# Approximation Algorithms for Covering and Packing Problems on Paths

Arindam Pal
Advisors: Prof Amit Kumar and Prof Naveen Garg

Department of Computer Science and Engineering
Indian Institute of Technology Delhi

January 31, 2014
IIT Delhi

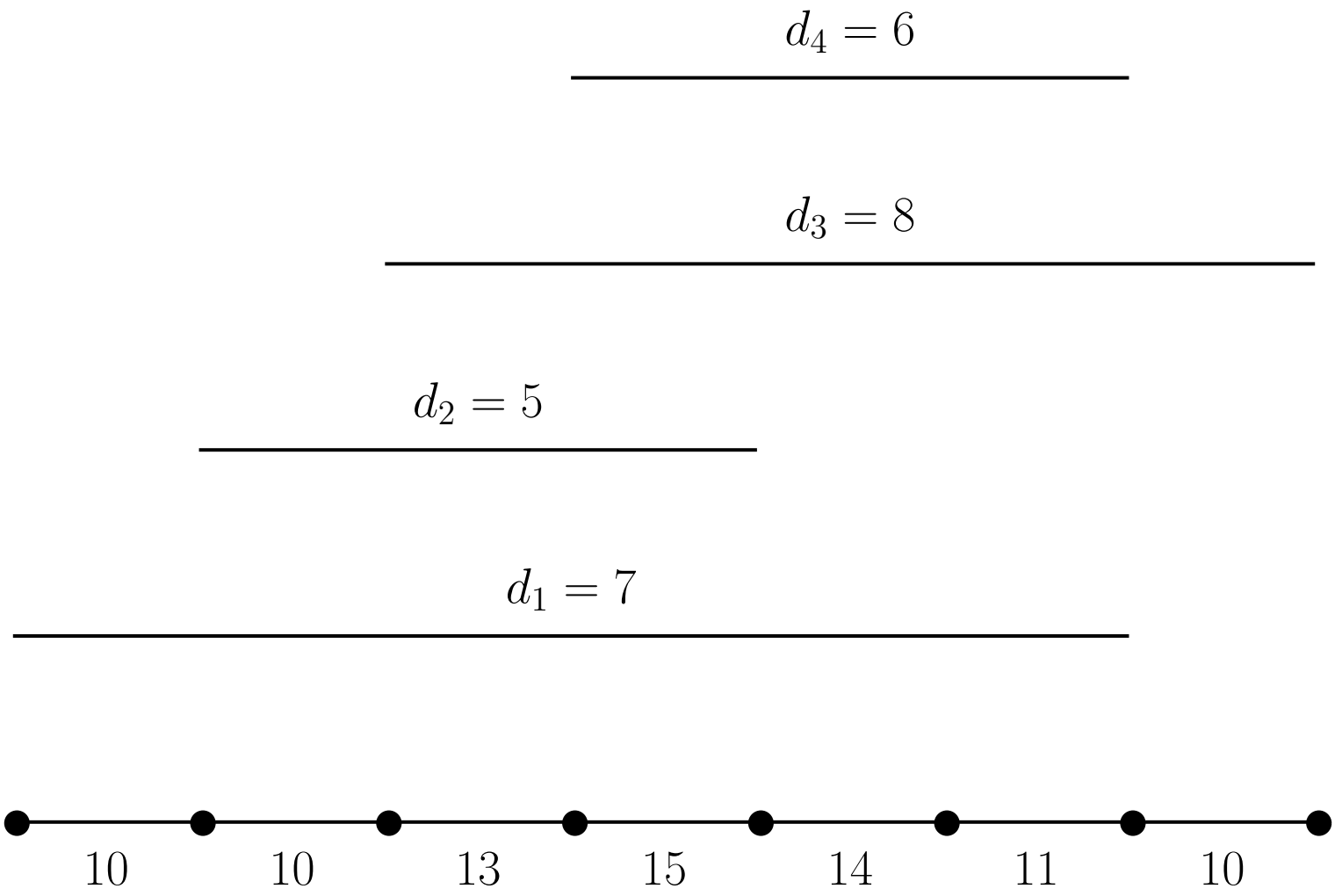
# Agenda

- The Unsplittable Flow Problem and its variants
- Survey of existing results and our contribution
- Resource allocation for scheduling jobs
- Related work and our contribution
- Conclusion and future work

