Resource Allocation for Covering Time Varying Demands

Venkat Chakravarthy, Amit Kumar, Sambuddha Roy[1], Yogish Sabharwal

[1] sambuddha@in.ibm.com
IBM Research – India

ESA 2011

Typical Scenarios in Call Centers

- Consider any call center scenario. There is a demand profile of the forecasted demand over an extended **timeline**, varying across the different *timeslots*.
- Different employees have some prescribed shift times, with the corresponding shift lengths.
- The forecasted demand needs to be fulfilled by hiring enough employees of the different resource types. The objective is to pick the minimum number of employees of the various resource types so as to fulfill the demand.

Typical Scenarios in Call Centers

- However, oftentimes it is the case that not all the timeslots in the timeline need to be fulfilled, but there are SLA requirements such as: “satisfy 90% of the timeslots".
- Let’s call this the SLA problem.
- Other scenarios where such problems arise naturally: sensor networks, cloud computing, energy management . . .

Typical Scenarios in Call Centers

- However, oftentimes it is the case that not all the timeslots in the timeline need to be fulfilled, but there are SLA requirements such as: “satisfy 90% of the timeslots".
- Let’s call this the SLA problem.
- Other scenarios where such problems arise naturally: sensor networks, cloud computing, energy management . . .

Abstraction of the SLA problem

- In this problem, we are given a timeline $\mathcal{T}$ consisting of timeslots $1, \cdots, T$ (where $T = |\mathcal{T}|$). Every timeslot has a demand d_t.
- We are also given a collection $\mathcal{R}$ of *resource intervals* (corresponding to the employees of various shifts). Each resource type has a cost per unit of the resource.
- Thus, a resource interval i is equipped with a “height" h_i (the *proficiency* of the resource) cost c_i, starting time s_i and ending time e_i.

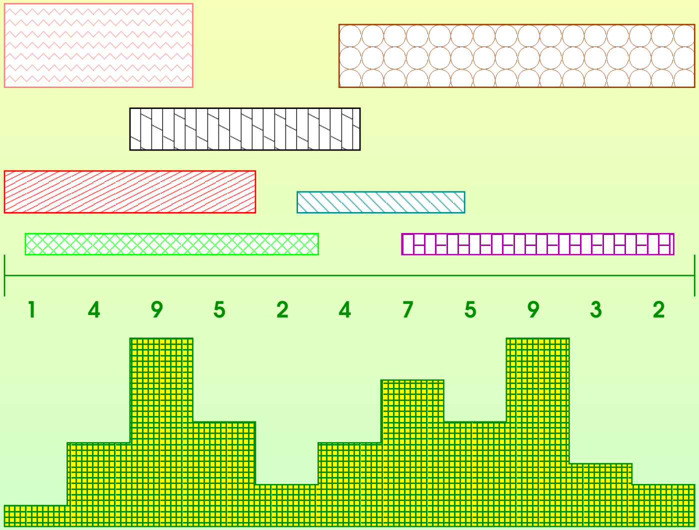
The SLA requirement

- We are also given a number k ($1 \leq k \leq T$), that corresponds to the SLA requirement: the number of timeslots that need to be fulfilled.
- The **objective** is to select a minimum cost multiset of resource intervals from $\mathcal{R}$ so that at least k of the timeslots in $\mathcal{T}$ are satisfied.
- Call this problem the **Partial Interval Covering Problem** (it is $\mathbb{NP}$-hard – Minimum Knapsack Cover is a special case).

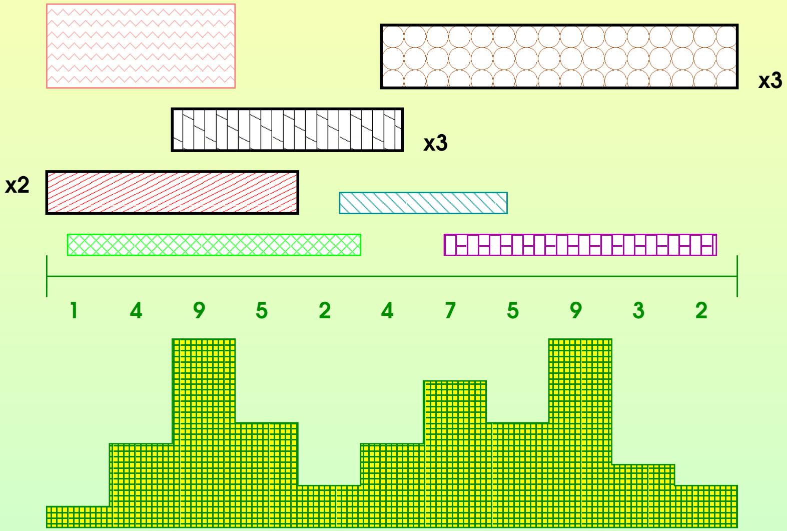
The SLA requirement

- We are also given a number k ($1 \leq k \leq T$), that corresponds to the SLA requirement: the number of timeslots that need to be fulfilled.
- The **objective** is to select a minimum cost multiset of resource intervals from $\mathcal{R}$ so that at least k of the timeslots in $\mathcal{T}$ are satisfied.
- Call this problem the **Partial Interval Covering Problem** (it is NP-hard – Minimum Knapsack Cover is a special case).

