CS340 - Analysis of Algorithms

Network Flow IV
Extensions to Network Flow

Announcements:

HW8 Deadline Extended

Due Wednesday November 26

Lab8 Due Tomorrow

Final Exam Options:

1. Dec 12 1-4pm Park 230
2. Dec 15 9:30-12:30 Park 159

Agenda

1. Project Presentations
2. Extensions to Network Flow
    a. Multiple sources and sinks (Circulations with Demands)
    b. Lower bounds on edges (Capacity bounds)

Registrar’s Problem

By Roman Gergun and Chang Sun

Introduction

Pseudocode

Phase 1

```
// Phase 1: Build conflict graph
// conflictGraph[c][c'] = number of students wanting both c and c'
```

Initialize $conflictGraph[c][c'] \leftarrow 0$ for all $c, c' \in C$;

foreach *student* s *in* S **do**

  foreach *pair* (c_i, c_j) *where* $i < j$ *in* s*'s preferences* **do**

    $conflictGraph[c_i][c_j] \leftarrow conflictGraph[c_i][c_j] + 1;$

    $conflictGraph[c_j][c_i] \leftarrow conflictGraph[c_j][c_i] + 1;$

  end

end

Phase 2

```
// Phase 2: Prioritize courses
// Priority = enrollment count + total conflict intensity
```

foreach *course c in C* **do**

$enrolled(c) \leftarrow \{\text{students who prefer } c\};$

$conflictIntensity(c) \leftarrow \sum_{c' \in C, c' \neq c} conflictGraph[c][c'];$

$priority[c] \leftarrow |enrolled(c)| + conflictIntensity(c);$

end

Sort courses by *priority* (descending);

Phase 3

```
// Phase 3: Schedule courses
foreach course c in sorted order do
    // Calculate load for each time slot
    foreach slot t in T do
        load[t] ← 0;
        foreach course c′ already scheduled at slot t do
            load[t] ← load[t] + conflictGraph[c][c′];
        end
    end
    Sort slots by load (ascending);
    teacher ← teacher assigned to c;
```

Phase 3

```
foreach slot t in sorted order do
    if teacher not busy at t then
        foreach room r in R do
            if r not occupied at t and capacity(r) ≥ |enrolled(c)| then
                Assign c to slot t and room r;
                Mark teacher busy at t;
                Mark room r occupied at t;
                break;
            end
        end
    end
    if c is scheduled then
        break;
    end
end
```

Phase 4

```
// Phase 4: Assign students
enrollments ← 0;
foreach student s in S (in input order) do
    Initialize studentSchedule[s][t] ← empty for all t ∈ T;
    foreach course c in s's preference list (in order) do
        if c is scheduled then
            t ← time slot of c;
            r ← room of c;
            if studentSchedule[s][t] is empty and room r has capacity then
                Enroll s in c;
                studentSchedule[s][t] ← c;
                enrollments ← enrollments + 1;
            end
        end
    end
end
return enrollments;
```

3

Time Analysis

Phase 1

```
// Phase 1: Build conflict graph
// conflictGraph[c][c'] = number of students wanting both c and c'
```

Initialize $conflictGraph[c][c'] \leftarrow 0$ for all $c, c' \in C$;
foreach *student s in S* **do**
  foreach *pair* (c_i, c_j) *where* $i < j$ *in s's preferences* **do**
    $conflictGraph[c_i][c_j] \leftarrow conflictGraph[c_i][c_j] + 1;$
    $conflictGraph[c_j][c_i] \leftarrow conflictGraph[c_j][c_i] + 1;$
  end
end

$$O(s \cdot k^2) = O(s)$$

Phase 2

```
// Phase 2: Prioritize courses
// Priority = enrollment count + total conflict intensity
```

foreach *course c in C* **do**

$enrolled(c) \leftarrow \{\text{students who prefer } c\};$

$conflictIntensity(c) \leftarrow \sum_{c' \in C, c' \neq c} conflictGraph[c][c'];$

$priority[c] \leftarrow |enrolled(c)| + conflictIntensity(c);$

end

Sort courses by *priority* (descending);

$$O(c^2 + c \log c) = O(c^2)$$

Phase 3

```
// Phase 3: Schedule courses
foreach course c in sorted order do
    // Calculate load for each time slot
    foreach slot t in T do
        load[t] ← 0;
        foreach course c′ already scheduled at slot t do
            load[t] ← load[t] + conflictGraph[c][c′];
        end
    end
    Sort slots by load (ascending);
    teacher ← teacher assigned to c;
```

$$O(c \cdot (c + t \log t + t \cdot r))$$

Phase 3

```
foreach slot t in sorted order do
    if teacher not busy at t then
        foreach room r in R do
            if r not occupied at t and capacity(r) ≥ |enrolled(c)| then
                Assign c to slot t and room r;
                Mark teacher busy at t;
                Mark room r occupied at t;
                break;
            end
        end
    end
    if c is scheduled then
        break;
    end
end
```

$$O(c \cdot (c + t \log t + t \cdot r))$$

Phase 4

```
// Phase 4: Assign students
enrollments ← 0;
foreach student s in S (in input order) do
    Initialize studentSchedule[s][t] ← empty for all t ∈ T;
    foreach course c in s's preference list (in order) do
        if c is scheduled then
            t ← time slot of c;
            r ← room of c;
            if studentSchedule[s][t] is empty and room r has capacity then
                Enroll s in c;
                studentSchedule[s][t] ← c;
                enrollments ← enrollments + 1;
            end
        end
    end
end
return enrollments;
```

