Approximation Algorithms for Unsplittable Flow and Interval Coloring Problems on Paths and Trees

Arindam Pal
arindamp@cse.iitd.ac.in

Department of Computer Science and Engineering
Indian Institute of Technology Delhi

November 2, 2012

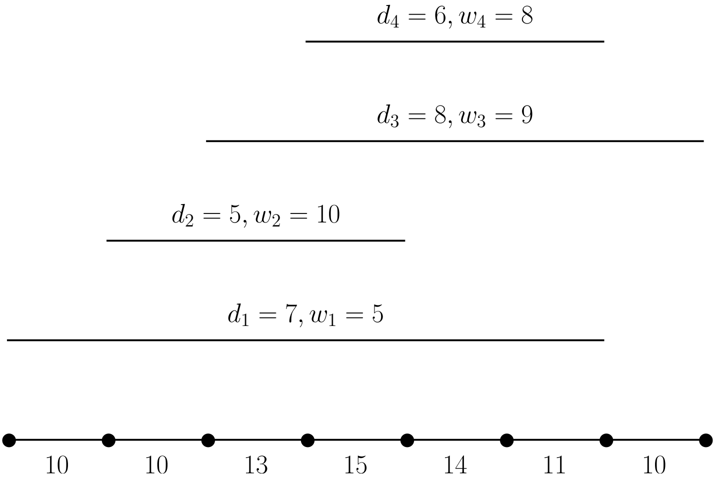
Agenda

- The Unsplittable Flow Problem and its variants
- Survey of existing results and our contribution
- Approximation algorithms for ROUND-UFP
- Approximation algorithms for MAX-UFP and BAG-UFP
- 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 $c(e)$.
- The 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 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.

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 ω colors, where ω 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 8ω, and a lower bound of 4.4ω was shown by Chrobak and Slusarek.

- 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 algorithm on trees under NBA.
- These algorithms are based on the idea of rounding a LP relaxation of Max-UFP.
- Bonsma et al. gave a polynomial time $(7+\epsilon)$-approximation algorithm for any $\epsilon > 0$, and a 25.12-approximation algorithm with running time $O(n^4 \log n)$ without NBA.
- 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

- Optimal algorithm for unit demands, arbitrary capacities for ROUND-UFP.
- 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 OPT be the minimum number of colors required for the given problem instance. Clearly, $\mathrm{OPT} \geq r$.
- If ω 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.

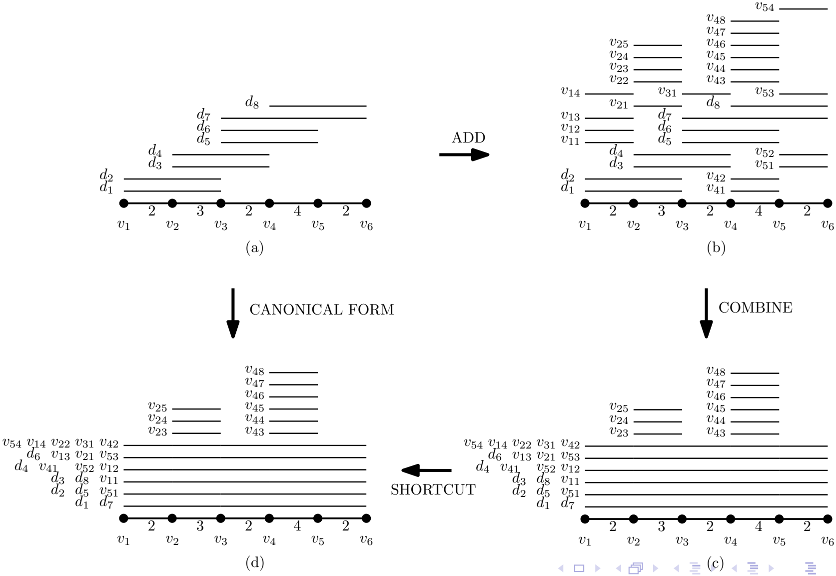
An Algorithm for ROUND-UFP for Unit Demands

- Preprocess the input graph to transform it into a *canonical form*.
- Create a bipartite graph with source and destination vertices.
- Find r edge-disjoint perfect matchings.
- Recover the coloring in the original graph.

