1 Introduction

In this Lecture we will introduce the Local Ratio Technique. We will show applications of this method to the following problems:

- Vertex cover
- Interval scheduling
- Interval scheduling with deadlines
- Interval scheduling with intervals of various widths.

2 Vertex Cover

Introducing the problem:
Input: Graph $G = (V,E)$, weight function $w : V \rightarrow R^+$
Feasible solution: $S \subseteq V \text{ s.t } \forall e = (v,u) \in E. u \in S \text{ or } v \in S$
Goal Function: Minimize $\sum_{v \in S} w(v)$.

Claim 1: The value of the solution is 0 $\iff$ the group of all the vertexes with weight 0 is a valid cover $\iff \forall e = (u, v) \in E u \in S \text{ or } v \in S$
Proof: follows by definition.

2.1 First Solution:
Given input: $G = (V,E)$, $w(v) \rightarrow R^+$, choose an edge $e=(u,v) \in E$ and subtract $X = \min \{w(u), w(v)\}$ from $w(v)$ and $w(u)$. The result is graph $G$ with a new weight function $w'$.

Figure 1: An example of $G$ with weights $w$ & $w'$:

Notice that:
(1) \( \text{OPT}(G,w') \leq \text{OPT}(G,w) - X \)
(2) \( A(G,w) \leq A(G,w') + 2X \)

Explanations:
(1) if \( S \) is a feasible solution to \((G,w)\) with value \( V \), it will also be a feasible solution to \((G,w')\) with value at most \( V - X \).
(2) if \( S \) is a feasible solution to \((G,w')\) with value \( V \), it will also be a feasible solution to \((G,w)\) with value at most \( V + 2X \) (if both \( v \) and \( u \) are in \( S \), we add \( 2X \)).

Claim 2: We can achieve a 2-approximation algorithm by recursion on \((G,w')\):
The Algorithm: choose an edge \( e \in E \) and generate \((G,w')\). Continue until the weight of at least one of the vertexes of each edge is 0.

Proof: By induction: (on the number of edges \((u,v)\) s.t \( w(u) > 0 \) and \( w(v) > 0 \))
Base case: At least one of the vertexes of each edge is 0 \( \rightarrow \) the value of the solution is 0 (we will take all the vertexes with weight 0). The solution is optimal, and therefore 2-approximate.
Induction Step: \( A(G,w') \leq A(G,w') + 2x \leq \text{induction} 2\text{OPT}(G,w') + 2x = 2(\text{OPT}(G,w') + x) \leq (1) 2\text{OPT}(G,w) \)

Figure 2: An Example of the recursive algorithm:

2.2 An alternative approach:

We divided the weight function \( w \) into 2 functions: \( w_1:V \rightarrow R^+ \) and \( w_2:V \rightarrow R^+ \) s.t \( w = w_1 + w_2 \).

Figure 3: The division of \( W \) in our example:

The Local Ratio Theorem: If solution \( S \) is \( \alpha \)-approximate for \( (G,w_1) \) and \( (G,w_2) \), \( S \) is also \( \alpha - \)approximate for \((G,w)\)
Proof: \( w(S) = w_1(S) + w_2(S) \leq \alpha OPT(G, w_1) + \alpha OPT(G, w_2) = \alpha (OPT(G, w_1) + OPT(G, w_2)) \leq \alpha OPT(G, w + w) = \alpha OPT(G, w) \)

**The local Ratio Theorem applied to the Vertex Cover:**

The algorithm: If a zero-cost solution can be found, return one. Otherwise, decompose \( w \) into \( w_1 \) & \( w_2 \), and solve the problem recursively, using \( w_1 \) as the weight function in the recursive call.

**Claim 3:** The algorithm is 2-approximate.

**Proof:** By Induction: (on the number of edges \( (u,v) \) s.t \( w(u) > 0 \) and \( w(v) > 0 \))

Base case: The algorithm returns a VC of zero cost, which is optimal.

Inductive step: consider the solution returned by the recursive call. In \( (G, w_1) \) there is one edge \( (u,v) \) less s.t \( w(u) > 0 \) and \( w(v) > 0 \), by the inductive hypothesis it is 2-approximate with respect to \( w_1 \). We claim that it is also 2-approximate with respect to \( w_2 \). In fact, every feasible solution is 2-approximate with respect to \( w_2 \): \( OPT(G, w_2) \geq x \) and \( Alg(G, w_2) = \sum_{v \in S} w_2(v) \leq \sum_{v \in V} w_2(v) = 2x \). \( (G, w_1) \) and \( (G, w_2) \) is 2-approximate \( \rightarrow (G, w) \) is 2-approximate.

### 3 Interval Scheduling

Introducing the problem:

**Input:** A set of intervals: \( I_i = (s_i, t_i, v_i) \) \( s_i \leq t_i \) \( v_i \) - the value of interval \( i \)

**Feasible solution:** subset \( S \) of non-conflicting intervals.

**Goal Function:** maximize the sum of values of \( S \).

**Figure 4:** Example of Interval scheduling problem:

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

3.1 Simple Case: \( \forall i. v_i = 1 \):

The Goal is to maximize the number of intervals in \( S \).