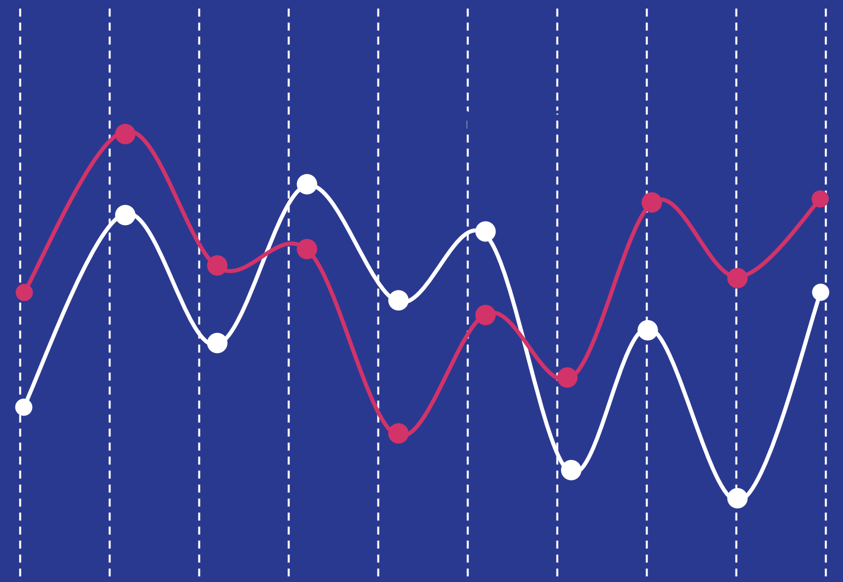
3

$$O(s \cdot k) = O(s)$$

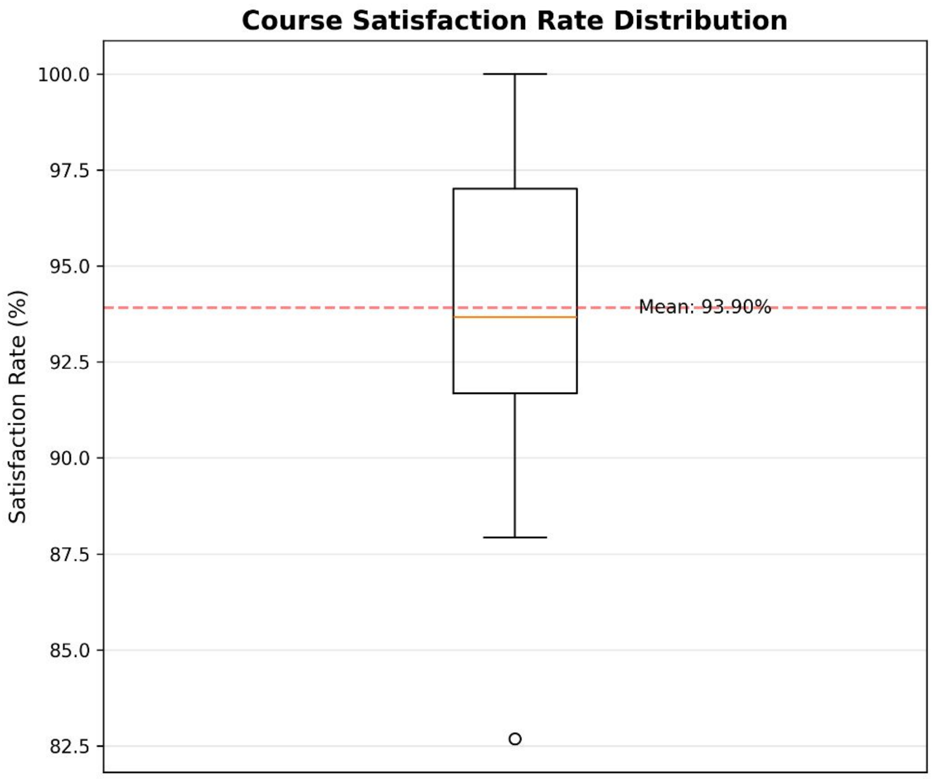
Overall Complexity

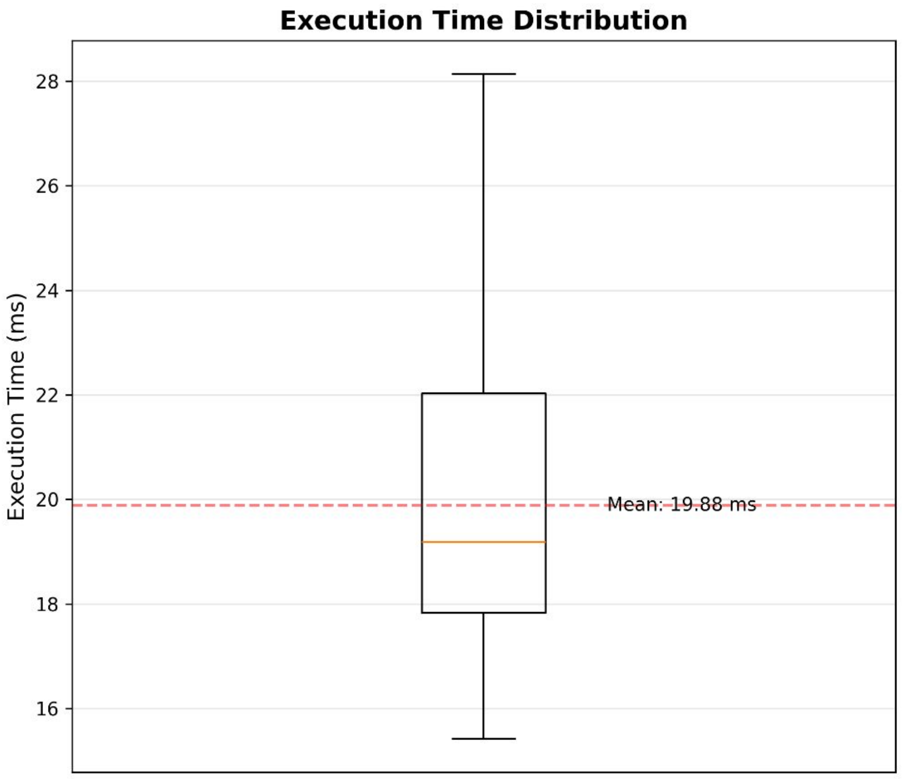
$$O(s) + O(c^2) + O(c^2 + c \cdot t \log t + c \cdot t \cdot r) + O(s) = O(s + c^2 + c \cdot t \log t + c \cdot t \cdot r)$$