# Unsplittable Flow Problem with Rounds (ROUND-UFP)

- Given a path $P = (v_1, e_1, v_2, e_2, \ldots, e_{n-1}, v_n)$ on $n$ nodes.
- Edge $e_i$ has capacity $c(e_i) \equiv c_i$.
- There are $k$ intervals (requests) $I_1, \ldots, I_k$.
- $I_i = [s_i, t_i]$ and there is a demand $d_i$ associated with it.
- A set of intervals $\mathcal{I}$ is *feasible* if the total demand of all intervals in $\mathcal{I}$ passing through any edge $e$ does not exceed it's capacity $c(e)$.
- Goal is to partition the requests $I_1, \ldots, I_k$ into a number of sets such that each set is feasible and the total number of sets is minimized.
- We can think of this as assigning colors to intervals so that each color class is feasible and we want to minimize the number of colors.
- This can also be thought of as routing the requests in a feasible manner in a number of rounds.
- Can be studied under offline or online setting.

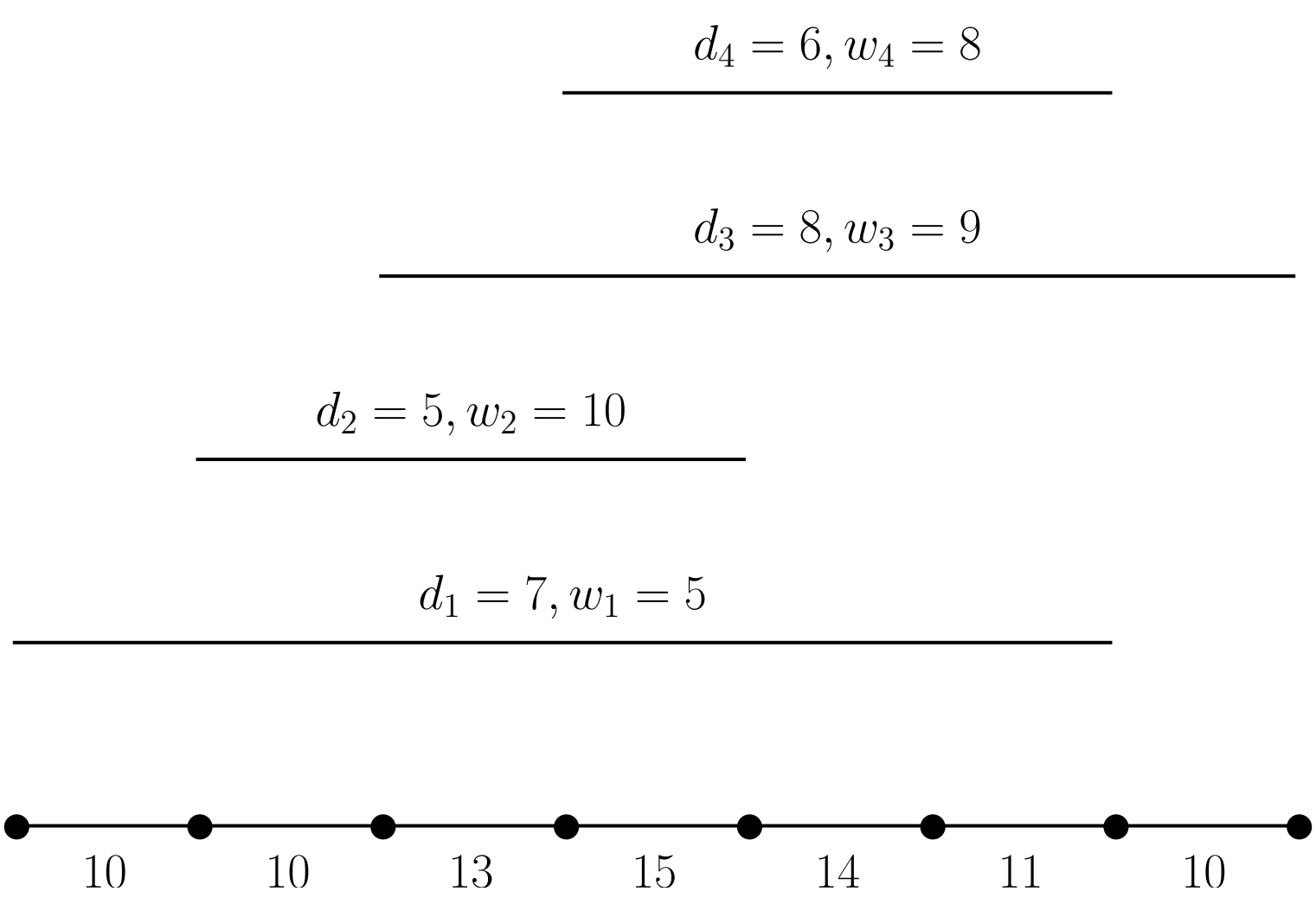
# A sample ROUND-UFP instance



# The MAX-UFP and BAG-UFP problems

- For MAX-UFP, the setting is similar to ROUND-UFP except, for each request $I_i$ there is a profit $w_i$.
- If we can route a request, we get the corresponding profit.
- The objective is to select a feasible subset of requests having the maximum profit.
- In BAG-UFP, there is a set of bags each containing a set of requests.
- Each bag $B_j$ has a profit $p_j$.
- At most one request can be selected from each bag. If we select a request from a bag $B_j$, we get the profit $p_j$.
- The objective is to select a feasible (both bag and capacity constraints) subset of requests of maximum profit.

# A sample MAX-UFP instance



# Motivation

- The path graph is a natural setting for many applications, where a limited resource is available and the amount of the resource varies over time.
- Many combinatorial optimization problems which are NP-HARD on general graphs remain NP-HARD on paths.
- We can represent time instants as vertices, time intervals as edges and the amount of resource available at a time interval as the capacity of the corresponding edge.
- The requirement of a resource between two time instants can be represented as a demand between the corresponding vertices with a certain profit associated with it.

# Application of Round-UFP

- Consider an optical line network, where each color corresponds to a distinct frequency in which the information flows.
- Different links along the line have different capacities, which are a function of intermediate equipment along the link.
- Each request uses the same bandwidth on all links that this request contains.
- As the number of distinct available frequencies is limited, minimizing the number of colors for a given sequence of requests is a natural objective.

# Related Work for Round-UFP

- Round-UFP is NP-Hard for arbitrary demands since, if we take $P$ to be a single edge, this is the Bin-Packing problem.
