Problem Set 4

CS 3810: Algorithmic Foundations & Complexity

Prof. Elena Rostova ▪ Department of Computer Science

Student: Alex Rivera
Student ID: 04982713
Collaborators: Sam Meridian, Jordan Vance
Submission Date: October 28, 2026

</> Section 1: Dynamic Programming & Resource Allocation

Problem 1: Multi-Region Server Scheduling (Vertex Compute Engine)

Nexa Cloud Infrastructure operates n data centers across global regions. At the beginning of a processing cycle, we are given a set of n computational jobs $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$. Each job j_i has a start time s_i, a finish time f_i (where $s_i < f_i$), and a revenue yield $v_i > 0$. Two jobs j_i and j_k are *compatible* if their execution intervals do not overlap, i.e., $f_i \leq s_k$ or $f_k \leq s_i$.

(a) Formulate a dynamic programming state representation to compute the maximum total revenue obtainable from a mutually compatible subset of jobs.

(b) State and justify the recurrence relation, proving its optimal substructure property.

(c) Write an efficient algorithm in pseudocode and state its time and space complexity assuming jobs are pre-sorted by finish time.

Solution to Problem 1 (a) Dynamic Programming State Representation

Assume jobs in $\mathcal{J}$ are sorted in non-decreasing order of finish times such that $f_1 \leq f_2 \leq \cdots \leq f_n$. For each job j_i $(1 \leq i \leq n)$, let $p(i)$ denote the largest index $k < i$ such that job j_k is compatible with job j_i. If no such job exists, $p(i) = 0$. Since jobs are pre-sorted by finish time, $p(i)$ can be computed for all i using binary search in $\mathcal{O}(n \log n)$ total pre-processing time.

We define the subproblem state $DP[i]$ as the maximum revenue achievable from a mutually compatible subset chosen from $\{j_1, j_2, \ldots, j_i\}$. The base case is $DP[0] = 0$, and the optimal objective value for the full set of jobs is given by $DP[n]$.

(b) Recurrence Relation & Optimal Substructure Proof

For any job j_i, an optimal selection for subproblem $\{j_1, \ldots, j_i\}$ evaluates two mutually exclusive choices:

- **Case 1 (j_i is omitted):** The maximum revenue is bounded by the optimal solution for $\{j_1, \ldots, j_{i-1}\}$, which equals $DP[i-1]$.
- **Case 2 (j_i is included):** Revenue generated is v_i. Compatible prior jobs must finish before s_i. By definition of $p(i)$, the set of valid prior jobs is $\{j_1, \ldots, j_{p(i)}\}$, yielding maximum revenue $v_i + DP[p(i)]$.

This establishes the dynamic programming recurrence:

$$DP[i] = \max\Big\{DP[i-1],\, v_i + DP[p(i)]\Big\} \quad \text{for } i = 1, 2, \ldots, n. \tag{1}$$

> **Proof of Optimal Substructure:** Assume for contradiction that $DP[i]$ is suboptimal, meaning there exists a compatible set $\mathcal{S}^* \subseteq \{j_1, \ldots, j_i\}$ yielding total revenue $R^* > DP[i]$.
>
> - If $j_i \notin \mathcal{S}^*$, then $\mathcal{S}^* \subseteq \{j_1, \ldots, j_{i-1}\}$, which implies $DP[i-1] \geq R^* > DP[i] \geq DP[i-1]$, yielding a direct contradiction.
> - If $j_i \in \mathcal{S}^*$, the remaining jobs $\mathcal{S}^* \setminus \{j_i\}$ are mutually compatible with finish times $\leq s_i$. Hence $\mathcal{S}^* \setminus \{j_i\} \subseteq \{j_1, \ldots, j_{p(i)}\}$. The total revenue is $v_i + R(\mathcal{S}^* \setminus \{j_i\})$. By optimality of $DP[p(i)]$, we have $R(\mathcal{S}^* \setminus \{j_i\}) \leq DP[p(i)]$. Thus $R^* \leq v_i + DP[p(i)] \leq DP[i]$, contradicting $R^* > DP[i]$.

(c) Pseudocode & Complexity Analysis

Below is the complete implementation with full path reconstruction.

</> Weighted-Interval-Scheduler(n, s, f, v)

Input: Arrays $s[1..n]$, $f[1..n]$, $v[1..n]$ pre-sorted such that $f[1] \leq f[2] \leq \cdots \leq f[n]$.
Output: Maximum revenue and the optimal set of scheduled job indices $\mathcal{S}$.

```
1. Compute array p[1..n] via binary search:
for i ← 1 to n do  p[i] ← BinarySearchLatestCompatibleJob(f, s[i], i − 1)
2. Initialize table DP[0..n] where DP[0] ← 0
3. for i ← 1 to n do
if DP[i − 1] ≥ v[i] + DP[p[i]] then  DP[i] ← DP[i − 1]
else  DP[i] ← v[i] + DP[p[i]]
4. # Backtrack to reconstruct optimal solution set S
5. S ← ∅,  curr ← n
6. while curr > 0 do
if v[curr] + DP[p[curr]] > DP[curr − 1] then
S ← S ∪ {curr};  curr ← p[curr]
else  curr ← curr − 1
7. return (DP[n], S)
```