Implementation

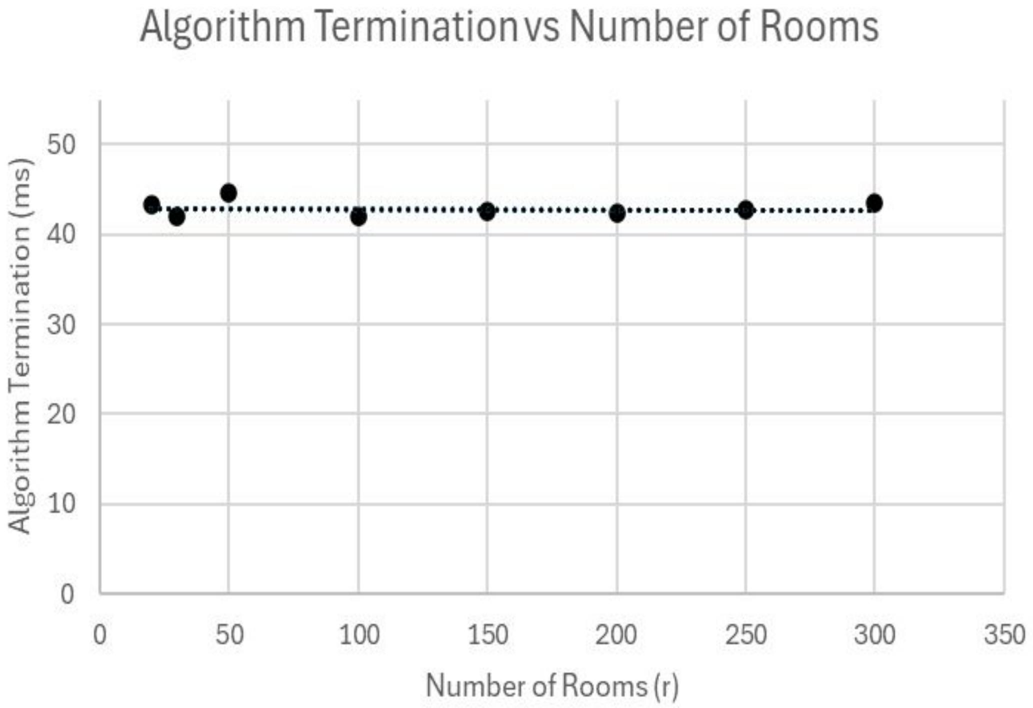


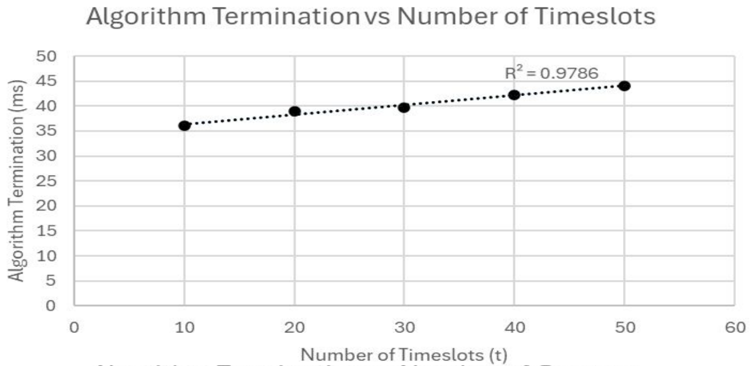
Results on BMC data

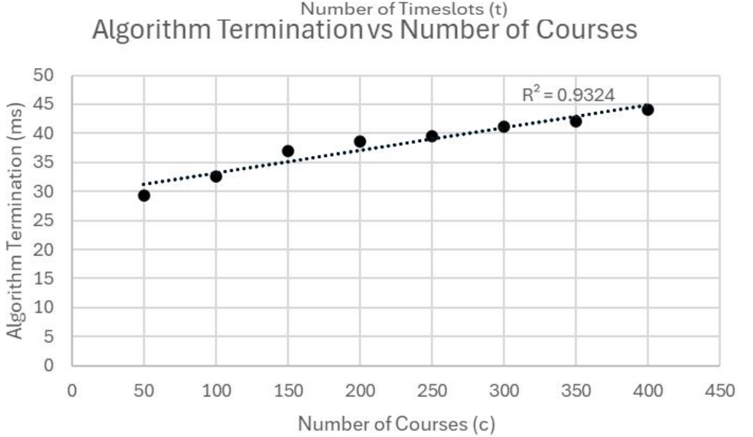




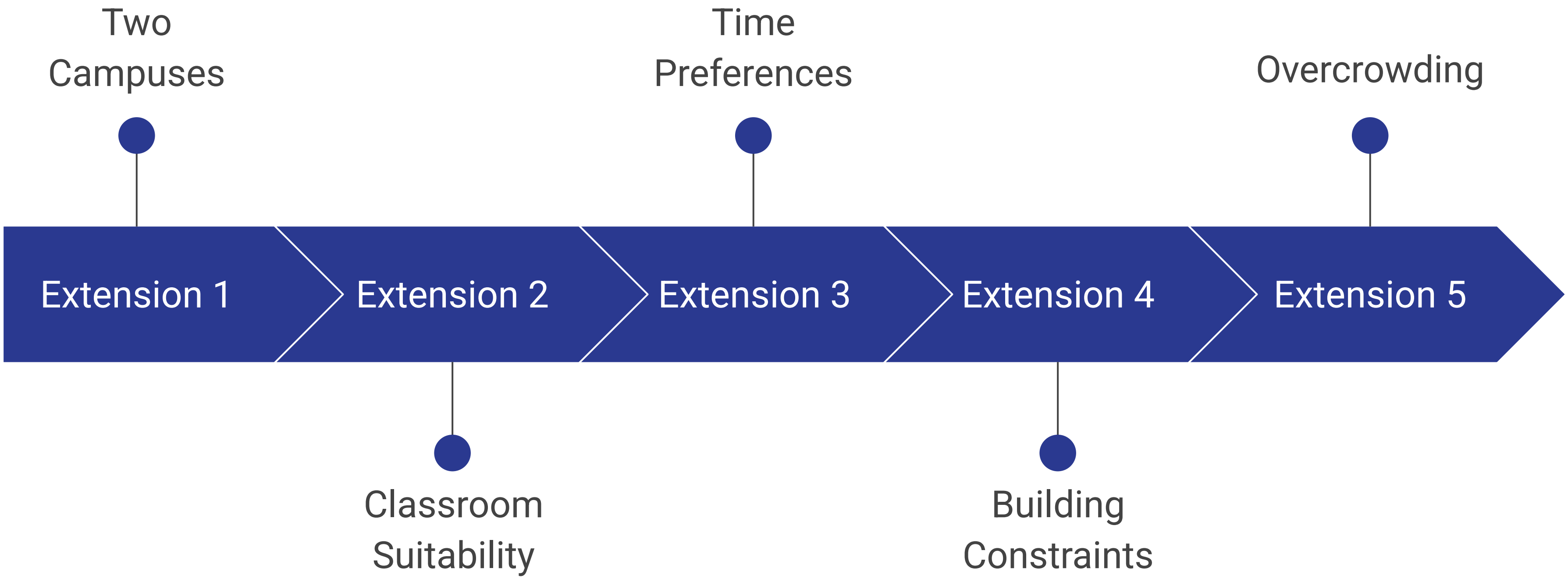
Synthetic Testing on Realistic Values







Extensions and Recommendations


Two Campuses
Time Preferences
Overcrowding
Extension 1
Extension 2
Extension 3