- If all capacities and demands are 1, this is the Interval Graph Coloring problem, for which a greedy algorithm gives the optimum coloring with $\omega$ colors, where $\omega$ is the maximum clique size of the *interval graph*.
- For the corresponding online problem, Kierstead and Trotter gave an online algorithm which uses at most $3\omega - 2$ colors. They also gave a lower bound of $3\omega - 2$ on the number of colors required in any coloring output by any deterministic online algorithm.
- The best upper bound known for the First-Fit algorithm due to Pemmaraju et al. is $8\omega$, and a lower bound of $4.4\omega$ was shown by Chrobak and Slusarek.

# Related Work for Round-UFP ...

- For unit capacities and arbitrary demands, Narayanaswamy gave a 10-competitive algorithm. Epstein et al. proved a lower bound of $\frac{24}{7} \approx 3.43$ for this problem.
- For arbitrary capacities and demands, Epstein et al. gave a 78-competitive algorithm, assuming the maximum demand is at most the minimum capacity (*no-bottleneck assumption*).
- They also proved that without this assumption, there is no deterministic online algorithm for interval coloring with nonuniform capacities and demands, that can achieve a competitive ratio better than $\Omega(\log \log n)$ or $\Omega\left(\log \log \log \left(\frac{c_{\max}}{c_{\min}}\right)\right)$. Here, $c_{\max}$ and $c_{\min}$ are the maximum and minimum edge capacities of the path respectively.

# Application of MAX-UFP and BAG-UFP

- Consider a system offering a resource in limited quantity, where the availability of this resource varies over time.
- There are a set of users who want to use different amounts of this resource over different time intervals and are ready to pay its owner.
- The aim of the owner is to select a subset of these users to maximize his profit, while satisfying the resource availability constraint at each instant.
- The concept of *bag constraints* (at most one request can be selected from each bag) in BAG-UFP arises in a situation where a job can specify a set of possible time intervals where it can be scheduled.

# Related Work for MAX-UFP and BAG-UFP

- MAX-UFP and BAG-UFP are *weakly* NP-HARD, since they contain the KNAPSACK problem as a special case, where there is just one edge.
- Recently, it has been proved that MAX-UFP is *strongly* NP-HARD, even for the restricted case where all demands are chosen from $\{1, 2, 3\}$ and all capacities are uniform.
- This means that the problem does not have a *fully polynomial time approximation scheme* (FPTAS).
- However, the problem is not known to be APX-hard, so a *polynomial time approximation scheme* (PTAS) may still be possible.
- When all capacities, demands and profits are 1, MAX-UFP specializes to the MAXIMUM EDGE-DISJOINT PATHS problem.