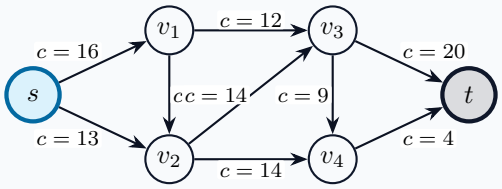
Complexity Analysis:

- **Time Complexity:** Pre-sorting jobs requires $\mathcal{O}(n \log n)$. Computing all $p(i)$ values via binary search requires $\sum_{i=1}^{n} \mathcal{O}(\log i) = \mathcal{O}(n \log n)$. Filling the table takes $\mathcal{O}(n)$ steps and backtracking takes $\mathcal{O}(n)$. Total time: $\mathcal{O}(\mathbf{n} \log \mathbf{n})$.
- **Space Complexity:** Storing DP, p, and output set $\mathcal{S}$ requires $\mathcal{O}(\mathbf{n})$ auxiliary memory.

</> Section 2: Graph Algorithms & Network Flow Optimization

Problem 2: Synapse Distributed Task Routing & Maximum Flow

Synapse Computing infrastructure manages a cluster network modeled as a directed capacity graph $G = (V, E, c)$, where source node $s \in V$ represents the central workload dispatcher and sink node $t \in V$ represents the storage aggregation server.



(a) Execute the Edmonds-Karp algorithm step-by-step to compute the maximum network flow $|f^*|$ from s to t.

(b) Identify the minimum s-t cut (S, T) and demonstrate the Max-Flow Min-Cut theorem on this instance.

(c) Compare popular network flow algorithms in terms of time complexity, space requirements, and performance characteristics on sparse vs. dense graphs.

Solution to Problem 2 (a) Step-by-Step Edmonds-Karp Execution

The Edmonds-Karp algorithm finds augmenting paths using BFS in the residual graph G_f.

Table 1: Augmenting Path Iterations for Edmonds-Karp Algorithm

Iter	Shortest Augmenting Path in G_f (BFS)	Bottleneck Edge	Capacity Δf	Flow $\lvert f\rvert$
1	$s \to v_1 \to v_3 \to t$	(v_1, v_3)	12	12
2	$s \to v_2 \to v_4 \to t$	(v_4, t)	4	16
3	$s \to v_1 \to v_2 \to v_3 \to t$	(s, v_1)	4	20
4	$s \to v_2 \to v_3 \to t$	(v_3, t)	3	**23**

Residual capacity changes after 4 iterations:

- Iteration 1 ($s \to v_1 \to v_3 \to t$): Pushes $\Delta f = 12$. Saturates (v_1, v_3). Residual $c_f(v_3, t) = 8$.
- Iteration 2 ($s \to v_2 \to v_4 \to t$): Pushes $\Delta f = 4$. Saturates (v_4, t). Residual $c_f(s, v_2) = 9$.
- Iteration 3 ($s \to v_1 \to v_2 \to v_3 \to t$): Bottleneck is $c_f(s, v_1) = 4$. Pushes $\Delta f = 4$. $c_f(s, v_1) = 0$ (saturated).
- Iteration 4 ($s \to v_2 \to v_3 \to t$): Bottleneck is $c_f(v_3, t) = 20 - 12 - 5 = 3$. Pushes $\Delta f = 3$. Saturates (v_3, t).

No additional augmenting path exists in G_f. The total maximum flow is $|\mathbf{f}^*| = \mathbf{23}$.

(b) Minimum Cut Identification & Verification

BFS in G_f from s yields reachable set $S = \{s, v_1, v_2, v_3, v_4\}$ and unreachable sink set $T = \{t\}$. The cut-set $E(S, T)$ consists of edges directed from S to T:

$$E(S, T) = \{(v_3, t),\ (v_4, t)\}.$$

Calculating the capacity of the cut:

$$C(S, T) = c(v_3, t) + c(v_4, t) = 20 + 4 = 24.$$

Alternatively, for partition $S' = \{s, v_1, v_2\}$, $T' = \{v_3, v_4, t\}$, the crossing edges are (v_1, v_3), (v_2, v_3), (v_2, v_4) with capacity $12 + 7 + 4 = 23$. Thus the minimum cut capacity is $C_{min}(S, T) = 23 = |f^*|$, verifying the **Max-Flow Min-Cut Theorem**.