Extension 4
Extension 5
Classroom Suitability
Building Constraints

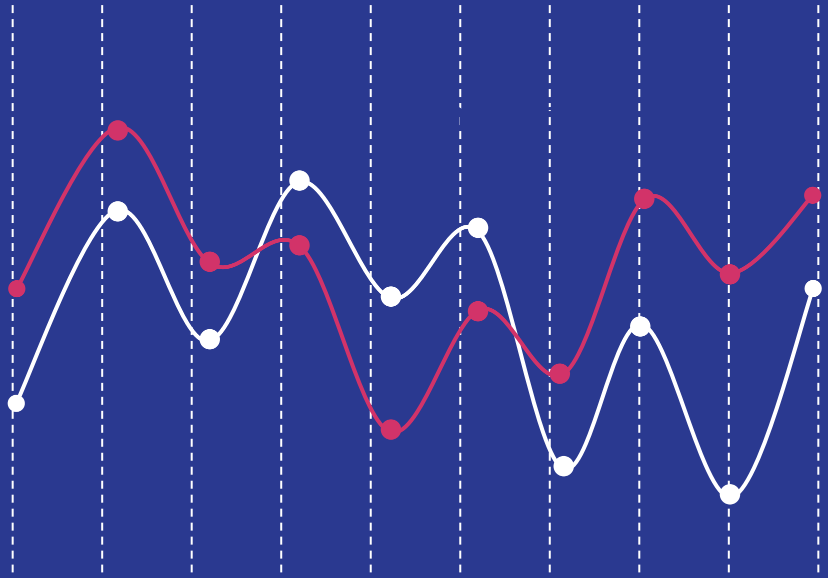
foreach *course* c' *already scheduled at slot* t **do**

$load[t] \leftarrow load[t] + conflictGraph[c][c'];$

end

$$load[t] \leftarrow \sum_{c' \text{ at } t} conflictGraph[c][c'] + travelPenalty(c, t) + preferencePenalty(c, t) + balancePenalty(t).$$

