Scheduling

- Basic scheduling problems: open shop, job shop, flow job
- The disjunctive graph representation
- Algorithms for solving the job shop problem
- Computational complexity of the job shop problem

Open shop, job shop, flow shop scheduling

- Perform certain *jobs*, each consisting of *operations*.
- Each operation can be performed on one of *machines.* Operations have a *duration* (an integer). Each machine can handle one operation at a time.
- Objective: schedule operations so that
  1. time consumption is less than or equal some constant T, or
  2. time consumption is smallest possible.

Open shop, job shop, flow shop scheduling

1. Open shop: no ordering constraints on operations
2. Job shop: Operations of a job totally ordered
3. Flow shop: in each job exactly one operation for every machine, all jobs go through all the machines in the same order

Preemptive scheduling: no operation may be interrupted when it has already been started.

Related problems

- Planning.
- Others:
  - Course scheduling for schools (lecture halls, lecturers)
  - Timetabling for railways
  - Crew scheduling for airlines/railways etc.
  - Flight timetabling for airlines
  - Fleet assignment for airlines

Formalization of job shop scheduling

A problem instance $P = \langle M, O, J \rangle$ in job shop scheduling consists of

- a set M of machines,
- a set O of operations o, each associated with a machine $m(o) \in M$ and having a duration $d(o) \in \mathcal{N}$, and
- a set J of jobs $\langle o_1, \ldots, o_n \rangle$ (each operation has exactly one occurrence.)

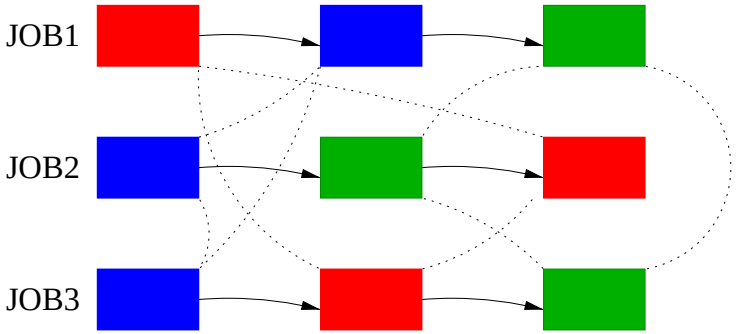
Formalization of job shop scheduling

A schedule S for P assigns to every operation o a time $b(o)$:

1. $b(o) \geq 0$ for all $o \in O$
2. $b(o) \geq b(o') + d(o')$ for operations o' preceding o in the same job
3. $b(o) \geq b(o') + d(o')$ or $b(o') \geq b(o) + d(o)$ for all $o' \in O$ with $m(o') = m(o)$ and $o \neq o'$

Schedule S has cost T if $b(o) + d(o) \leq T$ for all $o \in O$.

The (“disjunctive”) graph representation



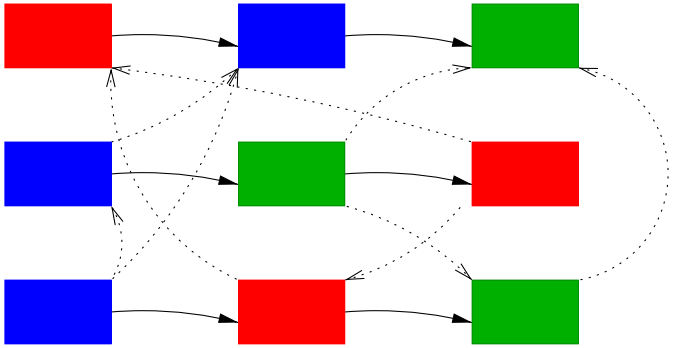
The dotted edges indicate that two operations are on the same machine, and one of the operations has to precede the other.

Finding a schedule proceeds as follows.

1. Assign a direction to every edge without introducing cycles.
2. Topologically sort the graph (total order.)
3. Assign starting and ending times to the operations.

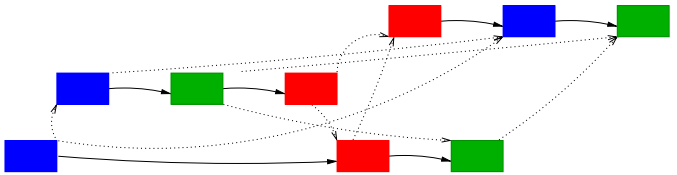
The topologically sorted graph determines the earliest possible starting and ending times of all operations uniquely.