# Approximation Algorithms for MAX-UFP and BAG-UFP

- For MAX-UFP, Chekuri et al. gave a $(2+\epsilon)$-approximation algorithm on paths and a $48$-approximation on trees under NBA.
- Bonsma et al. gave a polynomial time $(7+\epsilon)$-approximation algorithm for any $\epsilon > 0$ without NBA.
- This was later improved to a $(2+\epsilon)$-approximation algorithm by Anagnostopoulos et al., matching the bound of Chekuri et al.
- Chekuri et al. gave a $O(\log^2 n)$-approximation algorithm on trees without NBA.
- Chakaravarthy et al. gave a 120-approximation algorithm for the BAG-UFP problem on paths assuming NBA.

# Our Results (assuming NBA)

- 3-approximation algorithm for unit capacities, arbitrary demands for Round-UFP.
- 24-approximation algorithm for arbitrary capacities and demands with NBA for Round-UFP.
- 17-approximation algorithm for Max-UFP.
- 65-approximation algorithm for Bag-UFP.
- 58-competitive online algorithm for Round-UFP.
- 64-approximation algorithm for Round-UFP on trees.
- We give a unified framework for solving all these problems.

# Preliminaries

- $F_e$ = Set of all requests passing through edge $e$.
- $l_e$ = Total demand of all requests passing through $e = \sum_{i:I_i \in F_e} d_i$, is the *load* on edge $e$.
- $r_e = \left\lceil \frac{l_e}{c_e} \right\rceil$, is the *congestion* on edge $e$.
- $r = \max_{e \in E} r_e$, is the maximum congestion on any edge.
- Let $\mathrm{OPT}$ be the minimum number of colors required for the given problem instance. Clearly, $\mathrm{OPT} \geq r$.
- If $\omega$ demands are mutually incompatible with each other, then each of them has to be assigned a different color. Hence, $\mathrm{OPT} \geq \omega$.
- The *bottleneck edge* $b_i$ of a request $I_i$ is the minimum capacity edge on the path from $s_i$ to $t_i$.

# Unit capacities and arbitrary demands for ROUND-UFP

- Separate the requests based on whether $d_i > \frac{1}{2}$ (large demands) or $d_i \leq \frac{1}{2}$ (small demands).
- Two large demands passing through a common edge can't be given the same color.
- This is like the interval coloring problem, which can be solved optimally.
- We sort the small demands based on their left endpoints and then use first-fit.
- It can be proven that this requires at most $2 \cdot \mathrm{OPT}$ colors.
- In total, we require at most $3 \cdot \mathrm{OPT}$ colors.

# Arbitrary capacities, arbitrary demands for ROUND-UFP

- Separate the requests based on whether $d_i > \frac{1}{4}b_i$ (large demands) or $d_i \leq \frac{1}{4}b_i$ (small demands), where $b_i$ is the bottleneck edge capacity.
- For large demands, round down capacity of every edge to the nearest multiple of $c_{\min}$. This will lose a factor of 2.
- Round up every demand to $c_{\min}$. This will lose a factor of 4.
- The resulting instance has uniform demands, which can be colored with $r$ colors. So, large demands require $8r$ colors.
- We sort the small demands based on their left endpoints and then assign a demand to the first color where the total load on the bottleneck edge $e$ is at most $\frac{c_e}{16}$.
- It can be proven that this requires at most $16r$ colors and the coloring is feasible.
- In total, we require at most $24r \leq 24 \cdot \mathrm{OPT}$ colors.

# Linear Programming formulation for MAX-UFP

A natural linear programming formulation for MAX-UFP on a path is given below. Here $x_i$ denotes the fraction of the demand $i$ that is satisfied and $I_i$ is the unique path between $s_i$ and $t_i$.

$$\begin{aligned} \textbf{maximize} \quad & \sum_{i=1}^{k} w_i x_i && \textbf{(UFP-LP)} \\ \textbf{such that} \quad & \sum_{i:e\in I_i} d_i x_i \leq c_e && \forall e \in E \\ & 0 \leq x_i \leq 1 && \forall i \in \{1,\ldots,k\} \end{aligned}$$

If we replace the constraints $x_i \in [0,1]$ by the constraints $x_i \in \{0,1\}$ we get an integer program, which precisely models the MAX-UFP problem.