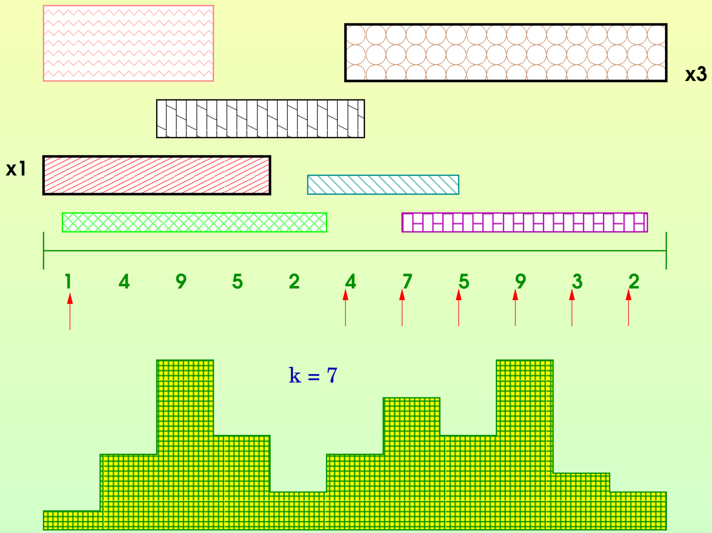
Illustration



Illustration



Illustration



Ambient space of our problem

- The **Partial Interval Covering Problem** is a **covering** problem.
- Instances of covering problems abound: Set Cover, Vertex Cover, Minimum Spanning Tree, Steiner Network etc.
- The covering variant corresponding to the SLA problem is a "partial" covering problem.

History of Partial Covering problems

- 2-factor approximations for the Partial Vertex Cover problem [Bar-Yehuda99, Gandhi-Khuller-Srinivasan01, Mestre07].
- The k-Median problem has constant factor approximations; methods are based on Lagrangian Relaxations [cf. JainVazirani99]
- 2-factor approximations for the k-MST problem [Garg05]
- etc. . .

History of Partial Covering problems

- 2-factor approximations for the Partial Vertex Cover problem [Bar-Yehuda99, Gandhi-Khuller-Srinivasan01, Mestre07].
- The *k*-Median problem has constant factor approximations; methods are based on Lagrangian Relaxations [cf. JainVazirani99]
- 2-factor approximations for the *k*-MST problem [Garg05]
- etc. . .

Main Result

- The **Partial Interval Covering Problem** has a constant factor approximation. Specifically,

Main Result

There is a polynomial time 16-factor approximation for the Partial Interval Covering problem.

Yet another partial covering problem. . .?

- Most of the prior partial covering literature concerns situations with unit demands.
- Unit demands basically mean, a **single** resource suffices to cover a demand.
- This work deals with *arbitrary* demands - in such a case, an optimal solution may have various different resources covering a single demand.

Yet another partial covering problem. . .?

- Most of the prior partial covering literature concerns situations with unit demands.
- Unit demands basically mean, a **single** resource suffices to cover a demand.
- This work deals with *arbitrary* demands - in such a case, an optimal solution may have various different resources covering a single demand.

The case of arbitrary demands

- Can we somehow reduce the problem with arbitrary demands to one with unit demands?
- Note that for our problem, we are allowed multiple copies of each resource interval.

The case of arbitrary demands

- Arbitrary demands can “start looking" like unit demands, if for instance, a demand is covered by a **single** resource. But this is too much to expect – a demand may be too high for any single resource.
- But it is always possible for a demand to be covered by a single **type** of resource (recall that we are allowed copies)!

The case of arbitrary demands

- Arbitrary demands can “start looking" like unit demands, if for instance, a demand is covered by a **single** resource. But this is too much to expect – a demand may be too high for any single resource.
- But it is always possible for a demand to be covered by a single **type** of resource (recall that we are allowed copies)!

With this motivation. . .

- We make the following definition:

nice solutions

A solution $\mathcal{S}$ to the Partial Interval Covering problem is said to be **nice** if for any timeslot t (with demand d_t) that is fulfilled in the solution $\mathcal{S}$, we can associate a single resource interval i in $\mathcal{S}$ whose copies in $\mathcal{S}$ cover the demand d_t.

- I.e, for any timeslot that we decide to *cover* in our solution, we use intervals of a single type (i.e. we use the required number of copies of a single interval).