The Algorithm: Choose the interval that ends first, and remove all of its conflicting intervals. Repeat until there are no more intervals.

**Claim 4:** The Algorithm gives an optimal solution.

**Proof:** The chosen interval is better than any other interval, because it can be replaced by any of the other intervals we removed.

3.2 The General Case \( \forall i. v_i \geq 0 \):

3.2.1 Solution using dynamic programming:

Sort the intervals by the ending time \( (t_i) : I_1 \leq I_2 \ldots \leq I_n \)

**definition:** \( F(I) = \) the maximum gain from intervals ending until the ending if the \( i \)-th interval ; \( F(0) = 0 \).
**Definition:** \( J_i = \) the interval with maximum index that ends before the beginning of interval \( i \).

\[ F(I) = \{ F(i-1), v(i) + F(j_i) \} \]

### 3.2.2 Solution using Local Ratio technique:

1. If there are intervals such that \( v_i < 0 \), remove them. If the input is \( \phi \), return \( \phi \).

2. The weight function \( w_1 \): Select the interval that ends first - \( I_1 \) with value \( v \) and remove \( v \) from all of \( I_1 \)’s conflicting intervals. (The weight function \( w_2 = w - w_1 \). Remove \( I_1 \).

3. Solve recursively.

4. Take the recursive solution and add \( I_1 \) to it if possible (= if interval \( I_1 \) doesn’t conflict with the other intervals in the solution).

**Claim 5:** The algorithm returns an optimal solution.

**Proof:** By induction (on the number of intervals):

Base case: If the input is \( \phi \), return \( \phi \), optimal solution.

Inductive step: First, notice that the all the intervals with value > \( 0 \) are conflicting with each other, and \( \text{OPT} = v \). Consider the solution \( S \) returned by the recursive call. In \( w_1 \) there is at least one interval less than in \( w \). By the inductive hypothesis \( S \) is optimal with respect to \( w_1 \).

If \( S \) contains an interval conflicting with \( I_1 \): There is an interval conflicting with \( I_1 \) in the solution \( \rightarrow \) the value of the solution in Alg(\( w_2 \)) = \( v \) \( \rightarrow \) optimal.

If \( S \) doesn’t contain an interval conflicting with \( I_1 \) we add it to the solution: Alg(\( w_2 \)) = \( v \) \( \rightarrow \) optimal. \( S \) is optimal with respect to \( w_1 \) and \( w_2 \) \( \rightarrow \) \( S \) is optimal with respect to \( w = w_1 + w_2 \).

### 4 Interval Scheduling with deadlines

Introducing the problem:

Input: A set of intervals: \( I_i = (s_i, t_i, d_i, v_i) \): interval \( i \) can be placed in the range \([s_i, t_i]\), the interval’s length is \( d_i \), and its value - \( v_i \).

Feasible solution: Subset \( S \) of non-conflicting intervals, and the starting location for each selected interval in its range.

Goal Function: Maximize the sum of values of \( S \).

**Remark:** The problem is NP-Complete (can be proved by reduction to Bin Packing problem).

Assumption: \( s_i, t_i, d_i \) are Integer.

The technique: Let’s open each interval to a set of all of it’s possible locations within the range \([s_i, t_i]\), and apply the algorithm described in section 3.2.2.

The Algorithm:

1. If there are intervals such that \( v_i < 0 \), remove them. If the input is \( \phi \), return \( \phi \).
2. The weight function $w_1$: Select the interval that that ends first - $I_1$ with value $v$ and remove $v$ from: (1) all of $I_1$'s conflicting intervals. (2) all the copies of $I_1$. (the weight function $w_2 = w - w_1$). Remove $I_1$.

3. Solve recursively.

4. Take the recursive solution and add $I_1$ to it if possible (= if interval $I_1$ doesn’t conflict with the other intervals in the solution, and there isn’t any copy of $I_1$ in the solution).

**Claim 6:** The algorithm is 2-approximate.

**Claim 7:** Optimal solution for $w_2 \leq 2v$.

**Proof (of Claim 7):** We take $v$ from all the intervals conflicting with $I_1$, and $v$ from a copy of $I_1$. We get $2v$.

**Claim 8:** The value of every maximal solution with respect to $I_1 \geq v$.

**Proof (of Claim 8):** By definition: If we take a copy of $I_1$ we get $v$. If we take a copy of an interval conflicting with $I_1$ we get $v$. Otherwise, we can add $I_1$ to the solution - we get $v$.

**Corollary 1:** From claims 7 and 8 we get that any feasible solution to $w_2$ is 2-approximate.

**Proof (of Claim 6):** By induction (on the number of intervals):

Base case: If the input is $\phi$ - return $\phi$ - optimal solution (and 2-approximate).

