

## 1 The Greedy Framework

Recall that an *optimization problem* is one for which an input has a collection of *feasible* solutions, each with an associated cost, and we need to find a feasible solution that *optimizes* the cost; here ‘optimizes’ would mean either *minimizes* or *maximizes*, depending on the nature of the problem. Such a solution is called an *optimal solution*.

A *greedy strategy* for an optimization problem constructs an optimal solution (which is typically a set of elements) *incrementally* by making a locally optimal choice at each step to decide the next element to be added to the solution. If this strategy generates an optimal solution, then this represents a *greedy algorithm*.

## 2 A Toy Problem

### Activity Selection Problem

Given a collection of activities  $S = \{(s_i, f_i), 1 \leq i \leq n\}$ , where  $s_i$  and  $f_i$  are the start and finish times of the  $i$ th activity, and  $f_1 \leq f_2 \leq \dots \leq f_n$ , a *feasible solution* is a subset of activities whose durations do not overlap.

A feasible solution with the maximum number of activities is an *optimal solution*.

### Algorithm Greedy Activity Selection

Start with an initially empty solution  $A$ . Let  $T := S$

**repeat**

- Select an activity  $\alpha$  in  $T$  with smallest finish time and set  $A := A \cup \{\alpha\}$ .
- Delete from  $T$  those activities whose durations overlap with the selected activity.

**until**  $T = \phi$

### Correctness

**Lemma 1 (Greedy Choice):** Let  $X$  be a subset of an optimal solution for the activity selection problem on  $S$ . Let  $S'$  be the set of activities in  $S$  that do not overlap with any activity in  $X$ , and let  $\alpha = (s, f)$  be the activity with smallest finish time in  $S'$ . Then,  $X' = X \cup \{\alpha\}$  is a subset of an optimal solution for  $S$ .

**Proof.** Clearly  $X'$  is a feasible solution since none of the activities in  $X'$  overlap with one another. Consider an optimal solution  $A$  for  $S$  that contains  $X$ . (We are given that such an optimal solution exists.)

If  $A$  contains  $\alpha$  then we are done. Otherwise let  $\beta = (s', f')$  be the activity with smallest finish time in  $A - X$ . Then,  $f' > f$  since  $(s, f)$  is the activity with smallest finish time that does not overlap with any activity in  $X$ . Since  $\alpha$  is in  $S'$ , it does not overlap with any activity in  $X$ . Also,  $\alpha$  does not overlap with any activity in  $A - X - \{\beta\}$  since any activity in  $A - X - \{\beta\}$  must have a start time greater than  $f'$  and the finish time  $f$  of  $\alpha$  is smaller than  $f'$ . Hence  $\alpha$  does not overlap with any activity in  $A - \{\beta\}$ .

Now consider the set  $A' = A - \{\beta\} \cup \{\alpha\}$ . This is a feasible set since none of the activities in  $A'$  overlap with each other. But  $|A'| = |A|$ , and since  $A$  is an optimal solution,  $A'$  is also an optimal solution, and is an optimal solution that contains  $X'$ .  $\square$

**Theorem:** Algorithm **Greedy Activity Selection** constructs an optimal solution for the activity selection problem.

**Proof:** For the repeat loop we will use the loop invariant that the solution set  $A_i$  at the start of  $i$ th iteration of the repeat loop is a subset of an optimal solution, and that  $T_i$ , the set of elements in  $T$  at the start of the  $i$ th iteration is the set of elements in  $S$  that do not overlap with any element in  $A_i$ .

**Initialization:**  $A_1 = \phi$  by the initialization in the algorithm. Hence  $A_1$  is clearly a subset of an optimal solution. Also, by the initialization  $T_1 = S$ , which is correct since no element in  $A_1$  overlaps with an activity in  $S$ .

**Maintenance:** Assume that the loop invariant holds at the start of the  $i$ th iteration. Hence  $A_i$  is a subset of an optimal solution and  $T_i$  contains all elements in  $S$  that do not overlap with any element in  $A_i$ .

Let  $\alpha = (s, f)$  be the activity that is added to  $A$  in the  $i$ th iteration of the repeat loop. Since  $\alpha$  is the activity with smallest finish time in  $T_i$ , by Lemma 1,  $A_i \cup \{\alpha\}$  is a subset of an optimal solution, hence  $A_{i+1}$  is a subset of an optimal solution. In the second step of the repeat loop any element in  $T_i$  that overlaps with  $\alpha$  is removed from it, hence  $T_{i+1}$  contains the elements in  $S$  that do not overlap with any element in  $A_{i+1}$ .