Illustration for **nice** solutions

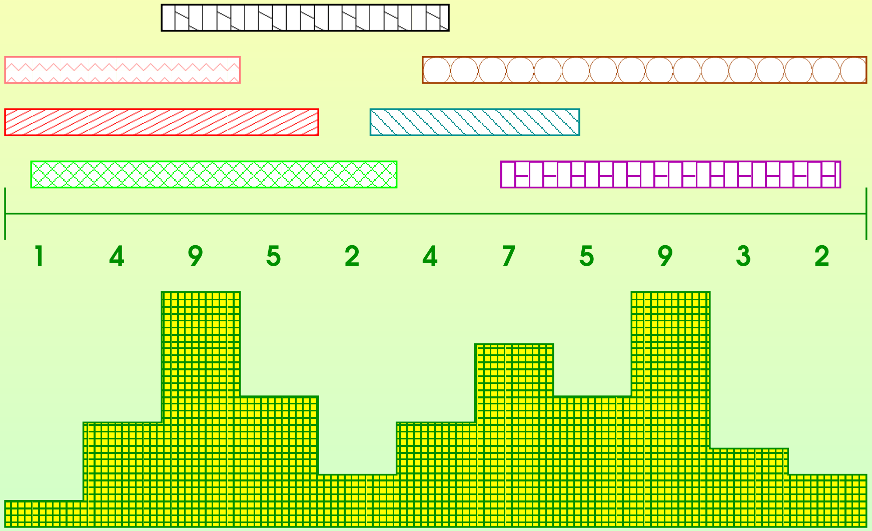


Illustration for **nice** solutions

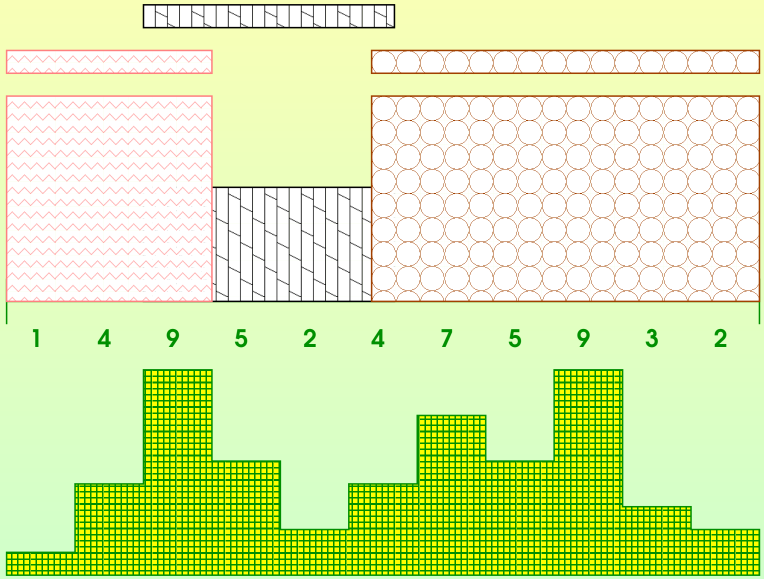
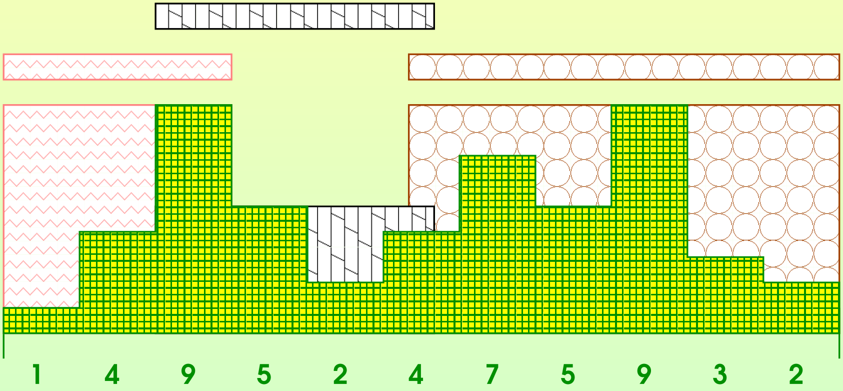


Illustration for **nice** solutions



About **nice** solutions

- The basic point being that for **nice** solutions, one can think of keeping multiple copies of the various resources in our problem instance; and the arbitrary demands now “look" like unit demands.
- This leads us to believe that given a Partial Interval Covering problem instance, finding an **optimal** **nice** solution should be relatively easy.
- This is in fact true; the Jain-Vazirani framework yields a constant factor approximation to this problem; more is true: Finding the optimal **nice** solution can be done in polynomial time! (by Dynamic Programming)

About **nice** solutions

- The basic point being that for **nice** solutions, one can think of keeping multiple copies of the various resources in our problem instance; and the arbitrary demands now “look" like unit demands.
- This leads us to believe that given a Partial Interval Covering problem instance, finding an **optimal** **nice** solution should be relatively easy.
- This is in fact true; the Jain-Vazirani framework yields a constant factor approximation to this problem; more is true: Finding the optimal **nice** solution can be done in polynomial time! (by Dynamic Programming)

About **nice** solutions

- The basic point being that for **nice** solutions, one can think of keeping multiple copies of the various resources in our problem instance; and the arbitrary demands now “look" like unit demands.
- This leads us to believe that given a Partial Interval Covering problem instance, finding an **optimal** **nice** solution should be relatively easy.
- This is in fact true; the Jain-Vazirani framework yields a constant factor approximation to this problem; more is true: Finding the optimal **nice** solution can be done in polynomial time! (by Dynamic Programming)

About nice solutions

- The basic point being that for **nice** solutions, one can think of keeping multiple copies of the various resources in our problem instance; and the arbitrary demands now “look" like unit demands.
- This leads us to believe that given a Partial Interval Covering problem instance, finding an **optimal nice** solution should be relatively easy.
- This is in fact true; the Jain-Vazirani framework yields a constant factor approximation to this problem; more is true: Finding the optimal **nice** solution can be done in polynomial time! (by Dynamic Programming)

What good are **nice** solutions?

