Mathematics for Decision Making: An Introduction

Lecture 12

Matthias Köppe

UC Davis, Mathematics

February 12, 2009

Dijkstra's Algorithm: Efficiency, IV

We now determine the precise number of elementary operations.

- We use the constants c_i associated with the data structures, which appeared to the previous slide.
- We use additional constants d_i to denote the number of elementary operations in other parts of the program.

Dijkstra's Algorithm

Input: A digraph $G = (V, A)$ with nonnegative arc costs, starting node r
Output: A predecessor vector **p**, encoding minimum-cost paths from r to all nodes.

1. Initialize **y**, **p** — $c_1 + d_1 + |V|(2c_4 + d_2)$ operations
2. Set $S := V$. — $d_4 + |V|(c_7 + d_3)$ operations
3. While $S \neq \emptyset$: — $|V|$ iterations and $(c_5 + d_5)(|V| + 1)$ operations
    - Choose $v \in S$ with y_v minimum. — $d_6 + |S|(c_4 + c_6 + d_7)$ operations
    - Set $S := S \setminus \{v\}$. — c_8 operations
    - For all arcs $(v, w) \in A$: — $\delta^+(v)$ iterations, $c_2 + \delta^+(v)c_3$ operations
        - If $y_v + c(v, w) \leq y_w$: — $2c_4 + d_8$ operations
            - $y_w := y_v + c(v, w)$ — c_4 operations
            - $p(w) := v$ — c_4 operations

Dijkstra's Algorithm: Efficiency, V

Adding up everything:

- The minimum-finding operation takes $d_6 + |S|(c_4 + c_6 + d_7)$ operations, where $|S|$ starts with $|V|$ and is decreased until it reaches 1. Thus its total time is:

$$\sum_{s=1}^{|V|} \left(d_6 + |S|(c_4 + c_6 + d_7)\right) = |V|d_6 + \frac{|V|(|V|+1)}{2}(c_4 + c_6 + d_7)$$

- All node-scanning operations (verifying all outgoing arcs) together take

$$\sum_{v \in V} \left(c_2 + \delta^+(v)(c_3 + 4c_4 + d_8)\right) = |V|c_2 + |A|(c_3 + 4c_4 + d_8)$$

- The remaining operations are easy to account for
- Together we obtain

$$e_1|V|^2 + e_2|V| + e_3|A| + e_4$$

  elementary operations, for some (complicated) constants e_i.
- **For sparse graphs, where $|A| \ll |V|^2$, the term $e_1|V|^2$ is the largest summand.** It comes from the minimum-finding operation!

Dijkstra's Algorithm: Efficiency, VI

- **We are not happy with the complicated analysis** (counting of operations, lots of constants, ...) we had to do to obtain this result.
- Moreover, the constants e_i we obtained still depend on the specific RAM we are using. For instance, on a version of a RAM with few registers, we might need more elementary operations to do the same thing.
- For these reasons, it is useful and convenient to **ignore the specific constants** and just ask **how does the running time grow for large problems** (i.e., asymptotically)
- We will use the **Landau notation** for asymptotic growth. Fix a function $g(n) \geq 0$.
  - A function $f(n) \geq 0$ is said to **grow (asymptotically) at most with order** $g(n)$ if

    $$\exists c > 0, n_0 \in \mathbf{N} : \forall n \geq n_0 : f(n) \leq cg(n).$$

    We use the notation $f(n) \in \mathrm{O}(g(n))$, this is read as "big oh of $g(n)$".
  - A function $f(n) \geq 0$ is said to **grow (asymptotically) at least with order** $g(n)$ if

    $$\exists c > 0, n_0 \in \mathbf{N} : \forall n \geq n_0 : f(n) \geq cg(n).$$

    We use the notation $f(n) \in \Omega(g(n))$, this is read as "big omega of $g(n)$".
  - A function $f(n) \geq 0$ is said to **grow (asymptotically) with order** $g(n)$ if $f(n) \in \mathrm{O}(g(n))$ and $f(n) \in \Omega(g(n))$ (note: different constants are allowed); we write $f(n) \in \Theta(g(n))$ (read: "big theta of $g(n)$")
  - Similarly, for functions of several arguments.

Dijkstra's Algorithm: Efficiency, VII

- Using Big-Oh notation, we obtain that the running time of our RAM implementation of Dijkstra's Algorithm is

$$\Theta(|V|^2).$$

  In particular, the number of arcs (and thus sparsity) is no longer visible.
- A Big-Oh calculus helps to simplify the expressions:
  - For example, any polynomial function $p(n) = \sum_{i=0}^{d} p_i n^i$ (with $p_d \neq 0$) is in $\Theta(n^d)$.
  - In particular, constants get consumed by higher-order terms
  - $\max\{f_1(n), f_2(n)\} \in \mathrm{O}(f_1(n) + f_2(n))$
- By keeping in mind that we are only interested in this kind of asymptotic estimate, we can simplify our counting of elementary operations: We can be "sloppy", in a controlled way.
  - It suffice to determine that some operation is $\mathrm{O}(1)$, or $\Theta(n)$; we don't need to discuss the precise number of iterations.