Inductive step: Consider the solution $S$ returned by the recursive call. In $w_1$ there is at least one interval less than in $w$. By the inductive hypothesis $S$ is optimal with respect to $w_1$. We have also shown (in corollary 1) that any feasible solution to $w_2$ is 2-approximate. $S$ is optimal with respect to $w_1$ and $w_2 \rightarrow S$ is optimal with respect to $w = w_1 + w_2$.

5 Interval Scheduling with intervals of various widths:

Introducing the problem:

Input : A set of intervals: $I_i = (s_i, t_i, w_i, v_i)$: $s_i$ - the start of the interval, $t_i$ - the end of the interval, $w_i$ - the interval’s width, $v_i$ - the interval’s value.

Feasible solution: Subset $S$ of intervals, s.t at any given time, the sum of the widths of the intervals is at most 1.

Goal Function: Maximize the sum of values of $S$.

**Figure 5:** An example of Interval Scheduling with intervals of various widths:

The Algorithm:

**Phase 1:** Separate the input into 2 groups: $I_1 = \{i | w_i > \frac{1}{2}\}; I_2 = \{i | w_i \leq \frac{1}{2}\}$

**Phase 2:** Solve each group separately:
Solution to I₁ : since \( w_i > \frac{1}{2} \), at each given moment, there can be placed at most 1 interval. This is the regular Interval Scheduling problem- we have shown how to solve this problem optimally.

Solution to I₂ :

1. If there is an interval s.t \( v_i < 0 \) - remove item. If the input is \( \phi - return \phi \).

2. The weight function \( w_1 \): Select the interval that that ends first - \( I_1 \) with value \( v \) and weight \( w \), and remove \( \frac{w}{1-w} \) from all of \( I'_1 \) conflicting intervals. \( (w_i \) is the weight of the conflicting interval). The weight function \( w_2 = w - w_1 \). Remove \( I_1 \).

3. Solve recursively.

4. Take the recursive solution and add \( I_1 \) to it if possible (= if interval \( I_1 \) adding interval \( I_1 \) doesn't result in a time \( t \) s.t the sum of the weights at that time is over 1).

Claim 9: Optimal solution for \( I_2, w_2 \leq 2v \).

Proof: \( OPT \leq max\{v \sum_{\frac{w}{1-w}}, v + v \sum_{\frac{w}{1-w}}\} \leq max\{v \frac{1}{1-w}, v + v \frac{1}{1-w}\} = max\{2v, 2v\} = 2v \). Explanation: The first option - \( OPT \leq v \sum_{\frac{w}{1-w}} \) represents the case where we didn’t add \( I_1 \) to the solution. In this case \( \sum w_i \leq 1 \) since it’s a valid solution, and since \( w_i > \frac{1}{2} \) we get \( OPT \leq 2v \). The second option(we add \( I_1 \) to the solution): \( OPT \leq v + v \sum_{\frac{w}{1-w}} - v \) for \( I_1 \) and \( v \sum_{\frac{w}{1-w}} \) for all of \( I'_1 \)'s conflicting intervals. We can add \( I_1 \), so \( \sum w_i + w \leq 1 \rightarrow \sum w_i \leq 1 - w \). We get \( OPT \leq 2v \).

Claim 10: the value of every maximal solution with respect to \( I_1, w_2 \geq v \).

Proof: \( \text{Alg} \geq min\{v, v \sum_{\frac{w}{1-w}}\} \geq min\{v, v \frac{1}{1-w}\} = min\{v, v\} = v \).

Explanation: The first option - \( \text{Alg} \geq v \) represents the case where we add \( I_1 \) to the solution-the solution value is at least \( v \) . The second option(we can’t add \( I_1 \) to the solution):

\( OPT \geq v \sum_{\frac{w}{1-w}} \). We can’t add \( I_1 \), so \( \sum w_i + w \geq 1 \rightarrow \sum w_i > 1 - w \). We get \( \text{Alg} \geq v \).

Corollary 2: From claims 9 and 10 we get that any feasible solution to \( w_2 \) is 2-approximate.

Claim 11: The solution to \( I_2 \) is 2-approximate.

Proof: The proof of claim 6 (the algorithm for interval scheduling with deadlines) also applies here.

Phase 3: We solve the problem for \( I_1 \) and \( I_2 \) and take the best solution.

Claim 12: The algorithm is 3-approximate.

Proof: \( OPT \leq OPT(I_1) + OPT(I_2) \). If \( OPT(I_1) \geq \frac{1}{3}OPT \) : \( \text{Alg}(I_1) \) returns an optimal solution, so we get : \( \text{Alg}(I_1) = OPT(I_1) \geq \frac{1}{3}OPT \). We get a 3-approximation. Otherwise, \( OPT(I_2) \geq \frac{2}{3}OPT \). We got a 2-approximation to \( I_2 \), so we get \( \text{Alg}(I_2) \geq \frac{1}{2}OPT(I_2) \geq \frac{1}{2} \cdot \frac{2}{3}OPT = \frac{1}{3}OPT \). In both cases we get 3-approximation.