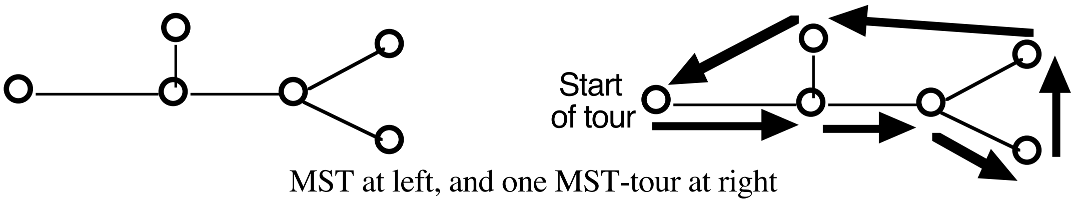
Ch 7 Approximation algorithms, online algorithms

Bertrand Russell (1672-1970): Although this may seem a paradox,
all exact science is dominated by the idea of approximation.

7.1 Minimum spanning tree approximation to the traveling salesman problem (TSP)

Algorithm: MST approx to TSP: Given a weighted complete graph G(V, E, w: E -> Reals), where w satisfies the triangle inequality: for all i, j, k: $w_{ik} \leq w_{ij} + w_{jk}$
Construct a minimum spanning tree MST and use it to construct an MST-tour as the figure shows.



MST at left, and one MST-tour at right

Constructing an MST-tour is most easily visualized as walking around the outside of a tree embedded in the plane. If this walk takes us to a vertex already visited, continue walking until you encounter an unvisted vertex, then draw a shortcut from the last vertex visited to the newly discovered unvisited vertex. But this construction requires no geometric properties, only the triangle inequality.

Start the tour at any vertex, say v1. Mark v1 as "visited" and call it the current vertex v. Pick any neighbor of v and call it the tentative vertex v'. If v' is not yet marked "visited", mark it "visited"and call it the current vertex v. If, on the other hand, v' is already marked "visited", replace v' by any of its neighbors, call this neighbor the tentative vertex v', and proceed, until all vertices are marked "visited"

From the construction we obtain the inequality: | MST-tour | ≤ 2 | MST |

Consider any optimal TSP, call it OPT. Any tour minus any of its edges is a tree, and by definition, at least as costly as an MST. This observation applied to OPT yields:

| OPT minus any of its edges | ≥ | MST | since we don't know what edges are in OPT ....
| OPT | ≥ | MST | + | a shortest edge in G | and using | MST | ≥ | MST-tour | / 2
| OPT | ≥ | MST-tour | / 2

Thus, the MST-tour heuristic is a 2-approximation algorithm.

7.2 Vertex cover

Consider a graph G(V, E), or a weighted graph G(V, E, w: E -> Reals).

Df: a subset C of V is a **vertex cover** of G(V, E) iff every edge e in E has at least one endpoint in C.

The bound problem: "is there a cover C with at most k nodes?" i s NP-complete.
The optimization problem: "construct a minimum cover" is NP-hard.

Greedy algorithm "Check each edge":
C <- {}; for i = 1 to m do if ei is **not yet covered**, place **both endpoints** of ei in C.

Thm: "Check each edge" is a 2-approximation algorithm.
Proof: The greedy algorithm adds **2** vertices to the cover C when an edge e is selected as "not yet covered". Due to this "overkill", edges selected "never touch each other", i.e. they are never incident to a common vertex (the edges selected form a matching in G). Thus, each pair of vertices of the cover C constructed by the greedy algorithm can be uniquely associated with "their own" selected edge. Of the 2 vertices of any such pair, any cover, including a minimum cover, must include at least 1. QED

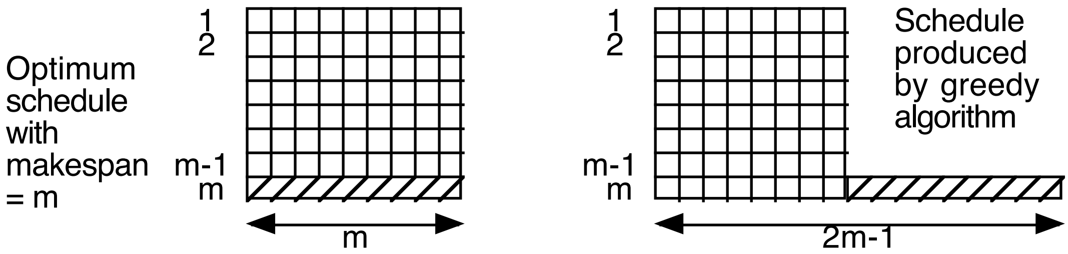
7.3 Online algorithms and competitive analysis

List scheduling, minimum makespan scheduling (R. Graham, 1966)

Given a sequence of n Jobs J1, .. , Jj, .., Jn and m machines M1, .. , Mi, .. , Mm, all identical, each machine can handle any job, but only one job at a time. The duration dj needed to process job Jj on any machine is given. Starting at time t = 0, schedule the jobs in the order j = 1, ... , j = n of the given sequence. This is called an **online algorithm** because, when you schedule any Jj to start at time sj on some machine Mi, you do not as yet know anything about the remaining jobs still to arrive. Define the **makespan** as the time when the last job finishes.

Greedy online scheduling algorithm G:
Schedule the next job in sequence as soon as possible on any available machine.

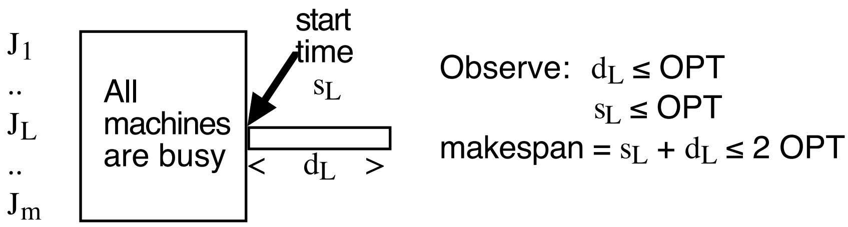
Ex: consider a sequence of m (m-1) jobs of duration 1, followed by a last job of duration m.
As the figure at left shows, an algorithm that analyzes the entire input before scheduling can produce an optimum schedule with makespan = m. An online algorithm, on the other hand, is handicapped by the fact that it sees the longest job only after it has scheduled every other job. The greedy algorithm G produce a schedule with makespan = 2m-1. In this example, the ratio R of the solution produced by G and an optimal schedule is R = (2m -1) / m = 2 - 1/m . We will see that this ratio R = 2 - 1/m is a worst case bound. G never needs more time than R = 2 - 1/m times what an optimal algorithm OPT requires, even though OPT inspects the entire input sequence before scheduling.



Thm: G is a 2-approximation algorithm, i.e. for any input I, $R = G(I) / OPT(I) \leq 2$.

Pf: Consider any job J_L that finishes last, i.e. defines the makespan. It was started at some time s_L , requires duration d_L , and finishes at time makespan = $s_L + d_L$. At time s_L all machines must have been busy, or else J_L would habe started earlier. Since an optimal algorithm OPT cannt do better than keeping all the machines busy at all time, we have $s_L \leq OPT$. And obviously, $d_L \leq OPT$. Adding these two inequalities yields:

makespan = $s_L + d_L \leq 2$ OPT.



Sharper version of the Thm: G is a (2 - 1/m)-approximation algorithm.
The proof is similar, justify and add the inequality: (Sum of all dj) / m $\leq$ OPT