Dijkstra's Algorithm: Efficiency, VIIa

We now revisit the analysis of Dijkstra's Algorithm, and use Big-Oh estimates for the number of elementary operations, rather than the precise numbers.

Dijkstra's Algorithm

Input: A digraph $G = (V, A)$ with nonnegative arc costs, starting node r
Output: A predecessor vector $\mathbf{p}$, encoding minimum-cost paths from r to all nodes.

1. Initialize $\mathbf{y}$, $\mathbf{p}$ — $O(|V|)$ operations
2. Set $S := V$. — $O(|V|)$ operations
3. While $S \neq \emptyset$: — $O(|V|)$ iterations and $O(|V|)$ operations
    - Choose $v \in S$ with y_v minimum. — $O(|S|) \subseteq O(|V|)$ operations
    - Set $S := S \setminus \{v\}$. — $O(1)$ operations
    - For all arcs $(v, w) \in A$: — $O(\delta^+(v))$ iterations, $O(\delta^+(v))$ operations
        - If $y_v + c(v, w) \leq y_w$: — $O(1)$ operations
            - $y_w := y_v + c(v, w)$ — $O(1)$ operations
            - $p(w) := v$ — $O(1)$ operations

Now we immediately see that we have $O(|V|^2 + |A|) = O(|V|^2)$ elementary operations in total.

Dijkstra's Algorithm: Efficiency, VIII

- We are **still not happy** with the performance of Dijkstra's Algorithm for large, sparse graphs
- We have found the reason: Running time is (asymptotically) dominated by the minimum-finding operation.
- A solution is to use **better concrete data structures**. Here it pays off to use a **binary heap** (an implementation of a **priority queue**) to implement the set S together with the potential vector $\mathbf{y}$.
- A priority queue stores elements v together with a **priority** y_v; it has **operations**:
  - Empty?
  - Insert and element v with priority y_v
  - Find, remove, and return the element v of smallest priority y_v
  - Find a given element v, and change its priority to y'_v.
- The binary heap implementation of this abstract data structure on a RAM has running time of $\mathrm{O}(\log n)$ for all of these operations, where n is the number of elements stored.

Dijkstra's Algorithm with Binary Heaps: Efficiency

We now revisit the analysis of Dijkstra's Algorithm, using binary heaps.

Dijkstra's Algorithm

Input: A digraph $G = (V, A)$ with nonnegative arc costs, starting node r
Output: A predecessor vector $\mathbf{p}$, encoding minimum-cost paths from r to all nodes.

1. Initialize $\mathbf{y}$, $\mathbf{p}$ — $O(|V|)$ operations
2. Initialize a binary heap $S := V$ with priorities $\mathbf{y}$. — $O(|V|)$ operations
3. While $S \neq \emptyset$: — $O(|V|)$ iterations and $O(|V|)$ operations
    - Choose $v \in S$ with y_v minimum and $S := S \setminus \{v\}$. — $O(\log|S|) \subseteq O(\log|V|)$ operations
    - For all arcs $(v, w) \in A$: — $O(\delta^+(v))$ iterations, $O(\delta^+(v))$ operations
        - If $y_v + c(v, w) \leq y_w$: — $O(1)$ operations
            - $y_w := y_v + c(v, w)$ and update the priority of w in S — $O(\log|S|) \subseteq O(\log|V|)$ operations
            - $p(w) := v$ — $O(1)$ operations

In total: $O(|V|\log|V| + |A|\log|V|)$ elementary operations.

Dijkstra's Algorithm with Binary Heaps: Efficiency, II

In total: $O(|V|\log|V| + |A|\log|V|)$ elementary operations.
Under the natural assumption that $|A| \geq |V| = 1$ (no isolated vertices), we can write this as: $O(|A|\log|V|)$.

- For a very dense graph with $|A| \in \Theta(|V|^2)$, we would get a running time estimate of $O(|V|^2 \log|V|)$ – **this is worse than the old implementation without binary heaps!**
- However, already for slightly sparser graphs with $|A| \in O(|V|^2/\log|V|)$, the running time estimate is $O(|V|^2)$, which is the same as the old implementation.
- The sparser the graph, the better! In particular, for very sparse graphs with $|A| \in O(|V|)$, the running time estimate is $O(|V|\log|V|)$, which is **much better** than the old implementation.

A straight-forward implementation of Dijkstra's Algorithm with binary heaps easily solves problems examples such as with 70,000 vertices and 300,000 arcs in less than 10 seconds.