Recommendations



Summarized Recommendations

Algorithm

Conflict Graph Promotes

- Efficiency
- Scalability
- Maintainability

Preferences

Consideration Improves

- Satisfaction
- Conflict Quantity
- Productivity

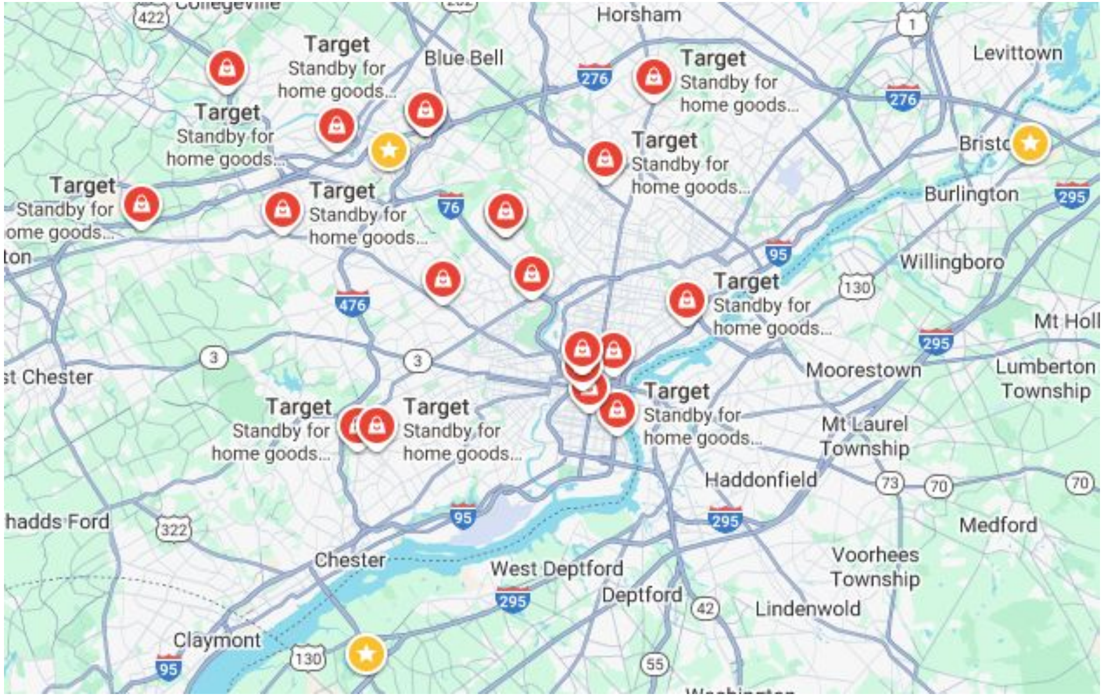
Travel

Incorporation Increases

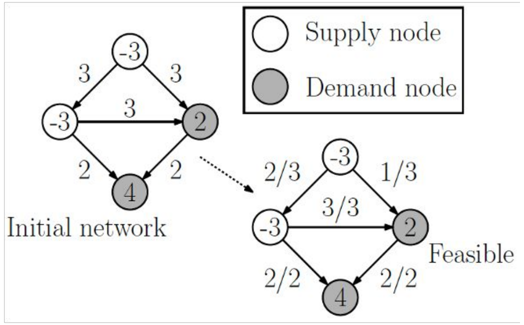
- Opportunity
- Realism
- Diversity of Thought

Thank You!

Circulations with Demands



Circulations with Demands

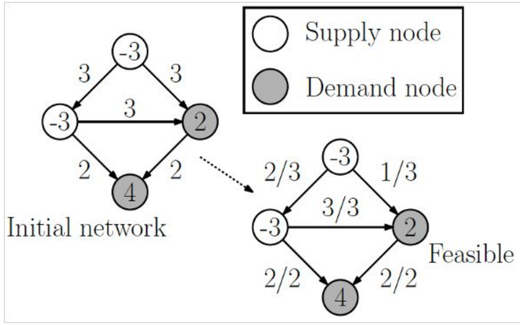


- Flow network G = (V E) with capacities on the edges
- Set of supply nodes S
- Set of demand nodes T
- Each node $v \in V$ has an associated demand value d_v
- If $d_v < 0$, v is a supply node with supply $-d_v$
    - A supply node may have incoming edges, but it must be compensated by the flow that leaves the node on outgoing edges
- If $d_v > 0$, v is a demand node
    - A demand node may have outgoing edges, but it must be compensated by the flow that leaves the node on incoming edges
- If $d_v = 0$, then the node v is neither a source nor a sink
- All capacities and demands are integers

Circulations

- A circulation is a function f that assigns a number to each edge and satisfies the following two conditions:
  - Capacity: For each $(u,v) \in E$, $0 \leq f(u,v) \leq c(u,v)$
  - Demand: For each $v \in V$, $f^{in}(v) - f^{out}(v) = d_v$
- If a vertex is neither in S or T, then $d_v = 0$ and $f^{in}(v) = f^{out}(v)$ (conservation)
- Decision / Feasibility problem: Does there exist a circulation that meets capacity and demand conditions?
- If there exists a feasible circulation, the total supply must equal the total demand

$$D = \sum_{v \in T} d_v = -\sum_{v \in S} d_v \Rightarrow \sum_{v \in V} d_v = 0$$



An Algorithm for Circulations

- We can reduce the problem of finding a feasible circulation with demands to the problem of finding a maximum s-t flow in a different network
- Create a new network G’ = (V’, E’) that has all the same vertices and edges as G
- Add to V’ a super-source s* and a super-sink t*
- For each supply node v ∈ S, add a new edge (s*, v) of capacity $-d_v$
- For each demand node u ∈ T, add a new edge (u, t*) of capacity d_v