- Do **nice** solutions get us any closer to our original goal of getting optimal solutions to the Partial Interval Covering problem?
- We show:

Main Lemma

Given any instance $\mathcal{I}$ of the Partial Interval Covering problem, there always exists a **nice** solution $\mathcal{S}$ with cost at most 16 times that of OPT.

- To paraphrase: there exist near-optimal **nice** solutions.

What good are **nice** solutions?

- Do **nice** solutions get us any closer to our original goal of getting optimal solutions to the Partial Interval Covering problem?
- We show:

Main Lemma

Given any instance $\mathcal{I}$ of the Partial Interval Covering problem, there always exists a **nice** solution $\mathcal{S}$ with cost at most 16 times that of OPT.

- To paraphrase: there exist near-optimal **nice** solutions.

Overall Algorithm

- The final algorithm would be then as follows:
- Find the optimal **nice** solution to the partial cover problem.
- Output this solution.

Existence of near-optimal nice solutions

- It suffices to show the result for the full cover version (Why?).
- In fact, we will only consider the special case where all the resource heights h_i equal 1. In this setting, the claim becomes:

Claim

Given an instance $\mathcal{I}$ of the full cover problem, there exists a **nice** solution $\mathcal{S}'$ with cost at most 8 times that of OPT (recall that OPT need not be **nice**).

- This proceeds via the primal-dual paradigm.

Existence of near-optimal **nice** solutions

- It suffices to show the result for the full cover version (Why?).
- In fact, we will only consider the special case where all the resource heights h_i equal 1. In this setting, the claim becomes:

Claim

Given an instance $\mathcal{I}$ of the full cover problem, there exists a **nice** solution $\mathcal{S}'$ with cost at most 8 times that of OPT (recall that OPT need not be **nice**).

- This proceeds via the primal-dual paradigm.

Existence of near-optimal **nice** solutions

- A (relatively) standard primal-dual algorithm is a 2-approximation for the full cover problem; this algorithm consists of a forward and a reverse phase.
- We will borrow the forward phase from that primal-dual algorithm, but the solutions produced by this algorithm (i.e. after the reverse phase) are not **nice**.
- We modify the reverse phase in order to produce near-optimal **nice** solutions. This also results in increasing the cost of the near-optimal **nice** solutions.
- Given time constraints, we will elaborate mostly using pictures.

Existence of near-optimal **nice** solutions

- A (relatively) standard primal-dual algorithm is a 2-approximation for the full cover problem; this algorithm consists of a forward and a reverse phase.
- We will borrow the forward phase from that primal-dual algorithm, but the solutions produced by this algorithm (i.e. after the reverse phase) are not **nice**.
- We modify the reverse phase in order to produce near-optimal **nice** solutions. This also results in increasing the cost of the near-optimal **nice** solutions.
- Given time constraints, we will elaborate mostly using pictures.

Existence of near-optimal **nice** solutions

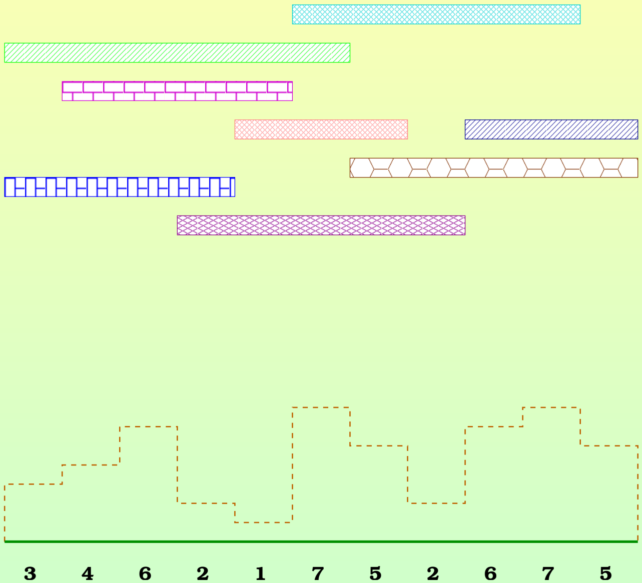
- A (relatively) standard primal-dual algorithm is a 2-approximation for the full cover problem; this algorithm consists of a forward and a reverse phase.
- We will borrow the forward phase from that primal-dual algorithm, but the solutions produced by this algorithm (i.e. after the reverse phase) are not **nice**.
- We modify the reverse phase in order to produce near-optimal **nice** solutions. This also results in increasing the cost of the near-optimal **nice** solutions.
- Given time constraints, we will elaborate mostly using pictures.

