CS 270 - Homework 1

Walid Krichene (23265217)

(1.a) We have

$$
\begin{aligned}
T_1(n) &= 4T_1(n/3) + O(n^2) \\
&= \sum_{k=0}^{\log_3 n} 4^k O((n/3^k)^2) \\
&= O(n^2) \sum_{k=0}^{\log_3 n} (4/9)^k
\end{aligned}
$$

and since $4/9 < 1$, the sum is a $O(1)$, therefore

$$T_1(n) = O(n^2)$$

We have

$$
\begin{aligned}
T_2(n) &= 27T_2(n/3) + O(n^2) \\
&= \sum_{k=0}^{\log_3 n} 27^k O((n/3^k)^2) \\
&= O(n^2) \sum_{k=0}^{\log_3 n} (27/9)^k \\
&= O(n^2) \sum_{k=0}^{\log_3 n} 3^k
\end{aligned}
$$

and since $3 > 1$, the sum is a $O(3^{\log_3 n}) = O(n)$, therefore

$$T_2(n) = O(n^3)$$

(1.b) Decomposing each matrix into 4 submatrices, we can reduce the problem of multiplying $n \times n$ matrices to 32 multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices, plus addition operations (which is $O(\frac{n}{2}^2)$). Thus we can write

$$
\begin{aligned}
T(n) &= 32T(n/2) + O(n^2) \\
&= \sum_{k=0}^{\log_2 n} 32^k O((n/2^k)^2) \\
&= O(n^2) \sum_{k=0}^{\log_2 n} \frac{32}{4}^k \\
&= O(n^2) \sum_{k=0}^{\log_2 n} 8^k
\end{aligned}
$$

the sum is $O(8^{\log_2 n}) = O(n^{\log_2 8}) = O(n^3)$, therefore

$$T(n) = O(n^5)$$

(1.c) This is not true in general. Counter-example: consider the graph $G = (V, E)$ where $V = \{s, v, t\}$ (with source s and sink t) and $E = \{(s, v, 1), (v, t, 1)\}$ (capacities are both 1). Then a min-cut is simply given by $\{(s, v, 1)\}$, however, increasing the capacity will not increase the max-flow, which will still have value 1.

$$\textcircled{s} \xrightarrow{\;\;1\;\;} \textcircled{v} \xrightarrow{\;\;1\;\;} \textcircled{t}$$

In fact, we know that the value of the max-flow and value of min-cut coincide (the problems are dual), thus if the min-cut is not unique, increasing the capacity of any edge in the min-cut (thus the value of the cut), will not change the value of the min-cut for the new problem.

(1.d) The problem is

$$\begin{aligned} &\text{maximize} && f(x, y, w) \\ &\text{subject to} && x + y + w = 1 \end{aligned}$$

where $f(x, y, w) = \min(x+y, y+w, 3x+w)$ is the minimum of linear functions (thus concave). The problem is equivalent to the epigraph form

$$\begin{aligned} &\text{maximize} && t \\ &\text{subject to} && x + y + w = 1 \\ &&& f(x, y, w) \geq t \end{aligned}$$

which is then equivalent to

$$\begin{aligned} &\text{minimize} && t \\ &\text{subject to} && x + y + w = 1 \\ &&& x + y \geq t \\ &&& y + w \geq t \\ &&& 3x + w \geq t \end{aligned}$$

(1.e) Let x^* be a solution to the relaxed problem

$$\begin{aligned} &\text{minimize} && \sum_v x_v \\ &\text{subject to} && \forall (u, v) \in E, x_u + x_v \geq 1 \\ &&& \forall u \in V, x_u \geq 0 \end{aligned}$$

with optimal value $p^* = \sum_v x_v$. We construct a feasible point (for the original problem) $\bar{x}$ as follows:

$$\bar{x}_v = \begin{cases} 0 & \text{if } x_v^* = 0 \\ 1 & \text{if } x_v^* > 0 \end{cases}$$

(1.f) We have: the optimal value $C(e_i, t)$ is the maximum of the values corresponding to the following situations:

- either we do not schedule job i on machine 1, and the value is $C(e_{i-1}, t)$
- or we do schedule job i on machine 1, in which case the previous job on machine 1 must end no later than $f(s_i)$. In which case the value is $v_i + C(f(s_i), t)$

Similarly for machine 2.