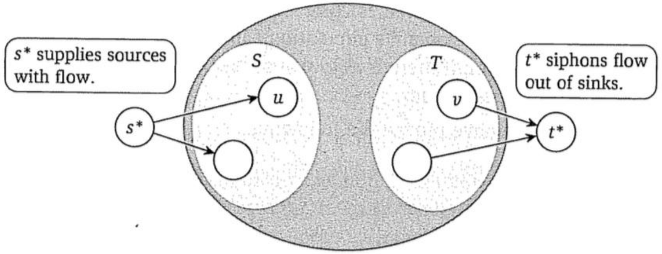
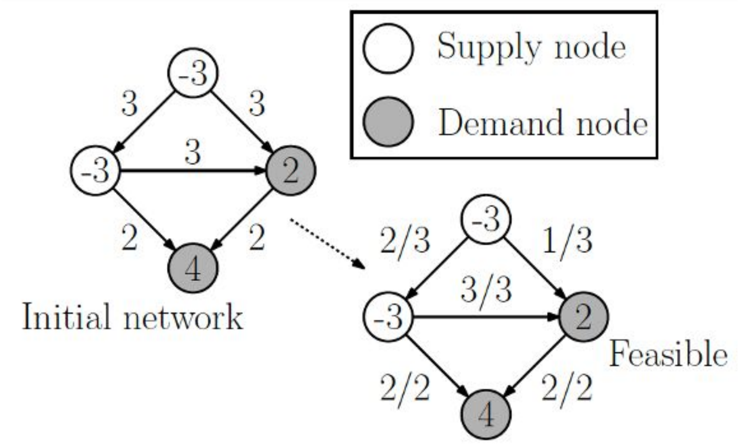
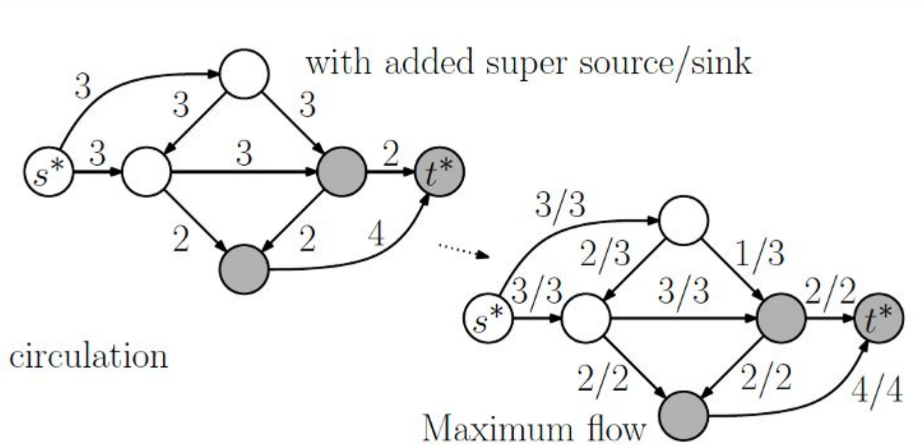


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.

Reduction to Network Flow

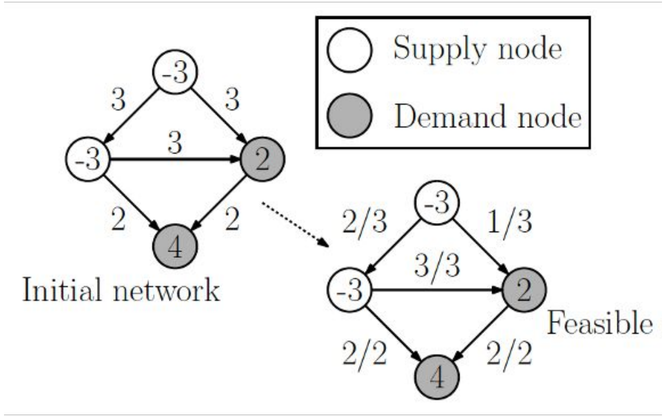


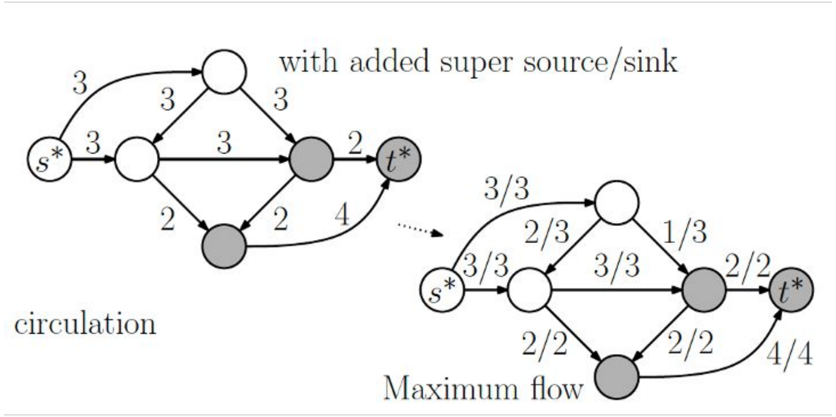


Reduction to Network Flow

<u>Lemma:</u> There cannot be an s*-t* flow in G’ of value greater than D

- Consider a cut (A,B) with A = {S*} and B = V\S*
- The cut capacity = total demand D





Time Analysis

- |V’|?
  a. n+2 = O(n)
- |E’|?
  a. m+|S|+|T|
  b. |S| < n, |T| < n
  c. O(m+n)
- Graph construction time?
  a. O(2n+m) = O(n+m)
- Floyd Fulkerson runtime:O(Cm)
  a. What is C for us?
     i. D (total demand out of S)
  b. m = O(m+n)
  c. O(D(m+n))

Network Flow Reductions

General advice:

1. Reduction Formulation:
   a. Describe how to build flow network $G' = (V', E')$
      i. Specify vertices, edges, and edge capacities
   b. Describe how the maxflow value $|f|$ of G' relates to the solution to the original problem
2. Time Analysis:
   a. Consider how the problem size grows when we reduce to Network Flow
      i. Compute n' and m'
   b. Consider the reduction graph construction time
   c. Consider total modified runtime of FF
3. **Correctness of the reduction:**
   a. If and only if equivalence proof
   b. $|f|$ on $G' \geq$ solving original problem
   c. $|f|$ on $G' \leq$ solving original problem

Correctness of the Reduction

<u>Claim</u>: There is a feasible circulation with demands in G iff the maximum s*-t* flow in G' has value D

1. $|f|$ on G' ≥ feasible circulation with demands
2. $|f|$ on G' ≤ feasible circulation with demands

Here we are checking the *existence* of a feasible circulation so it doesn't have a value. We can't compare greater than or less than with a max-flow value.