Preprocessing and Canonical Form

- ADD: In the original problem instance, introduce additional intervals of unit demand and unit length so that the congestion on every edge becomes r.
- COMBINE: Suppose there exists a pair of intervals $I_i = [s_i, t_i]$ and $I_j = [s_j, t_j]$ such that $t_i = s_j$. We combine these two intervals and replace them with a single (longer) interval $I_k = [s_i, t_j]$. We keep repeating this process until we can't find any such pair of intervals.
- SHORTCUT: If $(v_i, e_i, \ldots, e_{j-1}, v_j)$ is a path such that none of the vertices between v_{i+1} and v_{j-1} are the source or destination of any request, then we can replace the entire path with a single edge e between v_i and v_j, whose capacity is $c_e = \min(c_i, \ldots, c_{j-1})$. Since no request is starting or ending at these vertices, the load on all these edges $(rc_t, t = i, \ldots, j-1)$ is the same. Hence, the capacity of all these edges $(c_t, t = i, \ldots, j-1)$ is also the same.

Transforming a Problem Instance into Canonical Form



Properties of the Canonical Form

- The congestion on every edge is r and the load on edge e is rc_e.
- There is no vertex where one request ends and another request starts.
- Every vertex is either a source or a destination for some requests, but not both. This would mean that the whole vertex set V can be partitioned into two disjoint sets, namely *source vertices* S and *destination vertices* T, depending on whether requests start or end at a vertex.
- The first vertex v_1 is always a source vertex and the last vertex v_n is always a destination vertex.

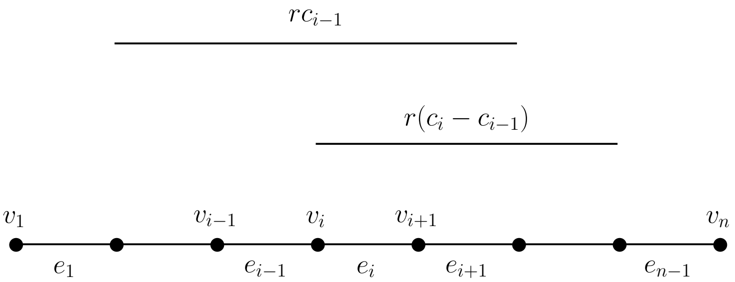
Properties of the Canonical Form continued...

- The number of requests starting/ending at a vertex v_i is $r|c_i - c_{i-1}|$. [1]
  - If $c_i > c_{i-1}$, the number of requests starting at vertex v_i is $r(c_i - c_{i-1})$.
  - If $c_i < c_{i-1}$, the number of requests ending at vertex v_i is $r(c_{i-1} - c_i)$.
- The number of colors required for the original instance is at most the number of colors required for the canonical form. That is,

$$\mathrm{OPT}(\text{original problem instance}) \leq \mathrm{OPT}(\text{canonical form}).$$

[1] For this claim, we define $c_0 = c_n = 0$.