One schedule for the problem instance



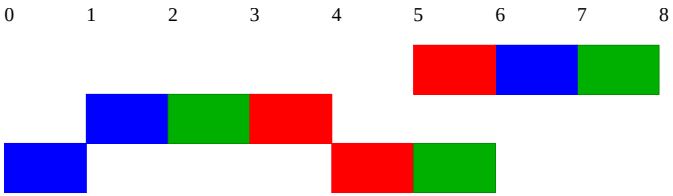
The ordering in the schedule

We draw the graph so that all edges go from left to right:



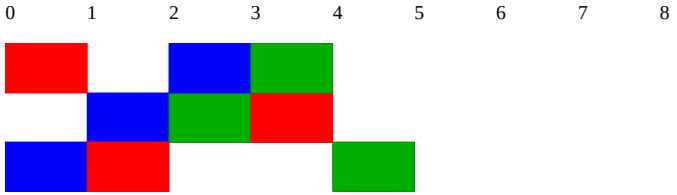
Assignment of time points to the schedule

Given the ordering of operations, assign all the operations the earliest possible time points (here all operations have duration one):



Assignment of time points to the schedule

Obviously, the preceding schedule is not the best possible. E.g. the following is much better.



Algorithms for scheduling

There are two main approaches to finding schedules:

1. branch and bound: systematic binary search in the space of all schedules,
2. local search: schedule is gradually improved.

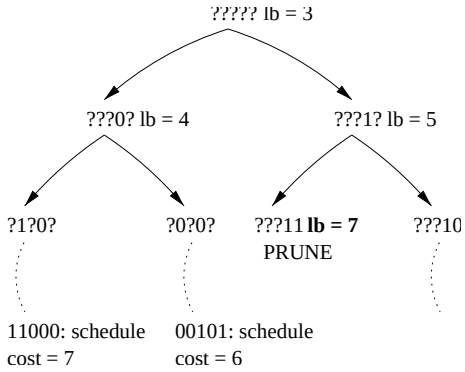
Both can be used with different schedule representations:

1. the disjunctive graphs (most popular representation), or
2. assignments of time points/intervals to operations.

Algorithms for scheduling: branch and bound

- Labels of search tree nodes are $x_1 x_2 \ldots x_n$, with $x_i \in \{0, 1, ?\}$ representing the undirected edges. ($\rightarrow$, $\leftarrow$, undecided.)
- One child assigns 0 to x_i and the other assigns 1.
- Search tree is pruned by computing lower bounds on the cost.
- If the graph becomes cyclic or lower bound exceeds cost of the best schedule so far, prune the subtree.
- When all x_i have value 0 or 1, we have found a schedule.

Branch and bound: an example



Lower bounds of schedule cost

Given a disjunctive graph, define for an operation o

- head(o): time necessarily needed before processing o

  Highest duration of a directed path that ends in o
- tail(o): time necessarily needed after processing o

  Highest duration of a directed path that starts from o

Lower bounds of schedule cost (cont'd)

Define for a set S of operations

- the shortest head $H(S) = \min_{o \in S} \text{head}(o)$
- the shortest tail $T(S) = \min_{o \in S} \text{tail}(o)$
- the sum of processing times $P(S) = \sum_{o \in S} d(o)$

Lower bounds of schedule cost (cont'd)

Given a set of operations S on one machine, $H(S)+P(S)+T(S)$ is a lower bound on the cost of the schedule:

- Operations S cannot overlap because they are on the same machine: at least time $P(S)$ is needed for processing S.
- If from operations in S the one with the smallest head is performed first, at least time $H(S)$ is needed before S.
- If from operations in S the one with the smallest tail is performed last, at least time $T(S)$ is needed after S.

Lower bounds of schedule cost (cont'd)

Let O_m be the set of operations on machine m.

Now a lower bound on the cost of schedule is

$$\max_{m \in M} \left(\max_{S \subseteq O_m} H(S) + P(S) + T(S) \right)$$

In other words, we compute the lower bounds on all sets S of operations that are computed on one machine.

Algorithms for scheduling: local search

- Idea: two schedules are neighbors if one can be obtained from the other by a small modification (to its graph).
- Modifications:
  - reverse an arrow, or
  - reorder consecutive operations in the graph (preserving their locations in their respective jobs.)

  Modifications must preserve acyclicity

Algorithms for scheduling: local search

Finding good schedules proceeds as follows:

1. Start from an randomly chosen schedule.
2. Go from the current schedule to a neighboring schedule (if the neighboring schedule is sufficiently good.)
3. Algorithms: simulated annealing, tabu search, ...

Computational intractability of scheduling

Optimal solutions for job shop scheduling can be found polynomial time if

- number of jobs is 2,
- number of machines is 2, all jobs have 1 or 2 operations, or
- number of machines is 2, all operations have duration 1.

In all cases the problem obtained by incrementing the number of machines, jobs, operations or durations by 1, is NP-hard.

Approximability of job shop scheduling

THEOREM (Williamson et al. 1993) Deciding if there is a schedule of length 4 is NP-complete.

COROLLARY There is no polynomial-time algorithm that finds schedules of length $< \frac{5}{4}$ from optimal (unless P=NP.)

PROOF SKETCH: A schedule of length 4 exists if and only if p-approximation algorithm with $p < \frac{5}{4}$ finds a schedule of length 4. (Schedule of length 5 would be more than p from the optimal.)

THEOREM (Shmoys et al. 1994) There is a poly-time algorithm that produces schedules of length $\frac{\log^2 m}{\log \log m}$ times the optimal.