We need to prove equality in both directions

Correctness of the Reduction

<u>Claim</u>: There is a feasible circulation with demands in G iff the maximum s*-t* flow in G' has value D

Direction 1: Start with a feasible circulation f and transform it to flow

Suppose G has a feasible circulation f

Construct a flow f' in G with f'(u,v) = f(u,v) for all (u,v) $\in$ E

Saturate the edges from s* with f'(s*, s) = $-d_s$ for all s $\in$ S

Saturate the edges to t* with f'(t, t*) = d_t for all t $\in$ T

f' is a valid flow in G'. It satisfies capacity constraints as f(u,v) satisfies capacity for (u,v) $\in$ E in G and the capacity on those edges is the same in G'. For f'(s*, s) and f'(t, t*), these also satisfy capacity constraints as the capacity of these edges is the demand.

f' satisfies conservation constraints. Forall (u,v) in E, conservation is satisfied. For source nodes, conservation is satisfied. For demand nodes, conservation is satisfied.

|f'| on G' = D

Correctness of the Reduction

<u>Claim</u>: There is a feasible circulation with demands in G iff the maximum s*-t* flow in G’ has value D

Direction 2: Start with a flow in G’ and transform it into a feasible circulation in G

Let f' be a maximum s*-t* flow in G' with |f'| = D.

For each (u,v) ∈ E, f(u,v) = f’(u,v)

f is a valid circulation. It satisfies capacity constraints because the edges (u,v) in G’ that exist in G have the same capacity. Since f is a valid flow, these satisfy capacity constraints.

f satisfies demand constraints.

A flow of size |f’| = D can be transformed to a feasible circulation in G.

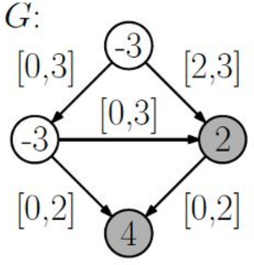
Extension #2

Capacity Lower Bounds

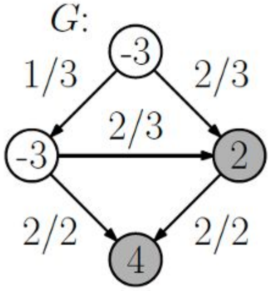
Circulations with Demands and **Lower Bounds**

- Add minimum capacity constraints *l(u, v)*
- Given a network G = (V, E)

- Each edge (u, v) ∈ E, $l(u,v) \le f(u,v) \le c(u,v)$
  - Capacity constraint
- For each vertex u ∈ V, $f^{in}(v) - f^{out}(v) = d_v$
  - Demand constraint
- Is there a feasible circulation?



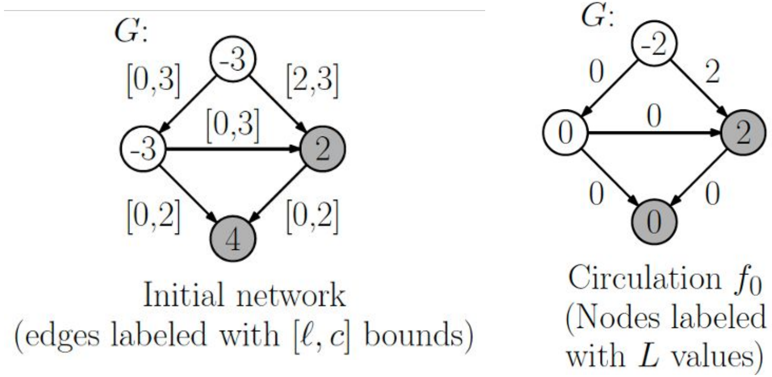
Initial network
(edges labeled with $[\ell, c]$ bounds)



A valid flow

Reduction to Circulations with Demands

- Generate an initial *pseudo* circulation f_0 that satisfies exactly the lower bounds $f_0(u,v) = l(u,v)$



- Satisfies all capacity conditions, but may not satisfy all demand conditions

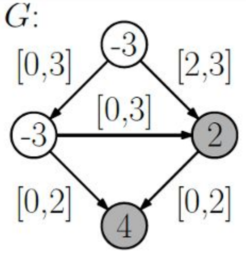
$$L_v = f_0^{in}(v) - f_0^{out}(v) = \sum_{(u,v)\in E} l(u,v) - \sum_{(v,w)\in E} l(v,w)$$

Reduction to Circulations with Demands

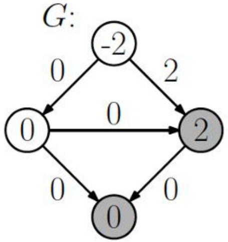
- For each node,
  - If $L_v = d_v$ then we have satisfied the demand condition at v
  - If not, we need to add more flow
- Superimpose a circulation f_1 on top of f_0 to meet demand
- What should this additional flow f_1 look like?
  - $f_1^{in}(v) - f_1^{out}(v) = d_v - L_v$
- We also need to make sure we don't add too much and break capacity constraint
  - Remaining capacity is c(u,v) - l(u,v)

Reduction to Circulations with Demands

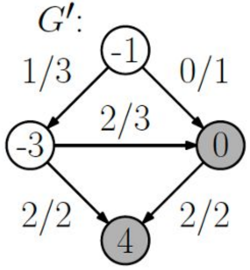
- Construct a new graph G' with the same nodes and edges, with capacities and demands, but with no lower bounds
- $c(u',v') = c(u,v) - l(u,v)$
- $d'_v = d_v - L_v$
- Compute standard circulation on G'
  - No valid circulation on G' = no valid circulation on G
  - Valid circulation f_1, combine f_1 with f_0



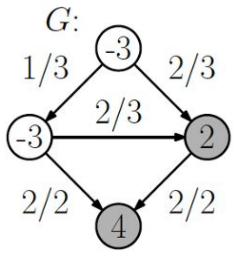
Initial network
(edges labeled with $[\ell, c]$ bounds)



Circulation f_0
(Nodes labeled with L values)



Valid circulation f_1



Final circulation
(for original network)

Correctness of the Reduction

Claim: The network G (with lower bounds) has a feasible circulation iff G' has a feasible circulation

Direction 1: Given a circulation with lower bounds, show you can produce a feasible circulation on G'

Suppose there is a circulation f in G.