Number of requests starting at a vertex v_i

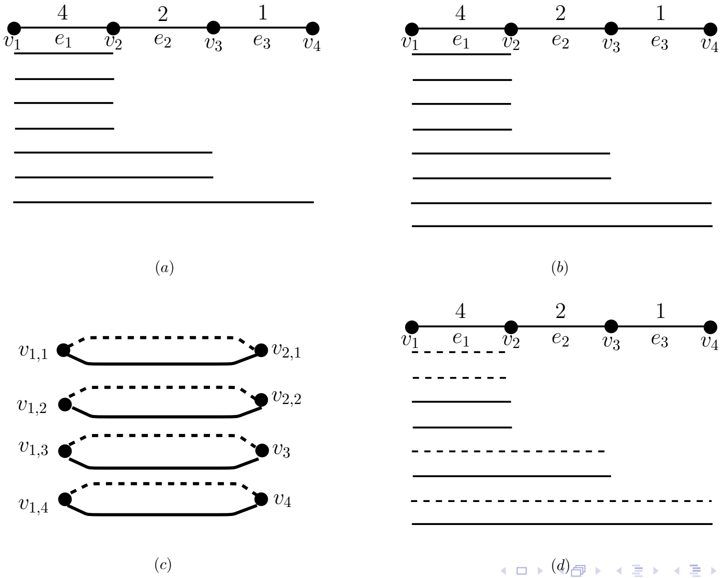


An Optimal Interval Coloring Algorithm for Unit Demands

- Let S and T be the set of source vertices and destination vertices. Every interval starts at some vertex $s \in S$ and ends at some vertex $t \in T$.
- For every vertex $v_i \in S$, $deg(v_i) = r(c_i - c_{i-1})$. Split the vertex into $(c_i - c_{i-1})$ vertices each having degree r.
- For every vertex $v_i \in T$, $deg(v_i) = r(c_{i-1} - c_i)$. Split the vertex into $(c_{i-1} - c_i)$ vertices each having degree r.
- Create a bipartite graph $H = (X, Y, F)$, where X is the set of all (including split) vertices created from vertices in S and Y is the set of all (including split) vertices created from vertices in T. F is the set of all edges between X and Y as defined by the requests in the canonical form. This can be done by splitting the vertices one at a time, first from the X side and then from the Y side.

- Suppose there is a vertex $x \in X$ from which rq edges go to vertices $y_1, \ldots, y_t \in Y$. We split the vertex x into vertices $x_1, \ldots, x_q \in X$. We distribute the edges from x so that the first r edges are incident on x_1, the next r edges are incident on x_2 and so on. We repeat this for all vertices in X one by one and then for all vertices in Y. The resulting graph is in general a r-regular multigraph.
- Since H is a r-regular bipartite graph, by Hall's theorem, it has a perfect matching M_1.
- Remove the edges in M_1. The resulting bipartite graph is $(r-1)$-regular and it has a perfect matching M_2.
- Continuing in this way we can see that F can be partitioned into r edge-disjoint union of perfect matchings $M_1, \ldots, M_r$.
- Since, we can assign one color to each matching M_i, every edge in H can be colored using r colors. Hence, every request in the canonical form and so in the original instance can be colored using r colors.

Illustration of the Interval Coloring Algorithm



Analysis of the Algorithm

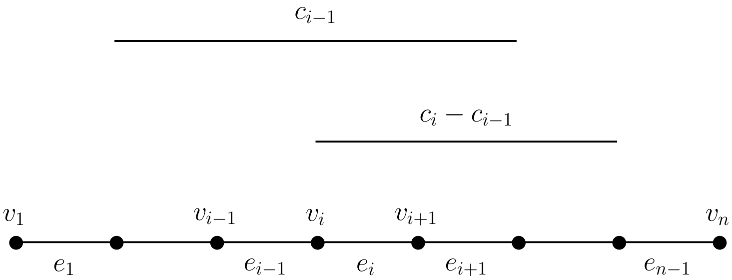
Theorem

The above algorithm feasibly colors all the requests with r colors.

- PROOF: Since we have already shown that every request in the canonical form and so in the original instance can be colored using r colors, the only thing to be proved is that requests in each color class is feasible.
- All that needs to be shown is that the total load on any edge e_i by all the requests in any color class is at most c_i.
- We proceed by induction on the index i of edge e_i.

