Chapter 16. Greedy Algorithms

Algorithms for optimization problems (minimization or maximization problems) typically go through a sequence of steps, with a set of choices at each step.

A greedy algorithm always makes the choice that looks the best at the current step. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

For some optimization problems, the greedy algorithm does not yield optimal solutions but for many problems, it does.

Examples of greedy algorithms that do not work:

- Shortest path through a layered network: Construct a path by always adding an edge of shortest length.
- Matrix multiplication chain: Repeatedly do the cheapest of the available multiplications.

16.1. Incompatible task scheduling

Suppose that there are n tasks $T_1, T_2, \ldots, T_n$, where task T_i must start at time s_i and finish at time f_i with $s_i \leq f_i$. No two tasks can be performed at the same time (as there is only one CPU). That is, two tasks T_i and T_j are **incompatible** if their time intervals (s_i, f_i) and (s_j, f_j) overlap. The problem is to perform as many tasks as possible.

- *First attempt at greedy solution:*
  Repeatedly choose the earliest starting task that is compatible with previously chosen tasks. **This doesn't work.**

- *Second attempt at greedy solution:*
  Repeatedly choose the earliest finishing task that is compatible with previously chosen tasks. This schedule algorithm usually is refereed as the EDF algorithm. **This works!**

16.1. Incompatible task scheduling (continued)

Theorem. *This greedy solution delivered by the EDF algorithm is optimal:* Repeatedly choose the earliest-finishing task that is compatible with previously chosen tasks.

Proof. Let $\langle S_1, S_2, S_3, \ldots, S_k \rangle$ be any solution (including the optimal solution) to the incompatible task scheduling problem, where S_i is the choice at the ith step.
Let $\langle G_1, G_2, G_3, \ldots G_k \rangle$ be the greedy solution.

According to the greedy rule, G_1 finishes no later than S_1. Therefore, S_2 is compatible with G_1, so G_2 finishes no later than S_2, S_3 is compatible with G_2, and so on. In general, for $1 \leq i < k$, G_i finishes no later than S_i, and so S_{i+1} is compatible with $G_1, \ldots, G_i$. Therefore, the greedy solution can be continued for another task G_{i+1}, which finishes no later than S_{i+1}. Thus, the solution consisting of $G_1, \ldots, G_k$ is at least as good as the solution consisting of $S_1, \ldots, S_k$.

16.1. Incompatible task scheduling (continued)

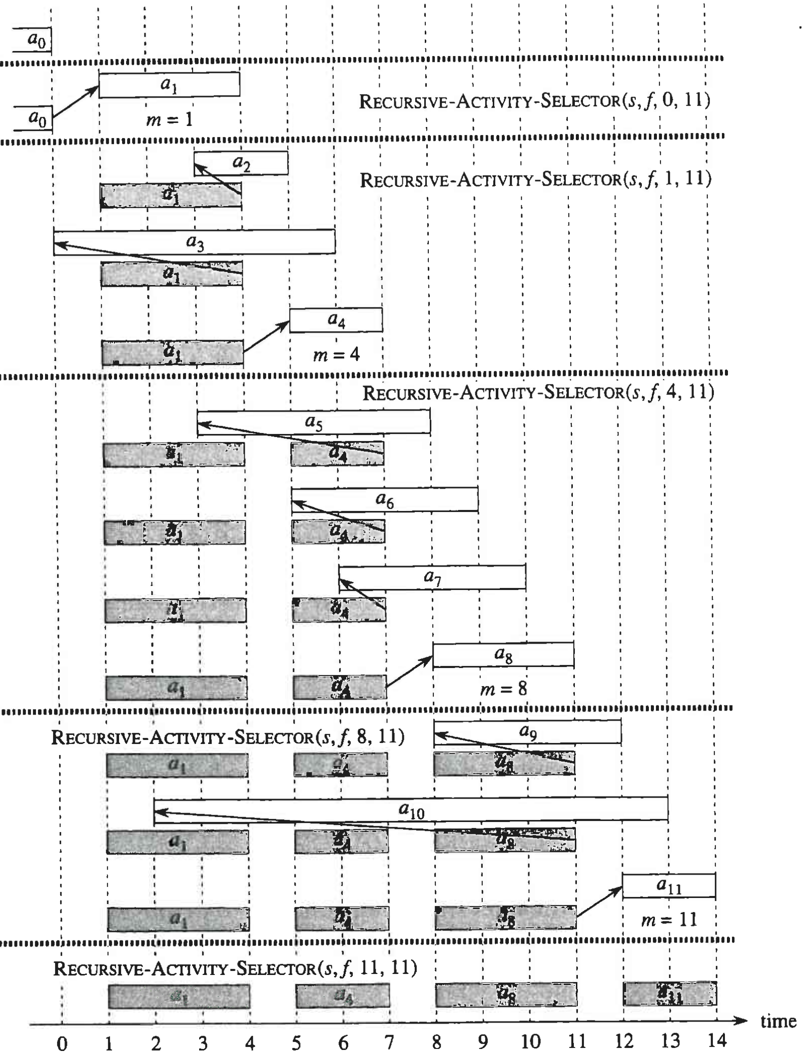
The greedy solution has at least as many tasks as any other solution. It thus must have the maximum possible number of tasks. We have also proved that the greedy solution finishes no later than any of optimal solutions.