Forward Phase

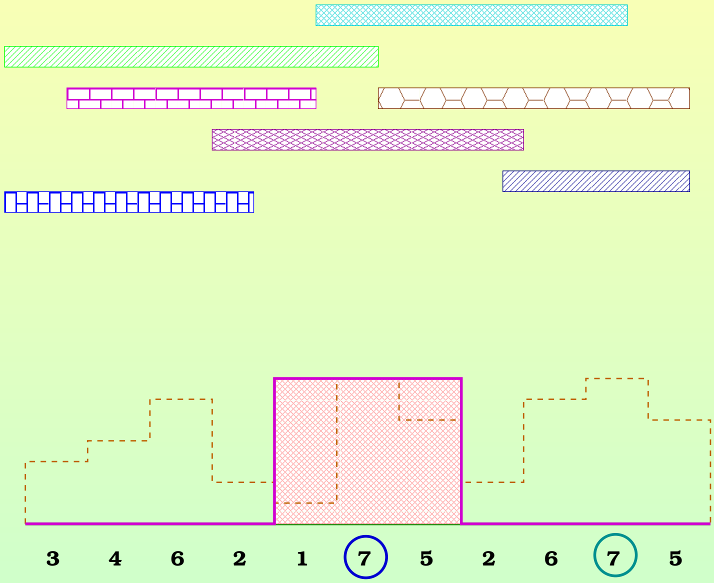
Motif

Go after the timeslot with the highest demand, and satisfy it with multiple copies of the (cheapest) resource interval.