- Base Case: For $i = 1$, since there are c_1 vertices created from v_1 from which requests passing through edge e_1 can originate, and in each matching at most one edge (request) incident on these vertices can be selected, the number of requests passing through edge e_1 in each color class is at most c_1.
- Induction Step: Suppose the result is true for $i - 1$. There are two cases to consider.
- v_i is a source vertex: We know that there are $(c_i - c_{i-1})$ vertices created from v_i from which requests starting at v_i can originate. The requests passing through edge e_i either started at v_i or are those also passing through edge e_{i-1}.
- Since v_i is a source vertex, no request passing through edge e_{i-1} can end at v_i and hence they must pass though e_i. The number of the first type of requests is $(c_i - c_{i-1})$, and the number of the second type of requests is c_{i-1} (inductively). Hence, the number of requests passing through edge e_i is at most $(c_i - c_{i-1}) + c_{i-1} = c_i$.

Feasibility of edge e_i



- v_i is a destination vertex: We know that there are $(c_{i-1} - c_i)$ vertices created from v_i to which requests ending at v_i can terminate. The requests passing through edge e_i either started at v_i or are those also passing through edge e_{i-1}.
- Since v_i is a destination vertex, no request passing through edge e_i can start at v_i and hence they must pass though e_{i-1}. The number of requests passing though e_{i-1} is c_{i-1} (inductively). Out of these, exactly $(c_{i-1} - c_i)$ requests end at v_i. Hence, the number of requests passing through edge e_i is at most $c_{i-1} - (c_{i-1} - c_i) = c_i$.
- Since, the total load on any edge e_i by all the requests in any color class is at most c_i, the algorithm colors all the requests with r colors in a feasible manner.

A 3-approximation Algorithm for Uniform Capacities

- Each edge of P has capacity c.
- A demand d_i is called *large* if $d_i > \frac{1}{2}c$. Otherwise, it is called *small*.
- We separate the demands into large and small demands.
- Let $\mathrm{OPT}(L)$ and $\mathrm{OPT}(S)$ be the optimum number of colors required for the instance containing only large demands and only small demands respectively.

An optimal algorithm for large demands

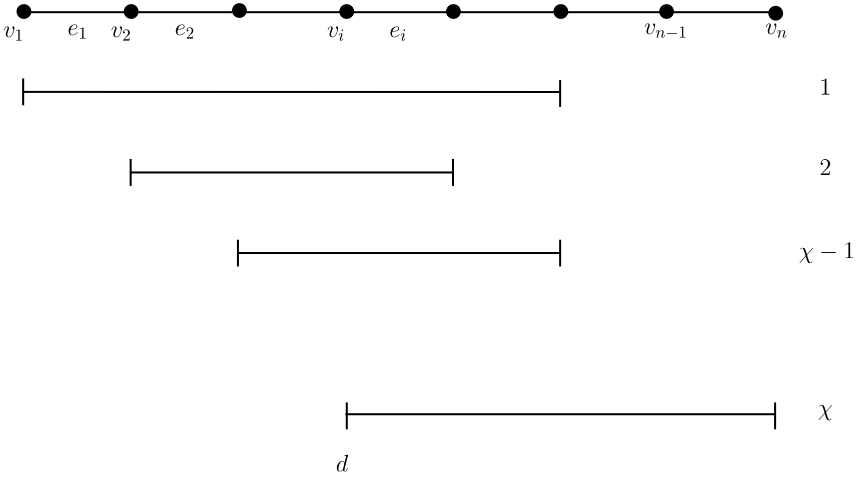
- Sort the demands based on their left endpoints.
- Let D_i be the set of requests starting at $v_i, 1 \leq i \leq n-1$.
- We will pack the requests in $D_1, \ldots, D_{n-1}$ in this order.
- Starting with the requests in D_1, we try to allocate the requests in $D_i, 1 \leq i \leq n-1$ in the current copy of the path, if it does not violate any edge capacities.
- Otherwise, we allocate a new copy and assign it there.

Lemma

If χ is the number of colors required for coloring the large demands, then $\mathrm{OPT}(L) \geq \chi$.