# Convex decomposition of a fractional LP solution

- Suppose $x$ is a feasible fractional solution for a maximization LP and $z_1, \ldots, z_k$ be feasible integral solutions for the LP.
- Let $x = \sum_{i=1}^{k} \lambda_i z_i$, where $\sum_{i=1}^{k} \lambda_i = \alpha$.
- Then the best solution, say $z_{\max}$ among $z_1, \ldots, z_k$ is at least $\frac{1}{\alpha}$ fraction of the value of $x$.
- This can also be viewed as covering the fractional solution with some integral solutions, which is like coloring.

# Max-UFP and Bag-UFP

- Separate the requests based on whether $d_i > \frac{1}{4}b_i$ (large demands) or $d_i \leq \frac{1}{4}b_i$ (small demands), where $b_i$ is the bottleneck edge capacity.
- For Max-UFP, large demands instance can be solved optimally using dynamic programming.
- We can get a 16-approximation using ideas from Round-UFP.
- Overall, we get a 17-approximation.
- For Bag-UFP, there is a 48-approximation for large demands.
- We can get a 17-approximation using ideas from Round-UFP and the fact that a factor of 1 will be added due to bag constraints.
- Overall, we get a 65-approximation.

# Online Interval Coloring with capacities and demands

- We scale down all capacities and demands by a factor of $c_{\min}$, so that the new $c_{\min} = 1$ and the new $d_{\max} \leq 1$.
- Then we round down all edge capacities to the nearest power of 2, so that if $c(e) \in [2^k, 2^{k+1})$ then the new $c(e) = 2^k$.
- The *class* of a demand $d_i$ is defined as $\ell_i = \log_2 c(b_i)$.
- For a demand $d_i$ in class $j \geq 1$, we call it a small demand if $d_i \leq \min(1, 2^{j-3})$.
- For a demand $d_i$ in class $0$, we call it a small demand if $d_i \leq \frac{1}{4}$.
- Note that large demands can exist only in classes 0, 1 and 2.

# Schematic representation of classes of demands

| Class | Small | Large | Bottleneck capacity | Allocated capacity |
|---|---|---|---|---|
| 0 | $\left(0, \frac{1}{4}\right]$ | $\left(\frac{1}{4}, 1\right]$ | 1 | 1 |
| 1 | $\left(0, \frac{1}{4}\right]$ | $\left(\frac{1}{4}, 1\right]$ | 2 | 1 |
| 2 | $\left(0, \frac{1}{2}\right]$ | $\left(\frac{1}{2}, 1\right]$ | 4 | 2 |
| 3 | $(0, 1]$ | NONE | 8 | 4 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $j$ | $(0, 1]$ | NONE | $2^j$ | $2^{j-1}$ |

# Handling small demands

- Small demands are $\frac{1}{4}$-small.
- The resulting instance has uniform capacity.
- 4-competitive algorithm for this.
- Additional loss of a factor of 8 due to rounding and allocating only $2^{j-1}$ capacity instead of $2^j$.
- So this is 32-competitive.

# Algorithm for small demands and uniform capacity

- Our algorithm partitions intervals into disjoint sets and colors each set independently with separate colors.
- $S = \{S_1, S_2, \ldots\}$ is the family of sets containing already processed requests.
- $S_i$ is the set of requests at *level* $i$.
- For each new request $R$, we look for a set with the lowest possible index $k$ such that the total load of all the demands in $\left(\bigcup_{i=1}^{k} S_i\right) \cup \{R\}$ on any edge $e$ of $R$ does not exceed $\frac{1}{4}k$.
- If on any edge $e$ this inequality is violated, we call $e$ a *critical edge* of $R$ on that level.
- Note that $e$ is the edge which prevented $R$ to be put on level $k$.

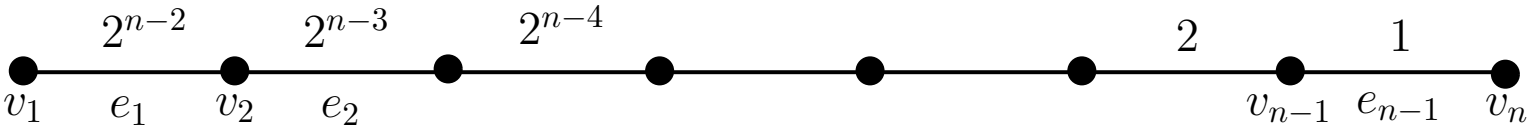
# An online algorithm for $\frac{1}{4}$-small demands

