

Semi-Active: some operations are scheduled at their est

Global left shift: move an operation earlier into machine idle time

Active: no global left shifts are possible (difficult to determine)

Non-Delay: no machine is idle if an operation is available

Optimal: minimizes the makespan

Algorithms for Scheduling

- Heuristic Constructive (e.g., LDS, HBSS)
- Constraint Programming (e.g.,dynamic programming, edge-finding)
- Local Search
- Genetic Algorithms

[Note: makespan computation tends to be most expensive aspect]

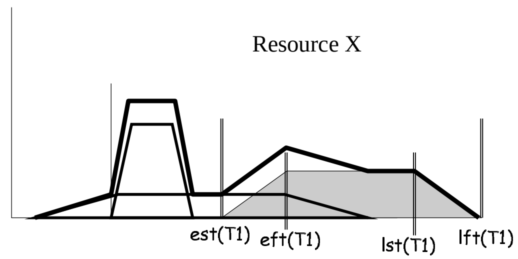
Slack Based Heuristics

- order operations in increasing (LST – EST)
- greedy version:
  - until all operations scheduled:
    - schedule open operation that has predecessors scheduled and minimizes slack at its EST
    - update ESTs and LSTs
- minslack: impose ordering on pairs of unsequenced tasks with minimal maximal slack:
  - (max(lft(A)-est(B), lft(B)-est(A)) – d(A) – d(B))

Texture Based Heuristics

- construct a profile of resource usage over time to identify and reduce contention
- SumHeight:
  - estimate probabilistically the competition of tasks on resources for each time
  - identify the point of highest contention
  - make ordering decision to reduce the contention

Texture Based Heuristics (cont.)



black lines are task demands, red line is aggregate demand

Constraint Programming

Edge Finding:

- determining whether an operation can, cannot, or must execute before (or after) other operations on a machine.
- Rules: (O is set of operations on machine, A is a particular operation, D is duration)

$$\forall O, \forall A \notin O, LFT_{O\cup\{A\}} - EST_O < \sum_{i\in O} D_i + D_A \Rightarrow A << O$$
$$\forall O, \forall A \notin O, LFT_O - EST_{O\cup\{A\}} < \sum_{i\in O} D_i + D_A \Rightarrow A >> O$$
$$A << O \Rightarrow end(A) \le \min_{i\in O}(LFT_i - D_i)$$
$$A >> O \Rightarrow start(A) \ge \max_{i\in O}(EST_i + D_i)$$

Edge-Finding (cont.)

1. Compute a schedule (*S*) by iteratively applying priority rule:
   - whenever a resource is free and an operation is available, schedule the operations with the smallest LFT
2. Given *S*, for each operation *A*,
   1. compute the set of operations (P) not finished at EST(A).
   2. For each P (in decreasing order of LFT(P)), update ordering and start/end times based on rules.

Local Search

- state of the art [Nowicki & Smutnicki 2003]
- solutions represented as disjunctive graph with edges only between adjacent operations
- JSP specific move operators (N5 for N&S03)
- initiate search from random semi-active or greedy solutions
- tabu search: checks for cycles in recent solutions (tabu tenure=8) and re-starts if cycles or if no progress over some time
- path relinking to generate new solutions for re-starts

An Aside: Critical Operations, Paths and Blocks

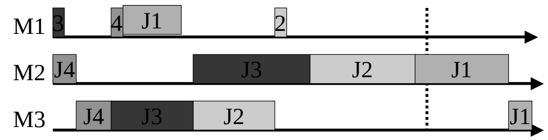
Critical Operation: operations in which est=lst. If start time is delayed, makespan will increase.

Critical Path: contiguous sequence of critical operations starting at t=0 and ending at makespan with no machine idle time.

Critical Blocks: subsequences of critical paths with all operations on the same machine.

Critical Operations, Paths and Blocks Example

Greedy Solution: 4,3,2,1



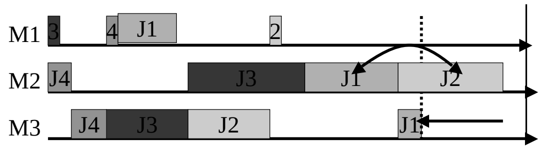
Critical operations:
J12, J13, J22, J32, J33, J42, J43

Critical path:
J42, J43, J33, J32, J22, J12, J13

Critical blocks:
{J43, J33}, {J32, J22, J12}

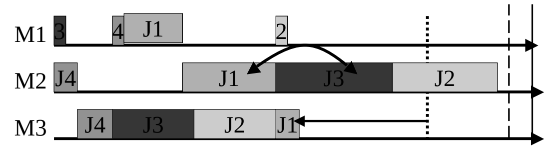
Critical Blocks Based Move Operators

- N1:
  - invert the order of a pair of adjacent operations on the same critical block
  - focuses on only 1) feasible neighbors and 2) those operations that can change the makespan
  - fully connected search space: there exists a path from any feasible solution to a global optimal

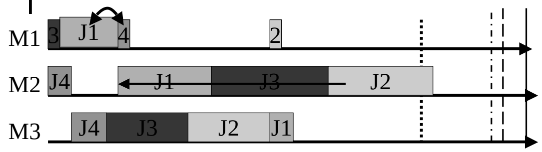


Critical Blocks Based Move Operators (cont.)

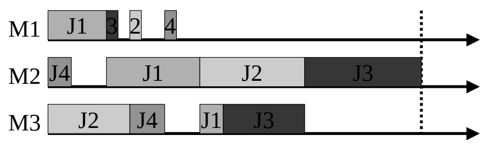
- N5, like N1 except
  - swap only adjacent operations that include either first or last operations in the critical block
  - do not swap either the first two operations in the first critical block or the last two operations in the last critical block
  - not completely connected



Effect of Moves (cont.)



Recall the optimal…



Genetic Algorithms

- Represent solution as permutation of tasks
- Requires a schedule builder to convert solution into schedule and assess objective function. [Indirect search]
- Syswerda's permutation crossover was used for scheduling.

Variations on JSP

- WorkFlow:
  - machines are partitioned into *wf* subsets
  - every job must be processed on all machines in a partition before moving to the next partition
- Flow Shop:
  - all jobs have identical order on all machines
  - workflow with *wf* = # machines

JSP Difficulty

- Analysis:
  - Optimization is NP-hard
  - No polynomial time algorithm can guarantee a solution within 20% of optimal makespan.
- Observations:
  - Given fixed *n* and *m*, workflow JSPs are typically more difficult than random JSPs.
  - Given fixed *n* and *m*, flowshop JSPs are typically more difficult than workflow JSPs.
  - Given fixed *m* and *wf*,square JSPs are typically more difficult than rectangular JSPs (more jobs than machines).
  - Given fixed *n*,*m* and *wf*, relative problem difficulty tends to be algorithm independent.