**Termination:** The repeat loop must terminate after at most  $|S|$  iterations since at least one element is deleted from  $S$  in each iteration. At termination,  $A$  is a subset of an optimal solution. Further, every element in  $S - A$  was deleted from  $S$  because it overlapped with some element in  $A$ , hence no other element can be added to  $A$  and still maintain feasibility. Hence  $A$  is an optimal solution for  $S$ .  $\square$

### 3 Minimum Spanning Tree

Given a connected undirected graph  $G = (V, E)$ , with a real-valued weight  $w(e)$  on each edge  $e \in E$ , a *minimum spanning tree (MST)* of  $G$  is a spanning tree  $T$  of  $G$  whose weight  $w(T)$  is minimum, where  $w(T)$  is given by

$$w(T) = \sum_{e \in T} w(e)$$

We now derive a greedy characterization of MST, from which we can derive several different efficient greedy MST algorithms. We start with some definitions.

A *cut* in an undirected graph  $G = (V, E)$  is a partition  $(S, V - S)$  of the vertex set  $V$ .

An edge  $(x, y)$  *crosses* cut  $(S, V - S)$  if one endpoint, say  $x$ , is in  $S$  and the other endpoint  $y$  is in  $V - S$ .

An edge  $e$  is a *light edge* crossing cut  $(S, V - S)$  if  $e$  crosses  $(S, V - S)$ , and  $w(e)$  is minimum among all edges crossing cut  $(S, V - S)$ .

**Theorem 3.1** *Let  $A$  be a subset of edges in an MST of  $G$ . If  $(S, V - S)$  is a cut such that no edge in  $A$  crosses the cut, and if  $e$  is a light edge crossing this cut, then  $A \cup \{e\}$  is also a subset of edges in an MST of  $G$ .*

**Proof:** (Sketch.) Let  $T$  be an MST of  $G$  that contains  $A$ . Let  $e = (x, y)$ . If  $e$  is not in  $T$ , then let  $f$  be an edge in  $T$  that crosses  $(S, V - S)$ , and lies on the path in  $T$  connecting  $x$  and  $y$ . Then  $T' = T - \{f\} \cup \{e\}$  is an MST of  $G$  that contains  $e$ .  $\square$

Theorem ?? immediately gives a polynomial-time algorithm to find an MST of a graph, though not necessarily a very efficient one. By suitably specializing the application of the theorem, *and by using suitable efficient data structures*, we can obtain efficient algorithms for MST. These include Kruskal's algorithm and Dijkstra-Jarník -Prim algorithm, both of which run in  $O(m \log n)$  time using simple data structures when the graph is represented by adjacency lists, where  $m$  is the number of edges in the graph and  $n$  is the number of vertices in the graph.

### 3.1 Kruskal's algorithm

BASIC-KRUSKAL( $G = (V, E), w$ )

**Input.** Undirected connected graph  $G = (V, E)$  with a weight function  $w : E \rightarrow R$ .

**Output.** A set  $A$  that contains the edges in an MST of  $G$ .

1.  $A := \phi$
2. Sort  $E$  in nondecreasing order of  $w(e)$  as  $e_1 = (u_1, v_1), \dots, e_m = (u_m, v_m)$
3. **for**  $1 \leq i \leq m$  **do**
4.     **if**  $u_i$  and  $v_i$  are not connected in the graph  $T = (V, A)$  **then**
5.          $A := A \cup \{e_i\}$
6.     **fi**
7. **rof**
8. **Return**  $A$

**Correctness.** For  $1 \leq i \leq m + 1$ , let  $E_i = \{e_1, \dots, e_{i-1}\}$  and let  $A_i$  be the set  $A$  at the start of iteration  $i$ .

Correctness can be proved (using Theorem ??) with the following loop invariant for the **for** loop starting on line 4:

- At the start of the  $i$ th iteration,  $T_i = (V, A_i)$  is a minimum spanning forest of the graph  $H_i = (V, E_i)$ .

**Running time.** We will see an efficient implementation of Kruskal's algorithm (for sparse graphs) later when we study data structures, but at this time you should be able to come up with an  $O(n^2 + m \log m)$  time implementation of BASIC-KRUSKAL, where  $n = |V|$  and  $m = |E|$ .

## 3.2 Dijkstra-Jarník -Prim algorithm

BASIC-DJP( $G = (V, E), w, r$ )

**Input.** Undirected connected graph  $G = (V, E)$  with a weight function  $w : E \rightarrow R$ , a specified vertex  $r \in V$  (the *root*).

**Output.** A parent pointer  $\pi(v)$  for each  $v \in V - \{r\}$  that points to another vertex in  $V$  such that the set  $\{(v, \pi(v)) \mid v \in V - \{r\}\}$  contains exactly the edges in an MST of  $G$ .