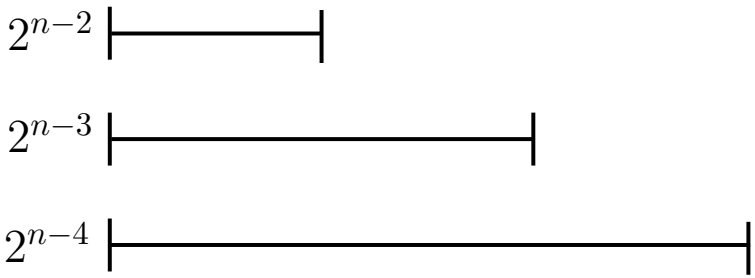
// $k_i$ is the number of levels used.
$k_i \leftarrow 1$;
**while** *there are still requests in the input* **do**
    let $R_i = (s_i, t_i, d_i)$ be the next request;
    find the smallest level $k \in \{1, \ldots, k_i\}$ such that for every edge $e \in I_i$, the total load of the requests in $\cup_{k'=1}^{k} S_{k'}$ (including $R_i$), i.e., $l_e(\cup_{k'=1}^{k} S_{k'} \cup \{R_i\})$, is at most $\frac{k}{4}$;
    **if** *no such level is found* **then**
        $k_i = k_i + 1$;
        go to line 7.
    **end**
    // assign $R_i$ to level $k$.
    $S_k \leftarrow S_k \cup \{R_i\}$;
**end**

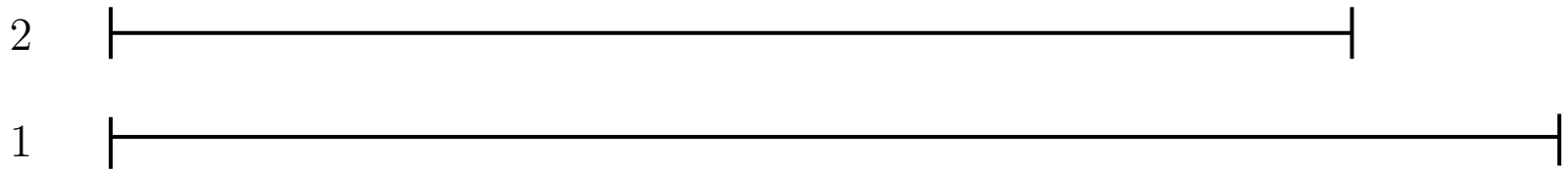
# Competitive ratio

- Small demands require at most $32 \cdot \mathrm{OPT}$ colors.
- Large demands in classes 0, 1 and 2 require at most $26 \cdot \mathrm{OPT}$ colors.
- Total number of colors required is at most $58 \cdot \mathrm{OPT}$.
- Hence, this algorithm is 58-competitive.