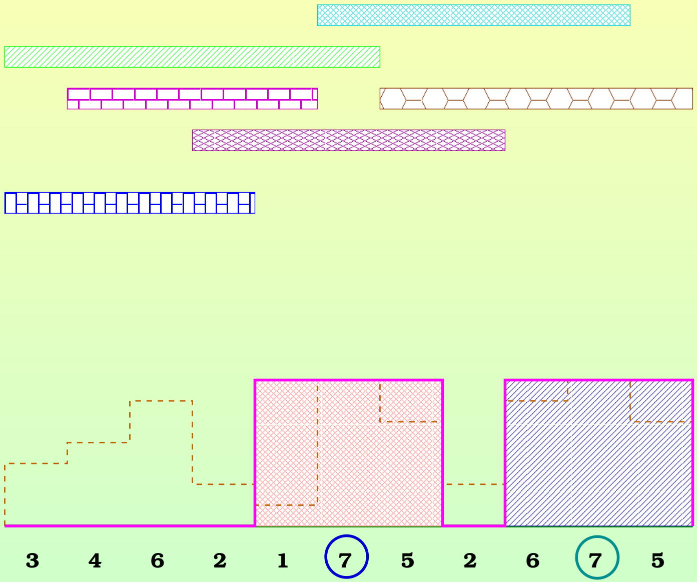
The Forward Phase



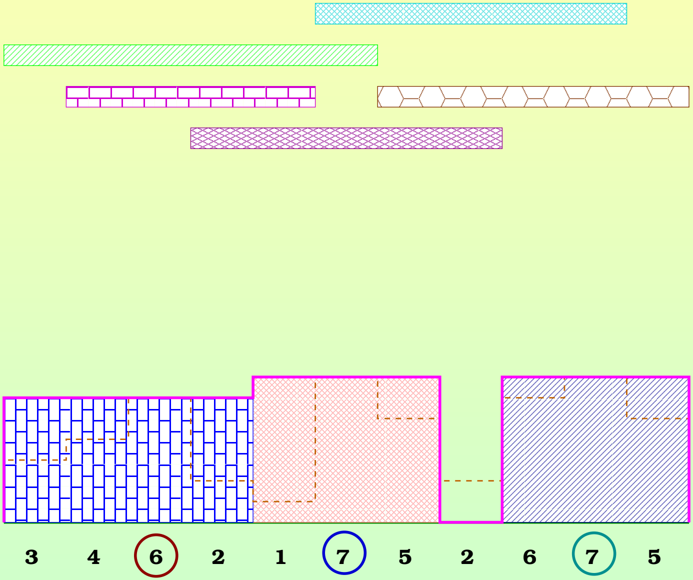
The Forward Phase



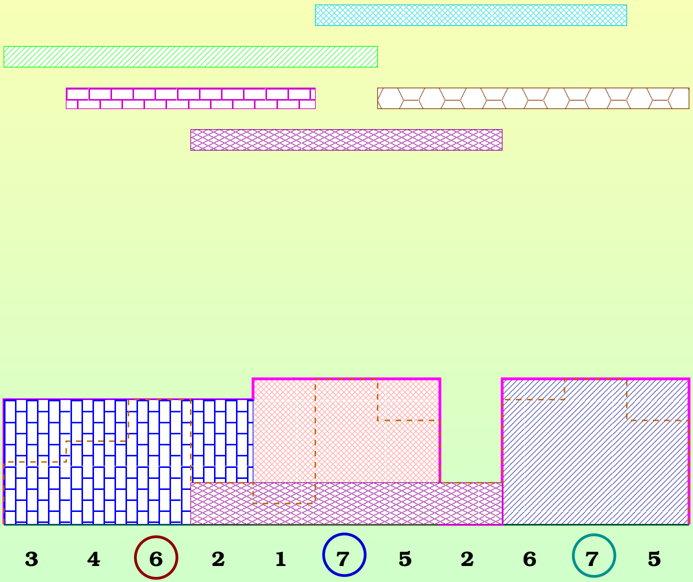
The Forward Phase



The Forward Phase



The Forward Phase



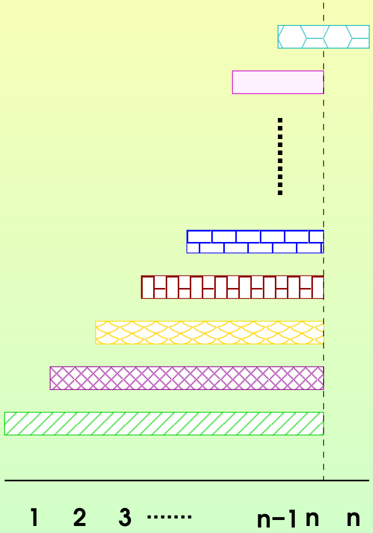
Is that good enough?

- The forward phase is *myopic*: at every point we select the **best** choice available “locally".
- Let’s run through an example to see how the forward phase can perform significantly worse than the actual OPT.

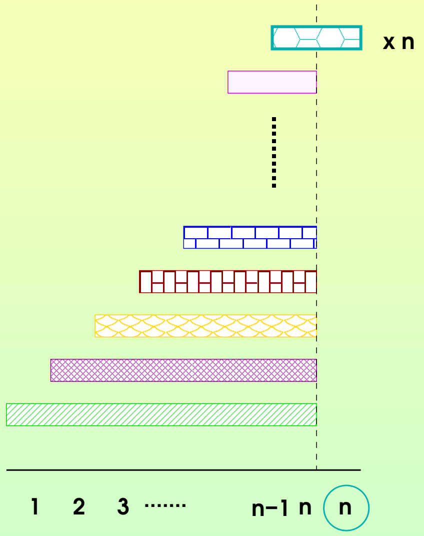
Is that good enough?

- The forward phase is *myopic*: at every point we select the **best** choice available “locally".
- Let’s run through an example to see how the forward phase can perform significantly worse than the actual OPT.

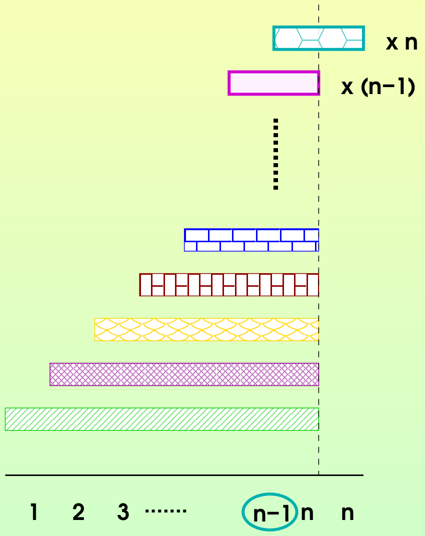
Running Example



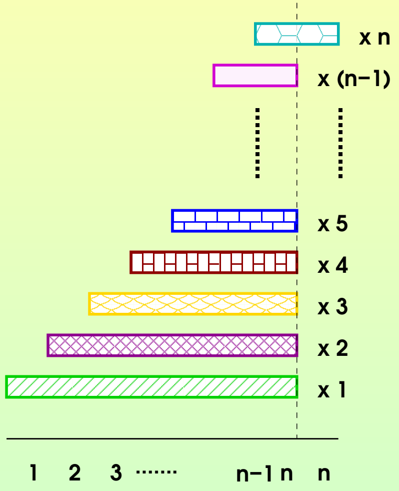
Running Example - Forward Phase



Running Example - Forward Phase



Running Example - Forward Phase



at end of forward phase

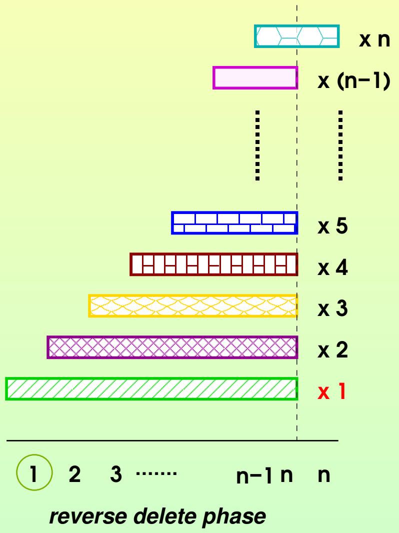
The cost of this solution $= O(n^2)$.

The Reverse Phase

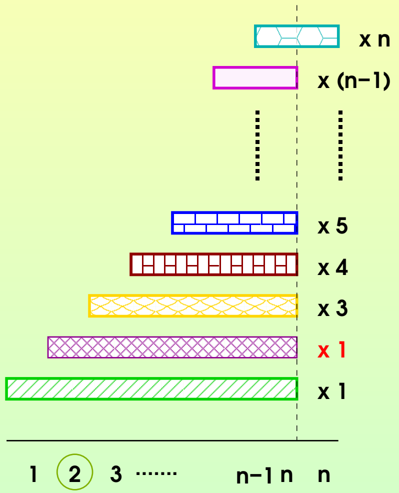
Motif

Remove redundancies in the solution produced by the Forward Phase; return a minimal feasible solution contained in the solution produced by the Forward Phase.