1. **for** each  $u \in V$  **do**  $key[u] : \infty; \pi[u] := NIL$  **rof**
2.  $key[r] := 0$
3.  $Q :=$  the set  $V$ , with each  $u \in V$  having the *key* values from steps 1 and 2
4. **while**  $Q \neq \emptyset$  **do**
5.     remove a vertex  $u$  with minimum *key* value from  $Q$
6.     **for** each  $v \in Q$  **do**
7.         **if**  $w((u, v)) < key[v]$  **then**
8.              $\pi[v] := u$
9.             Change the key value of  $v$  to  $w((u, v))$
- fi**
- rof**
- end(while)**

**Correctness.** Using Theorem ?? we can prove correctness of algorithm DJP with the following loop invariant for the **while** loop starting on line 4:

At the start of each iteration of the **while** loop,

- let  $S$  be the set of vertices that have been removed from  $Q$ , i.e.,  $S = V - V(Q)$ , and
- let  $A = \{(\pi(v), v) \mid v \in S - \{r\}\}$ .

Then,

- Edges in  $A$  form a tree on  $S$  rooted at  $r$ , which is a subtree of an MST of  $G$ ;
- for each  $v \in Q$ ,
  - if  $\pi[v] = NIL$  then there is no edge in  $G$  between  $v$  and a vertex in  $S$ , and  $key[v] = \infty$  if  $v \neq r$ ;
  - if  $\pi[v] \neq NIL$  then  $\pi[v]$  is in  $S$ , and  $(v, \pi(v))$  is a minimum-weight edge in  $G$  that connects  $v$  to a vertex in  $S$ , and the weight of that edge is  $key[v]$ .

**Running time** of BASIC-DJP is  $O(n^2)$ .

Observe that instead of executing the **for** loop in step 6 for all  $v \in Q$ , it suffices to execute it only on vertices  $v \in Adj[u] \cap Q$ . We will use this feature later together with a good data structure to reduce the running time of this algorithm when the graph is sparse.

## Single-source Shortest Paths

Recall that in the *single-source shortest path (SSSP)* problem, we are given a *source vertex*  $s \in V$ , and we need to compute shortest paths from  $s$  to every vertex in  $V$ .

The paths in the SSSP problem can be organized as an out-tree rooted at  $s$  that contains all vertices reachable from  $s$ , as we will see in Dijkstra's algorithm.

Dijkstra's algorithm is a fast, greedy algorithm for the SSSP problem for the case when the edge-weights are non-negative.

BASIC-DIJKSTRA( $G = (V, E), w, s$ )

**Input.** Directed graph  $G = (V, E)$  with a non-negative weight function  $w$  on the edges, a specified source vertex  $s \in V$ .

**Output.** A parent pointer  $\pi(v)$  for each  $v \in V - \{s\}$  that points to the predecessor of  $v$  in a shortest path from  $s$  to  $v$ , and for each vertex  $v$ ,  $d[v]$  contains the shortest path length from  $s$  to  $v$ .

```
1. for each  $u \in V$  do  $d[u] := \infty$ ;  $\pi[u] := NIL$  rof
2.  $d[s] := 0$ 
3.  $Q :=$  the set  $V$  with each  $u \in V$  having the  $d$  value from steps 1 and 2
4. while  $Q \neq \emptyset$  do
5.     remove a vertex  $u$  with minimum  $d$  value from  $Q$ 
6.     for each  $v \in Adj[u]$  do
7.         if  $v \in Q$  and  $d[u] + w((u, v)) < d[v]$  then
8.              $\pi[v] := u$ 
9.             Change the value of  $d[v]$  to  $d[u] + w(u, v)$ 
        fi
    rof
end(while)
```

**Correctness.** We prove correctness of algorithm DIJKSTRA with the following loop invariant for the **while** loop starting on line 4:

At the start of each iteration of the **while** loop,

- let  $S$  be the set of vertices that have been removed from  $Q$ , i.e.,  $S = V - V(Q)$ , and
- let  $A = \{(\pi(v), v) \mid v \in S - \{s\}\}$ .

Then,

- $A$  forms a rooted out-tree on  $S$  with root  $s$ , and the path in  $A$  from  $s$  to each vertex  $u$  in  $S$  is a shortest path in  $G$  from  $s$  to  $u$ , with  $d[u] = \delta(s, u)$ ;
- for each  $v \in Q$ ,
  - if  $\pi[v] = NIL$  then there is no edge from a vertex in  $S$  to  $v$ , and  $key[v] = \infty$  if  $v \neq s$ ;
  - if  $\pi[v] \neq NIL$  then  $\pi[v]$  is in  $S$ , and *among all paths from  $s$  to  $v$  in  $G$  that use only vertices in  $S$  as intermediate vertices*, the path from  $s$  to  $\pi(v)$  followed by the edge  $(\pi(v), v)$  has minimum cost, and the cost of that path is  $d[v]$ .

**Running time** of BASIC-DIJKSTRA is  $O(n^2)$ .