# How bad the congestion bound can be?



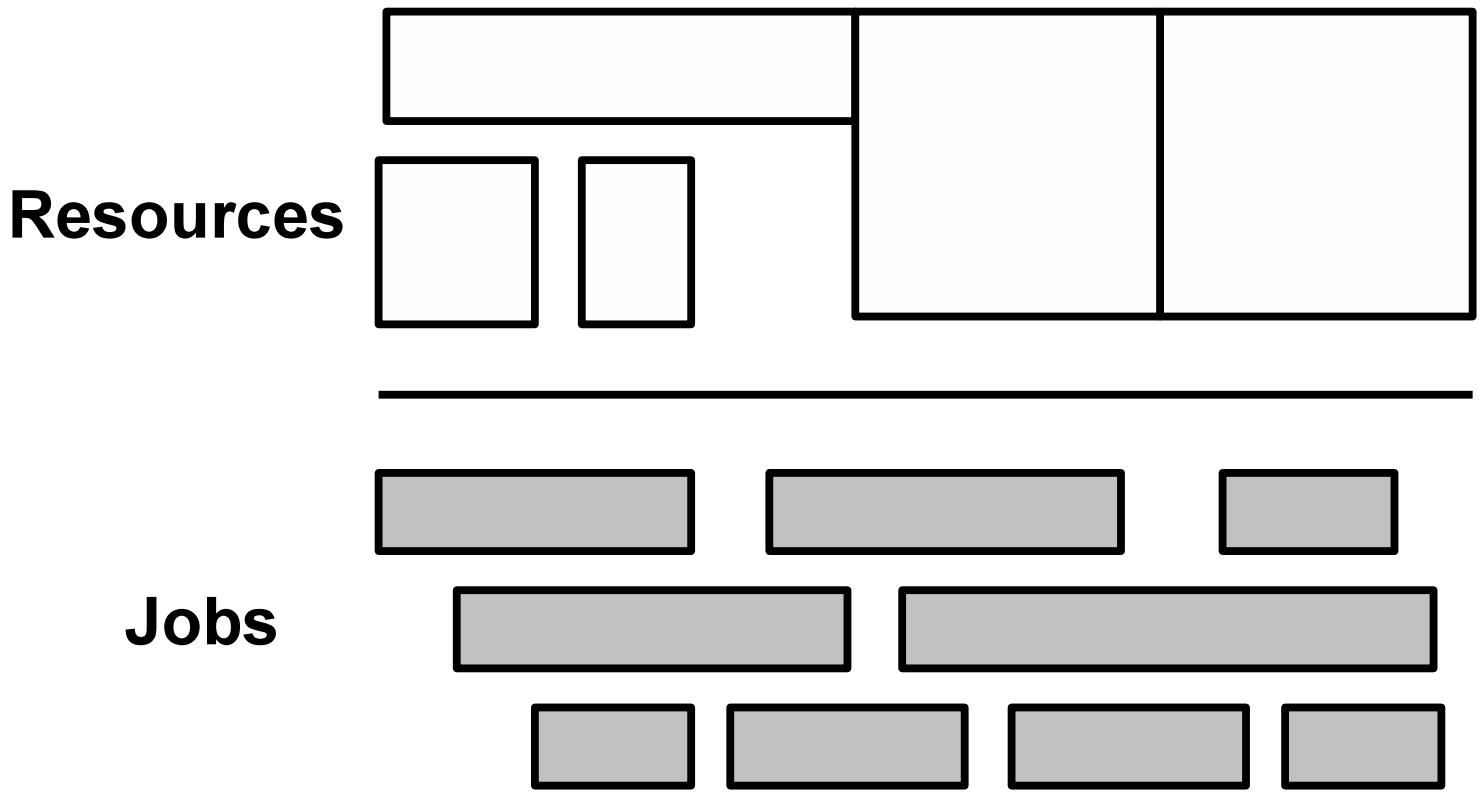




$$\text{OPT} = n, r = 2, \omega = n.$$

# Resource allocation for scheduling jobs

- The timeline is uniformly divided into discrete timeslots $t$ from 1 to $T$.
- We have a set of *jobs* $\mathcal{J}$, and a set of *resources* $\mathcal{I}$.
- Each job $j \in \mathcal{J}$ is specified by an interval $I(j) = [s(j), e(j)]$, where $s(j)$ and $e(j)$ are the *start time* and *end time* of the job $j$.
- Each job $j$ has a *demand* $d(j)$, which we assume to be one unit.
- Further, each resource $i \in \mathcal{I}$ is specified by an interval $I(i) = [s(i), e(i)]$, where $s(i)$ and $e(i)$ are the *start time* and the *end time* of the resource $i$.
- Each resource $i$ has a *capacity* $w(i)$ and a *cost* $c(i)$.
- A feasible solution is a set of resources such that at any timeslot, the total capacity of the resources is at least the total demand of the jobs active at that timeslot, i.e., the selected resources must cover the jobs. We call this the Resource Allocation problem (ResAll).

# An instance of a resource allocation problem



# The partial covering and prize-collecting versions

- PartialResAll: In this problem, the input also specifies a number $k$ (called the *partiality parameter*) that indicates the number of jobs to be covered. A feasible solution is a pair $(R, J)$ where $R$ is a multiset of resources and $J$ is a set of jobs such that $R$ covers $J$ and $|J| \geq k$. The cost of the solution is the sum of the costs of the resources in $R$ (taking copies into account). The problem is to find a feasible solution of minimum cost.
- PrizeCollectingResAll: In this problem, every job $j$ also has a penalty $p_j$ associated with it. A feasible solution is a pair $(R, J)$ where $R$ is a multiset of resources and $J$ is a set of jobs such that $R$ covers $J$. The cost of the solution is the sum of the costs of the resources in $R$ (taking copies into account) and the penalties of the jobs not in $J$. The problem is to find a feasible solution of minimum cost.

# Related work

- Partial covering and prize-collecting problems are natural generalization of full cover problems. Some examples are $k$-MST, partial vertex cover and prize-collecting Steiner tree problems.
- Bar-Noy et al. studied the full cover version where all the jobs have to be covered. They gave a $4$-approximation algorithm for this problem.
- Chakaravarthy et al. considers a scenario, where the timeslots have demands and a solution must satisfy the demand for at least $k$ of the timeslots. They gave a 16-approximation for this problem.
- In our setting, a solution needs to satisfy $k$ *jobs*, where each job can span multiple timeslots. A job may not be completely spanned by any resource, and thus may require *multiple* resource intervals for covering it. This is a generalization of both these problems.