Running Example - Reverse Phase

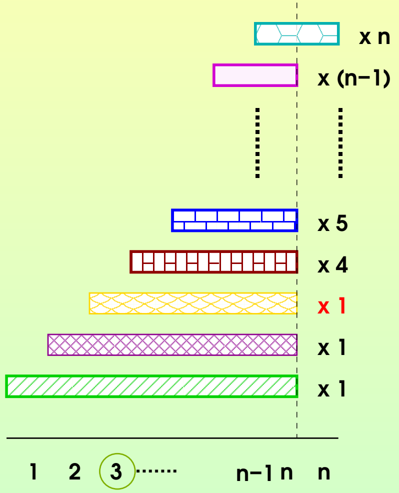


Running Example - Reverse Phase



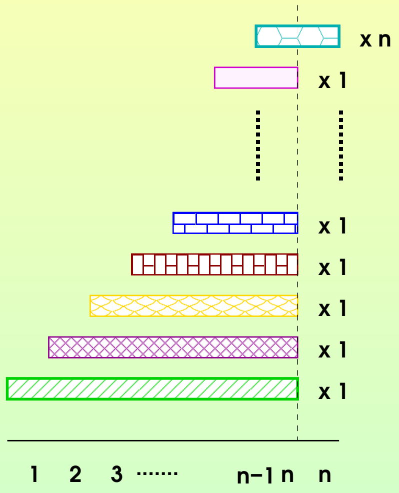
reverse delete phase

Running Example - Reverse Phase



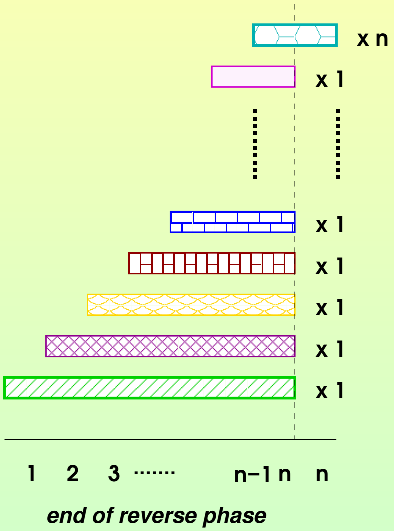
reverse delete phase

Running Example - Reverse Phase



end of reverse phase

Running Example - Reverse Phase



The cost of this solution $\approx 2n$.

Reverse Phase

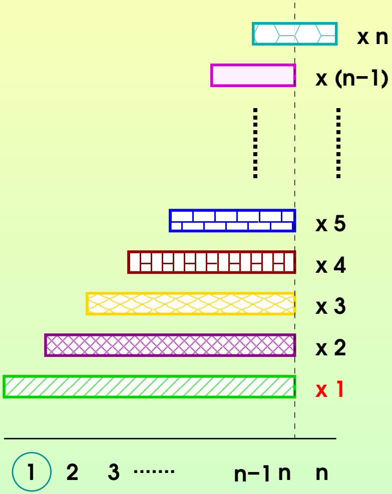
- The output of the Reverse Phase is a 2-factor approximation to the full cover problem.
- The analysis follows the primal-dual paradigm.
- But the output is not **nice**! On to Reverse Phase*. . .

Reverse Phase*

Motif

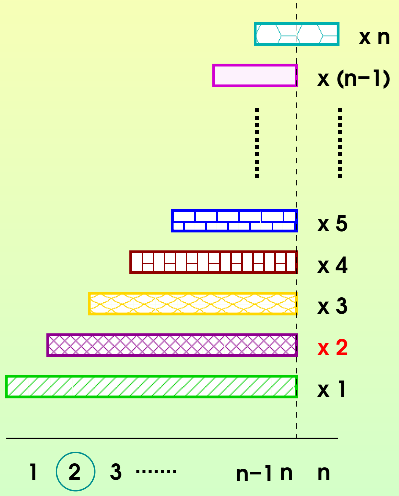
If in the reverse phase, we cannot delete all the copies of an interval included in the forward phase, then retain **all the copies**.