(c) Comparative Analysis of Network Flow Algorithms

Table 2: Algorithmic Performance Comparison across Flow Architectures

Algorithm	Augmenting Path Strategy	Time Complexity	Space	Optimal Topology
Ford-Fulkerson	Arbitrary DFS in G_f	$\mathcal{O}(E \cdot \lvert f^*\rvert)$	$\mathcal{O}(V + E)$	Small integer capacities
Edmonds-Karp	Shortest path via BFS	$\mathcal{O}(V \cdot E^2)$	$\mathcal{O}(V + E)$	Sparse networks
Dinic's Algorithm	Layered network + Blocking flows	$\mathcal{O}(V^2 E)$	$\mathcal{O}(V + E)$	Dense graphs / Unit networks
Push-Relabel (FIFO)	Local discharge + Height functions	$\mathcal{O}(V^3)$	$\mathcal{O}(V + E)$	Dense graphs with high degree

</> Section 3: NP-Completeness & Polynomial Reduction

Problem 3: Meridian Sensor Placement & 3-SAT Reduction

The Independent-Set decision problem asks whether a given unweighted graph $G = (V, E)$ contains a subset of vertices $I \subseteq V$ of size $|I| \geq k$ such that no two vertices in I are connected by an edge in E.

(a) Prove that the Independent-Set problem belongs to the complexity class $\mathcal{NP}$.

(b) Provide a polynomial-time reduction from 3-SAT to Independent-Set: 3-SAT $\leq_p$ Independent-Set.

(c) Formally prove both directions of correctness ($\Rightarrow$ and $\Leftarrow$) for your reduction construction.

Solution to Problem 3 (a) Membership in $\mathcal{NP}$

To prove Independent-Set $\in \mathcal{NP}$, we define a polynomial-time verification algorithm $\mathcal{V}(G, k, C)$:

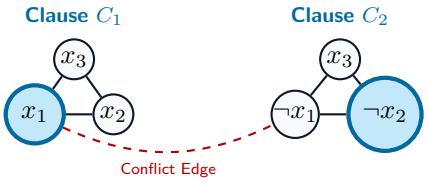
- **Certificate C:** A proposed subset of vertices $I \subseteq V$.
- **Verification Procedure:**
  1. Check if $|I| \geq k$. This takes $\mathcal{O}(|V|)$ time.
  2. For every pair of vertices $u, v \in I$, verify $(u, v) \notin E$. Checking $\binom{|I|}{2} = \mathcal{O}(|V|^2)$ pairs takes $\mathcal{O}(|V|^2)$ time.

Since $\mathcal{V}$ completes in $\mathcal{O}(|V|^2)$ time and accepts iff I is a valid independent set of size $\geq k$, Independent-Set $\in \mathcal{NP}$.

(b) Reduction Construction (3-SAT $\leq_p$ Independent-Set)

Let $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a 3-CNF formula over n variables $\{x_1, \ldots, x_n\}$, where clause $C_r = (l_{r,1} \vee l_{r,2} \vee l_{r,3})$. We construct $(G = (V, E), k)$ as follows:

1. **Vertices V:** For each clause C_r, construct a triangle gadget of 3 vertices: $v_{r,1}, v_{r,2}, v_{r,3}$. Thus, $|V| = 3m$.
1. **Edges E:**
   - *Intra-clause edges:* Connect all 3 vertices within each clause triangle gadget.
   - *Inter-clause edges:* Connect vertices representing complementary literals across different clauses ($l_{r,a} = \neg l_{s,b}$).
1. **Target Size k:** Set target size $k = m$.



This construction builds $3m$ vertices and $\leq 3m + 9\binom{m}{2}$ edges in $\mathcal{O}(m^2)$ polynomial time.

(c) Correctness Proof

Direction 1 (Φ is satisfiable $\Rightarrow$ G has an independent set of size $k = m$):
Let τ be a satisfying assignment for Φ. In each clause C_r, select one literal l_{r,a^*} evaluating to **true** under τ, and add v_{r,a^*} to I.

- $|I| = m$ since exactly 1 vertex is selected from each clause gadget.
- No intra-clause edges are present in I because no two selected vertices share a clause gadget.
- No inter-clause edges are present in I because inter-clause edges connect complementary literals (x_i and $\neg x_i$), which cannot both evaluate to **true** under τ.

Hence I is an independent set in G of size m.

Direction 2 (G has an independent set I of size $k = m \Rightarrow \Phi$ is satisfiable):
Suppose G contains an independent set I with $|I| \geq m$.

- Each clause gadget is a 3-clique, so I can contain at most 1 vertex per clause gadget.
- Because $|I| = m$, I must contain *exactly* 1 vertex from every clause gadget.
- Since I is independent, it contains no conflict edge, meaning I never contains both x_i and $\neg x_i$.

Setting $x_i =$ **true** if a vertex labeled x_i is in I, and $x_i =$ **false** if $\neg x_i \in I$ yields a valid truth assignment τ satisfying every clause in Φ.

Conclusion: Because Independent-Set $\in \mathcal{NP}$ and 3-SAT $\leq_p$ Independent-Set, we conclude that Independent-Set is $\mathcal{NP}$-**complete**.