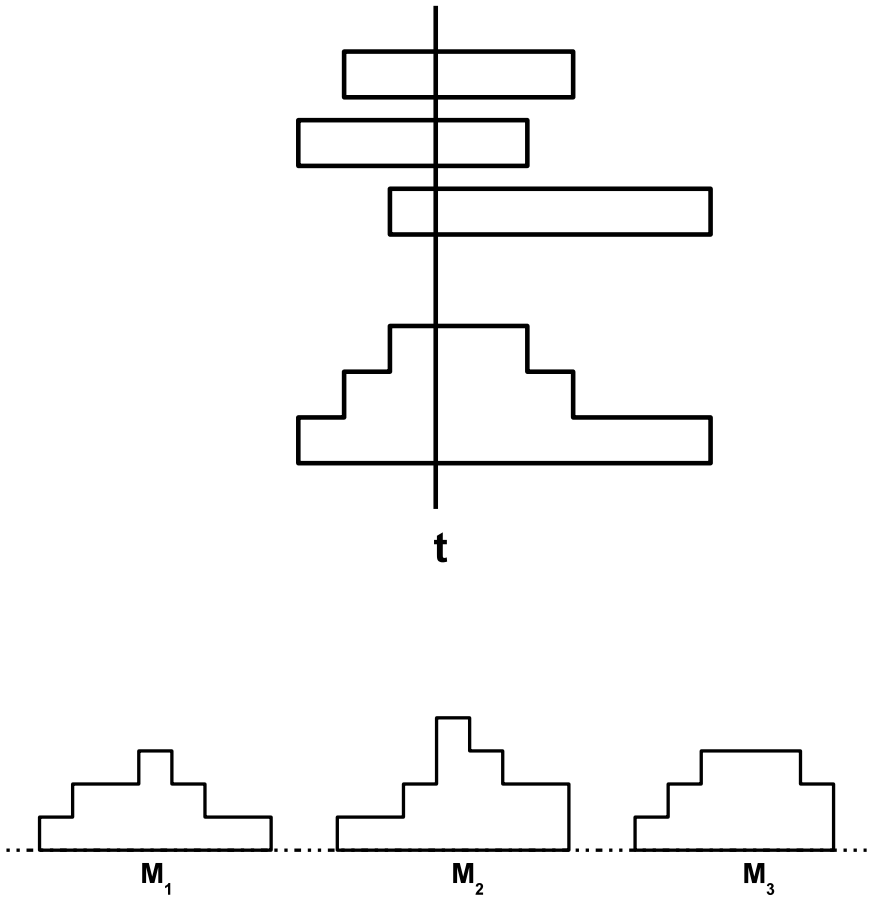
# Our Results

- We present an $O(\log(n+m))$-approximation algorithm for the PartialResAll problem, where $n$ is the number of jobs and $m$ is the number of resources respectively.
- For the PrizeCollectingResAll problem, we give a $4$-approximation algorithm.

# Approximation Algorithm for PartialResAll

- A collection of jobs $M$ is called a *mountain*, if there exists a timeslot $t$ such that all the jobs in this collection span the timeslot $t$.
- A collection of jobs $\mathcal{M}$ is called a *mountain range*, if the jobs can be partitioned into a sequence $M_1, M_2, \ldots, M_r$ such that each $M_i$ is a mountain and the spans of any two mountains are non-overlapping.
- We show that the input set of jobs can be partitioned into $O(\log(m+n))$ mountain ranges.
- Then we give a constant factor approximation algorithm for the special case of the PartialResAll problem, where the input set of jobs form a single mountain range $\mathcal{M}$.
- Using these two results along with dynamic programming, we get an approximation algorithm for the PartialResAll problem.

# Mountain and mountain ranges



# Conclusion and future work

- Can we improve the approximation factor of ROUND-UFP, MAX-UFP and BAG-UFP problems on paths and trees?
- What is the approximability of these problems without the *no-bottleneck assumption*? For MAX-UFP on paths, a $(2+\epsilon)$-approximation is known.
- Is there a better constant factor competitive algorithm for the ONLINE INTERVAL COLORING problem?
- Is there a constant factor approximation algorithm for the PARTIALRESALL problem?
- Is there a constant factor approximation algorithm for the PRIZECOLLECTINGRESALL problem having the *Lagrangian Multiplier Preserving* property?
- What is the hardness of approximation of these problems?