Therefore we have

$$C(e_i, t) = \max(C(e_{i-1}, t), v_i + C(f(s_i), t))$$
$$C(t, e_i) = \max(C(t, e_{i-1}), v_i + C(t, f(s_i)))$$

(1.g) Running time: assuming there are n jobs (sorted by ascending end time, i.e. $e_1 \leq \cdots \leq e_n$), the solution is given by $C(e_n, e_n)$. We initialize the DP using $C(0,0) = 0$, then compute $C(e_i, e_j)$ for all $i, j \in \{1, \ldots, n\}$. This is done in $O(n^2)$ time.

(2.a) Assuming no two points are identical.

First, sort the points by increasing x (and break ties by sorting by increasing y).

$$i \geq j \Rightarrow (x_i > x_j) || (x_i = x_j \wedge y_i > y_j)$$

Then we observe that

- the first point $p_{\sigma(1)}$ dominates no other point
- if we fix $i \in \{1, \ldots, n\}$, then
  - for all $j > i$, p_i does not dominate p_j (by the sorting)
  - and for all $j < i$, we have $x_i \geq x_j$, thus p_i dominates p_j if and only if $y_i \geq y_j$

  therefore

$$(p_i \text{ dominates no other point}) \Leftrightarrow (y_i < \min_{j<i} y_j)$$

Therefore we obtain the following algorithm (let V be the set of points that dominate no other point)

Algorithm 1 Compute V

---

sort points according to the previous rule
initialize V to $\{p_1\}$
$min\_y := y_1$
for $i = 1$ to n **do**
    if $y_i < min\_y$ **then**
        add p_i to V
        $min\_y := y_i$
    end if
end for

Complexity: sorting is done in $O(n \log n)$, then computing the set S takes $O(n)$. Thus the total complexity is $O(n \log n)$.

(2.b) First, let us order the sequence $x_1, \dots, x_n$ in ascending order, and call the resulting sequence (a_i), and sort the sequence $y_1, \dots, y_n$ in descending order, and call the resulting sequence (b_j). So we have

$$a_1 \leq \cdots \leq a_n$$
$$b_1 \geq \cdots \geq b_n$$

Let, for all i, j in $\{1, \dots, n\}$,

$$S_{i,j} = S \cap \{(x, y) : x \leq a_i, y \geq b_j\}$$

$S_{i,j}$ is a subset of S, and we have in particular $S_{n,n} = S$.

Then let $U_{i,j}$ be the solution for the input set $S_{i,j}$ (i.e. $U_{i,j}$ is the largest subset of $S_{i,j}$ such that for each pair of points, no one dominates the other. If it is not unique, then any $U_{i,j}$ is any largest set). For convenience, let $C_{i,j}$ denote the cardinality of $U_{i,j}$.

We seek to compute $U_{n,n}$. We have

- $U_{1,1} = S_{1,1}$ since $S_{1,1}$ contains at most one point (note that it can be the empty set)
- to compute $U_{i,j}$, we simply take the union of $P_{i,j}$ and $U_{i,j}^{pre}$ where

  $$P_{i,j} = \begin{cases} \{(a_i, b_j)\} & \text{if } (a_i, b_j) \in S \\ \emptyset & \text{otherwise} \end{cases}$$

  and

  $$U_{i,j}^{pre} = \begin{cases} U_{i-1,j} & \text{if } c_{i-1,j} > c_{i,j-1} \\ U_{i-1,j} & \text{otherwise} \end{cases}$$

  (here we use the convention $U_{0,j} = U_{i,0} = \emptyset$) Justification: this second identity uses the following observations:

  1. if $p = (a_i, b_j)$ is a point in S, then any other point in $S_{i,j}$ is not dominated by p and does not dominate p, since $\forall (x, y) \in S_{i,j}$,

     $$x \geq a_i$$
     $$y \leq b_j$$

     therefore if (a_i, b_j) is a point, it is necessarily in the optimal set, and we can write

     $$U_{i,j} = P_{i,j} \cup U_{i,j}^{pre}$$

     where $U_{i,j}^{pre}$ is the solution for the set $S_{i-1,j} \cup S_{i,j-1}$ (noting that any point in $S_{i,j}$ is either in $P_{i,j}$ or in $S_{i-1,j} \cup S_{i,j-1}$).
  2. then we observe that $U_{i,j}^{pre}$ is necessarily either a subset of $U_{i-1,j}$, or a subset of $U_{i,j-1}$, otherwise, we would have two points $(x, y), (x', y') \in U_{i,j}^{pre}$ such that $(x, y) \in S_{i-1,j} \setminus S_{i,j-1}$ and $(x', y') \in S_{i,j-1} \setminus S_{i-1,j}$, but then

     $$x > a_{i-1} \geq x'$$
     $$y > b_{j-1} \geq y'$$

     so (x, y) would dominates (x', y') (see figure)

finally we obtain a simple dynamic program

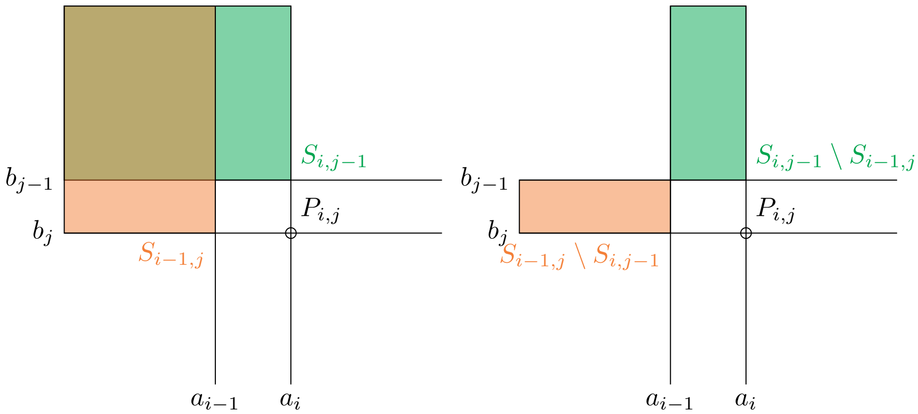

Figure 1: Illustration of the dynamic program

Algorithm 2 Compute U

sort $a_1 \leq \cdots \leq a_n$
sort $b_1 \geq \cdots \geq b_n$
initialize $U_{i,j} = \emptyset$ for all i, j in $\{0, \ldots, n\}$
initialize $C_{i,j} = 0$ for all i, j in $\{0, \ldots, n\}$
for $i = 1$ to n **do**
  for $j = 1$ to n **do**
    if (a_i, b_j) is a point **then**
      add it to $U_{i,j}$
    end if
    if $C_{i-1,j} > C_{i,j-1}$ **then**
      add $U_{i-1,j}$ to $U_{i,j}$ (union)
    else
      add $U_{i,j-1}$ to $U_{i,j}$ (union)
    end if
    $C_{i,j} := \text{card}(U_{i,j})$
  end for
end for

(3) Rule for clause vertex c, corresponding to the clause

$$\vee_{i \in I} s_i$$

where I is a subset of $\{1, \ldots, n\}$ and for each $i \in I$, s_i is either x_i or $\bar{x}_i$:

- for all $i \notin I$, add edge (c, w_i)
- for all $i \in I$, if $s_i = x_i$, add edge (c, v_i'), otherwise add edge (c, v_i)

justification: under this scheme, each clause vertex is connected to exactly n vertices, and among those, $\{w_i, i \in I^c\}$ will always be colored in distinct true colors. Thus we can color c in a true color if and only if one (at least) of the variable vertices connected to c is colored in "false". In other words, given an assignment and the corresponding coloring:

$$\begin{aligned} \text{The clause evaluates to true} &\Leftrightarrow \exists i \in I: \ s_i \text{ is true} \\ &\Leftrightarrow \exists i \in I: \ c \text{ is connected to a false color} \\ &\Leftrightarrow c \text{ can be colored in true} \end{aligned}$$

(4.a) For each $i \in \{1, \ldots, n\}$, count the mismatches between $s_1(i : i + m - 1)$ and s_2, if less than or equal to k, then i is a valid location. The total complexity is $O(nm)$ time.