Greedy_CPU_Scheduling(s, f)

$A \leftarrow \{T_1\}$;
$j \leftarrow 1$;
for $i \leftarrow 2$ to n
  do if $s_i \geq f_j$
    then $A \leftarrow A \cup \{T_i\}$;
      $j \leftarrow i$;
return A.

To implement the algorithm, sort the tasks in order of finishing time. Then, for each task in the list, execute it if its starting time has not already past, and wait until it is finished before continuing. Time: $O(n \log n)$ for sorting, $O(n)$ for the rest.

k	s_k	f_k
0	–	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14
12	∞	–



Load balancing problem

Given a set of m machines $M_1, M_2, \ldots, M_m$ and a set of n jobs, each job j has a processing time $t_j > 0$ with $1 \le j \le n$. We seek to assign each job to one of the m machines so that the loads placed on all machines are as "balanced" as possible, where the load at a machine is the sum of processing times of all jobs allocated to the machine.

Unfortunately this problem is NP-hard, i.e., it is unlikely to be solved in polynomial time unless P=NP. Instead, we aim to find *a feasible solution* to it, and if we are able to show that there is a certain degree of guarantee between the feasible solution and its optimal solution, then we call this algorithm is *an approximation algorithm* for the load-balancing problem.

Load balancing problem (cont.)

Let $A(i)$ denote the set of jobs assigned to machine M_i. Under an assignment, machine M_i needs to work for a total time of $T_i = \sum_{j \in A(i)} t_j$, which is the load at machine M_i for all i, $1 \leq i \leq m$. We seek to minimize a quantity known as the makespan, i.e., the maximum load among all machines $T = \max\{T_i \mid 1 \leq i \leq m\}$ is **minimized**. In other words, our objective is to

$$\textit{minimize} \quad \max\{T_i \mid 1 \leq i \leq m\},$$

The Greedy Strategy: Assign the current job j to a machine M_i with the minimum load at each time.

Load balancing problem (cont.)

Greedy_Balance

for $i \leftarrow 1$ to m **do**

    $T_i \leftarrow 0$;

    $A(i) \leftarrow \emptyset$;

endfor;

for $j \leftarrow 1$ to n **do**

    Let M_i be a machine achieving the minimum load,

        i.e. $T_i = \min\{T_{i'} \mid 1 \leq i' \leq m\}$;

    Assign job j to machine M_i;

    $A(i) \leftarrow A(i) \cup \{j\}$;

    $T_i \leftarrow T_i + t_j$;

endfor

Exercise: What's the running time of Algorithm **Greedy_Balance**?

Load balancing problem (cont.)

Lemma: Let T^* be the optimal makespan (load), then

(i) $T^* \geq \frac{1}{m}\sum_{j=1}^{n} t_j$;

(ii) $T^* \geq \max\{\, t_j \mid 1 \leq j \leq n\}$, as each job is not allowed to be partitioned into multiple machines.

Load balancing problem (cont.)

Theorem: Algorithm Greedy-Balance produces an assignment of jobs to machines with makespan $T \leq 2T^*$, where T and T^* are the loads delivered by the greedy algorithm and an optimal load of the problem.

Proof: We assume that machine M_i attains the maximum load T in our assignment and job j is the last job assigned to machine M_i. The load of M_i then is the smallest prior to the addition of job j, which is $T_i - t_j$, and every other machine has a load at least $T_i - t_j$. Thus, we have

$\sum_{k=1}^{m} T_k \geq m(T_i - t_j)$, or $T_i - t_j \leq \frac{1}{m}\sum_{k=1}^{m} T_k = \frac{1}{m}\sum_{j=1}^{n} t_j$.

We thus have $T_i - t_j \leq \frac{1}{m}\sum_{j=1}^{n} t_j \leq T^*$ by the lemma.
As we assume that the makespan T is equal to T_i, we have

$$T = T_i = (T_i - t_j) + t_j \leq T^* + T^* = 2T^*.$$