- PROOF: If two large demands share any edge, they can't be given the same color, because the total load on the edge is more than c.
- Consider the demand d for which the last color χ was opened. Since d could not be assigned any one of the first $\chi - 1$ colors, there are $\chi - 1$ large demands, one for each color, which shared an edge with d. Since the demands have been considered in a left to right manner, all these $\chi - 1$ large demands will pass through the first edge e of d.
- Together with d, there are χ large demands passing through the edge e. Hence, the optimum has to give each of them a separate color, so it will also require at least χ colors. Hence, this algorithm uses the minimum number of colors.

Analysis for large demands



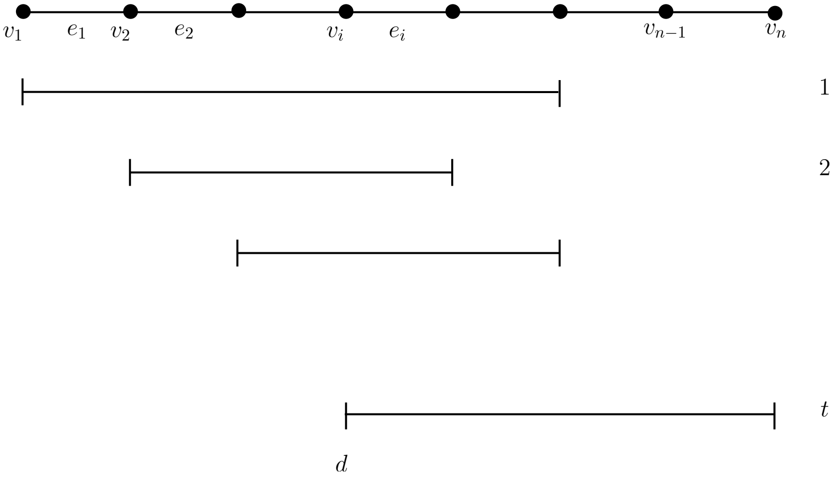
A 2-approximation algorithm for small demands

- The algorithm is the same as that for the large demands.
- Let t be the number of copies of the path P required to assign all the requests in $D_1, \ldots, D_{n-1}$.
- Let l_i be the load on edge e_i.

Lemma

When all the requests in $D_1, \ldots, D_{n-1}$ have been colored, there is an edge e_i such that in at least $t-1$ copies of P, $l_i > \frac{1}{2}c$.

Analysis for small demands



- Consider the demand $d \in D_i$ (for some i) due to which the last color t was opened. At the time d was considered, all the requests started on or before v_i.
- Since d could not be assigned any of the previous $t-1$ colors, there are $t-1$ edges, one for each color, such that the total load put by the existing small demands on each of these edges is strictly more than $c - d \geq \frac{1}{2}c$ since, $d \leq \frac{1}{2}c$.
- Since the demands have been considered in a left to right manner, the load on the first edge e_i of d on each of these $t-1$ colors is at least as much.
- Hence, e_i is the edge such that $l_i > \frac{1}{2}c$.

- The total load put by requests in $D_1, \ldots, D_{n-1}$ on e_i is greater than $\frac{1}{2}c(t-1)$.
- Hence, any packing of these requests (including $\mathrm{OPT(S)}$) will require more than $\frac{1}{2}(t-1)$ copies, since the edge capacity is c.
- Thus, $\mathrm{OPT(S)} > \frac{1}{2}(t-1)$.
- Hence, $t < 2 \cdot \mathrm{OPT(S)} + 1 \leq 2 \cdot \mathrm{OPT(S)}$.
- Since, we can assign all the requests in $D_1, \ldots, D_{n-1}$ using $t \leq 2 \cdot \mathrm{OPT(S)}$ copies, this is a 2-approximation algorithm.