Algorithm 3 find occurrence locations(s_1, s_2, k)

```
for i = 0 to n − m do
    mismatches := 0
    for j = 0 to m − 1 do
        if ¬(s2(i + j) = s1(j)) then
            mismatches + +
        end if
    end for
    if mismatches ≤ k then
        add i to the list of locations
    end if
end for
```

(4.b) Let the bit patterns s_1 and s_2 be given by two sequences of -1 and $+1$. Then consider the reversed sequence $\bar{s_2}$ given by

$$\forall i \in \{0, \ldots, n-1\},\ \bar{s_2}(i) = s_2(n-1-i)$$

and the convolution of the two signals is given by, for $i \in \{0, \ldots, n-1\}$

$$\begin{aligned} \forall s_1 * \bar{s}_2(i) &= \sum_{j=0}^{m-1} s_1(j)\bar{s_2}(i-j) \\ &= \sum_{j=0}^{m-1} s_1(j)s_2(n-1-i+j) \end{aligned}$$

we observe that for a fixed $i \in \{1, \ldots, n\}$, $s_1 * \bar{s}_2(n-1-i)$ is the sum $\sum_{j=0}^{m-1} s_1(j)s_2(i+j)$, and each term in this sum of m terms is

$$s_1(j)s_2(i+j) = \begin{cases} +1 & \text{if the two bits agree} \\ -1 & \text{otherwise} \end{cases}$$

therefore the number of mismatches is equal to $m - s_1 * \bar{s}_2(n-1-i)$, and we have

$$(s_1 \text{ occurs in } s_2 \text{ at the location } i) \Leftrightarrow (m - s_1 * \bar{s}_2(n-1-i) \leq k)$$

we obtain the following algorithm The complexity is $O(n \log n)$ (the convolution step is $O(n \log n)$, the

Algorithm 4 find occurrence locations(s_1, s_2, k)

$\bar{s}_2 := mirror(s_2)$
$f := conv(s_1, \bar{s}_2)$
for $i = 0$ to $n-1$ **do**
  if $m - f(n-1-i) \leq k$ **then**
    add i to the list of locations
  end if
end for

rest is linear)

(5.a) Each reversal $\rho(i,j)$ may only create or remove breakpoints at $(i-1,i)$ and $(j,j+1)$, therefore on reversal may decrease the number of breakpoints by at most 2. And since the target permutation (the identity) has 0 breakpoints, we have a lower bound on the minimal number of reversals

$$m(\pi) \geq b(\pi)/2$$

where $m(\pi)$ is the

(5.b) Idea of the algorithm:

- at each step, prioritize reversals that decrease the most the the number of breakpoints
- when there are ties, prioritize those that preserve maximal decreasing runs

---

Algorithm 5 find occurrence locations(s_1, s_2, k)

---

while permutation contains a breakpoint **do**
  choose reversal that maximizes decrease in breakpoints
  if tie **then**
    choose reversal that preserves a maximal run of decreasing sequence
  end if
end while

Claim: the number of reversals $k(\pi)$ required by this algorithm is

$$\begin{aligned} k(\pi) &\leq 2b(\pi) \\ &\leq 4\frac{b(\pi)}{2} \\ &\leq 4m(\pi) \end{aligned}$$

the bound $k(\pi) \leq 2b(\pi)$ follows from these two facts:

- if there is a decreasing run (of any length > 2) then there exists a reversal that decreases the number of breakpoints (can be seen by enumerating the possibilities)
- if there is no decreasing run, any reversal will create a decreasing run, in particular there exists a reversal that will create a decreasing run without increasing breakpoints.