Define a circulation f' in G' as $f'(u,v) = f(u,v) - l(u,v)$

This is a valid circulation because

1. it satisfies capacity constraints: $f'(u,v) \leq c'(u,v)$
2. It satisfies demand constraints: $f'^{in}(v) - f'^{out}(v) = d'_v$

Correctness of the Reduction

Claim: The network G (with lower bounds) has a feasible circulation iff G' has a feasible circulation

Direction 2: Given a circulation on G', show you can produce a feasible circulation with lower bounds on G

Suppose there is a circulation f' in G'.

Define a circulation f in G by $f(u,v) = f'(u,v) + l(u,v)$

This is a valid circulation because

1. it satisfies capacity constraints $l(u,v) \leq f(u,v) \leq c(u,v)$
2. It satisfies demand constraints: $f^{in}(v) - f^{out}(v) = d_v$

Extension #3

Survey Design

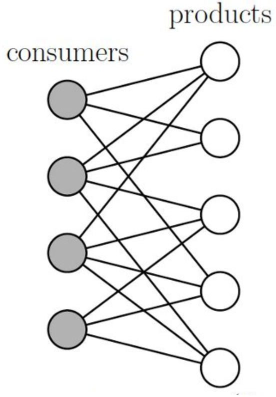
Survey Design

- A company sells k products and maintains a database with purchase history
- The company wishes to conduct a survey sending customized questionnaires to a particular group of n of its customers
    - Survey will only ask about products the customer has purchased
    - Ask customer i about at least c_i products but no more than c_i'
    - Ask at least p_j and at most p'_j customers about product j

Survey Design

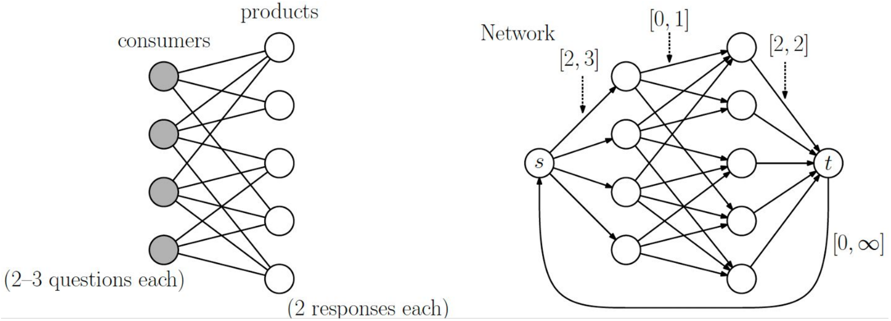
- We can represent the input to this problem as a bipartite graph G
- Nodes are customers and products
- There exists an edge between customer i and product j if they have ever purchased j

- We also have constraints:
  - Lower and upper bounds on number of questions per customer
  - Lower and upper bounds on number of questions about each product



Reduction to Circulation with Lower Bounds

- Create a graph G'
  - V' = V + {s, t}
  - Add a directed edge (i,j) from each customer i to product j that they purchased
    - With lower bound 0 and upper bound 1
  - Add edges (s,i) from s to each customer node
    - With lower bound c_i and upper bound c'_i
  - Add edges (j,t) from each product node to t
    - With lower bound p_j and upper bound p'_j
  - Add an edge (t,s) with [0,inf]



- Set all demands to 0

Survey Design - Proof of Correctness

Claim: The graph G' has a feasible circulation iff there is a feasible way to design the survey

Direction 1: Given a feasible circulation in G' transform it into a valid survey

Suppose we have a feasible circulation f in G'

Customer i should be asked about product j iff there is a flow across edge (i,j)

This is a valid survey because flow respects

- Lower and upper bounds on number of questions per customer
  - $l(s,i) \leq f(s,i) \leq c(s,i)$ where $l(s,i) = c_i$ and $c(s,i) = c'_i$
  - $f^{in}(i) = f^{out}(i)$
- Lower and upper bounds on number of questions about each product
  - $l(j,t) \leq f(j,t) \leq c(j,t)$ where $l(j,t) = p_j$ and $c(j,t) = p'_j$
  - $f^{in}(j) = f^{out}(j)$

Survey Design - Proof of Correctness

Claim: The graph G’ has a feasible circulation iff there is a feasible way to design the survey

Direction 2: Given a valid survey, transform it into a feasible circulation in G’

Suppose we have a valid survey.

The edge (i,j) will carry 1 unit of flow if customer i is asked about product j

The flow on edges (s,i) is the number of questions asked to customer i

The flow on edges (j,t) is the number of customers who were asked about product j

This is a valid flow because:

1. Demand constraint is satisfied
2. Capacity constraints are satisfied

Survey Design - Runtime Analysis

Let c = # of customers and Let p = # of products

$|V'| = c + p + 2$

$|E'| = |E| + c + p$

This was for the circulations with demands!

O(D(m+n))

D?

Total runtime?

Summary

- Many problems can be reduced to Network Flow or
  - Circulations with demands or
  - Circulations with demands with lower bounds

- Lab 8 due tomorrow

- **HW8 due Wednesday**