A 3-approximation algorithm

- We solve the instance containing only large demands and the instance containing only small demands separately.
- $\mathrm{ALG}(L) = \mathrm{OPT}(L)$ and $\mathrm{ALG}(S) \leq 2 \cdot \mathrm{OPT}(S)$.
- $\mathrm{OPT} \geq \max\{\mathrm{OPT}(L), \mathrm{OPT}(S)\}$.
- Hence the total number of colors required by the algorithm is

$$\begin{aligned} \mathrm{ALG} &= \mathrm{ALG}(L) + \mathrm{ALG}(S) \\ &\leq \mathrm{OPT}(L) + 2 \cdot \mathrm{OPT}(S) \\ &\leq 3 \cdot \mathrm{OPT}. \end{aligned}$$

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.
- 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.

- For large demands, round down capacity of every edge to the nearest multiple of $c_{\min}$.
- This will increase the congestion r by a factor of 2.
- Round up every demand to $c_{\min}$. Note that for any large demand, $d_i > \frac{1}{4}b_i \geq \frac{1}{4}c_{\min}$.
- Moreover, $d_i \leq c_{\min}$ because of NBA.
- This will increase the congestion r by a factor of 4.
- The resulting instance has uniform demands, which can be colored with r colors. So, large demands require $8r$ colors.
- 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.

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

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	$(0, \frac{1}{4}]$	$(\frac{1}{4}, 1]$	1	1
1	$(0, \frac{1}{4}]$	$(\frac{1}{4}, 1]$	2	1
2	$(0, \frac{1}{2}]$	$(\frac{1}{2}, 1]$	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 $\frac{1}{4}kc$.
- 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.

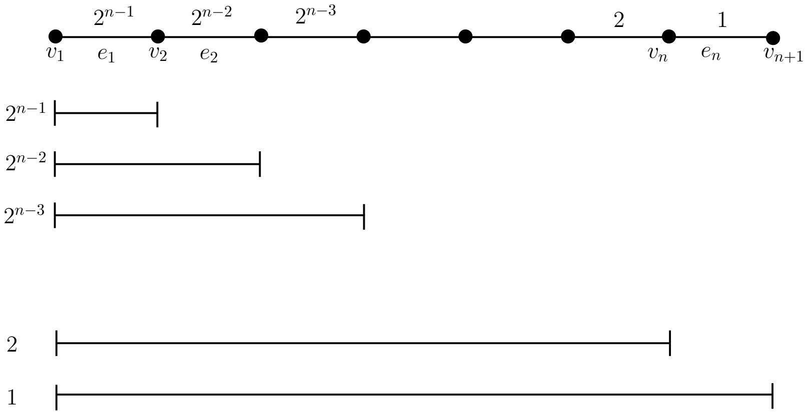
An online algorithm for $\frac{1}{4}$-small demands

$k \leftarrow 1$;
while *there are still requests in the input* **do**
  let R be the next request;
  while *for any edge* $e \in R$, $l_e\left(\left(\bigcup_{i=1}^{k} S_i\right) \cup \{R\}\right) > \frac{1}{4}kc$ **do**
    `//` e `is called a critical edge of` R `on level` k`.`
    $k \leftarrow k+1$;
  end
  $S_k \leftarrow S_k \cup \{R\}$;
  give R the lowest numbered color not used in any sets $S_1, \ldots, S_{k-1}$ and consistent with S_k;
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?



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

Conclusion

- In this talk, we presented several algorithms for solving various instances of the Round-UFP problem.
- We saw that some special cases of this problem can have much better algorithms.
- We also showed how an algorithm for Round-UFP can be used to solve the Max-UFP and Bag-UFP problems.
- The idea of convex decomposition of fractional LP solutions is useful for this.
- This gives a unified framework for solving these problems.
- We also gave a constant competitive algorithm for the Online Interval Coloring problem.

Future work

- Can we improve the approximation factor for uniform capacities from 3 to 2?
- Is there a unified algorithm for ROUND-UFP for all cases?
- 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 $(7+\epsilon)$-approximation is known.
- Is there a better constant factor competitive algorithm for the ONLINE INTERVAL COLORING problem?
- What is the hardness of approximation of these problems?