Running Example – Reverse Phase*



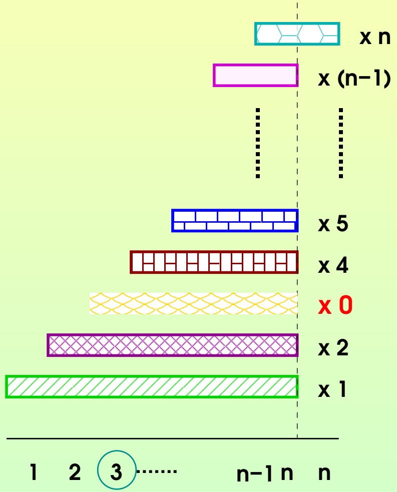
modified reverse phase

Running Example – Reverse Phase*



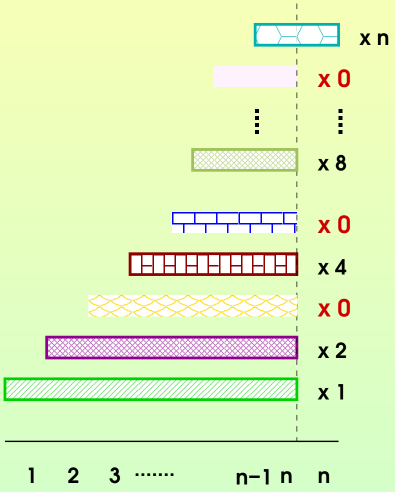
modified reverse phase

Running Example – Reverse Phase*



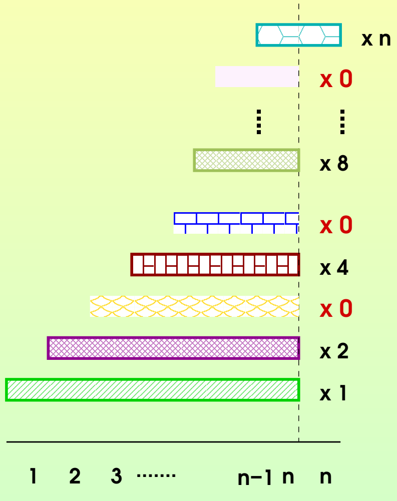
modified reverse phase

Running Example – Reverse Phase*



modified reverse phase

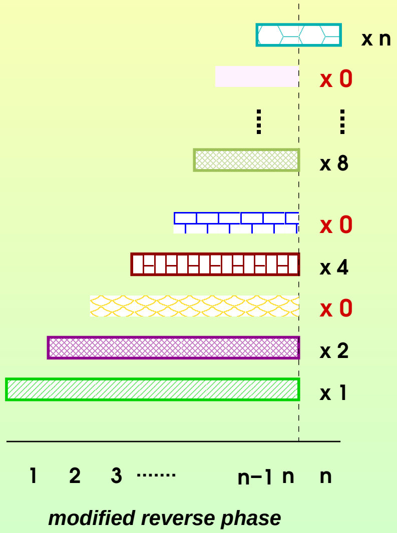
Running Example – Reverse Phase*



modified reverse phase

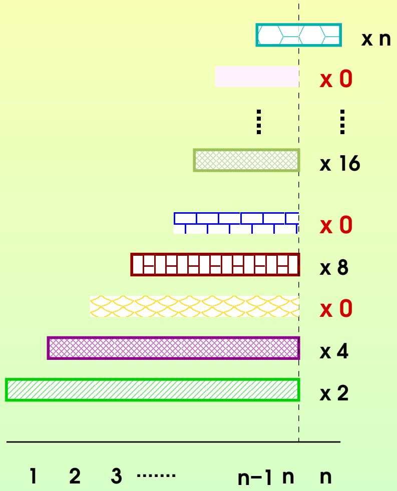
The cost of this solutions is $\approx 3n$, but. . .still not nice

Running Example – Reverse Phase*



The cost of this solutions is $\approx 3n$, but. . .still not **nice**.

Running Example – Reverse Phase*



twice of previous solution

Now, this is **nice**. . ., with cost $\approx 6n$

Reverse Phase*

- A primal-dual analysis shows that this is an 8-factor approximation.

Summing up

- The general case of arbitrary heights introduces an extra factor of 2 in the approximation factor, owing to knapsack cover reasons.
- Altogether, we get a 16-factor approximation for the general case.

THANK YOU.

Open Questions

- What if we were not allowed copies of intervals; i.e. every interval can be picked at most once?
- One can show a 4-factor approximation to the full cover version; the partial cover version is open.

Open Questions

- What if we were not allowed copies of intervals; i.e. every interval can be picked at most once?
- One can show a 4-factor approximation to the full cover version; the partial cover version is open.

Comments about the $\{0, 1\}$ version

- Suppose we take motivation from the Vertex Cover problem; redefine the Vertex Cover problem to have demands d_e on the edges, where $d_e \in \{1, 2\}$.
- Consider the partial covering question, where each vertex can be picked at most once.
- This problem turns out to be (almost) as **hard** as the Densest k-subgraph problem!
- However, if we relax the condition that vertices can be picked at most once, we get a constant factor approximation to the partial covering question.

Comments about the $\{0, 1\}$ version

- Suppose we take motivation from the Vertex Cover problem; redefine the Vertex Cover problem to have demands d_e on the edges, where $d_e \in \{1, 2\}$.
- Consider the partial covering question, where each vertex can be picked at most once.
- This problem turns out to be (almost) as **hard** as the Densest k-subgraph problem!
- However, if we relax the condition that vertices can be picked at most once, we get a constant factor approximation to the partial covering question.

Comments about the $\{0, 1\}$ version

- Suppose we take motivation from the Vertex Cover problem; redefine the Vertex Cover problem to have demands d_e on the edges, where $d_e \in \{1, 2\}$.
- Consider the partial covering question, where each vertex can be picked at most once.
- This problem turns out to be (almost) as **hard** as the Densest k-subgraph problem!
- However, if we relax the condition that vertices can be picked at most once, we get a constant factor approximation to the partial covering question.

THANK YOU.