CS340 - Analysis of Algorithms

Network Flow III
Extensions to Network Flow

Announcements:

HW7 due today

HW8 Released - Due Monday November 24

Final Exam Options:

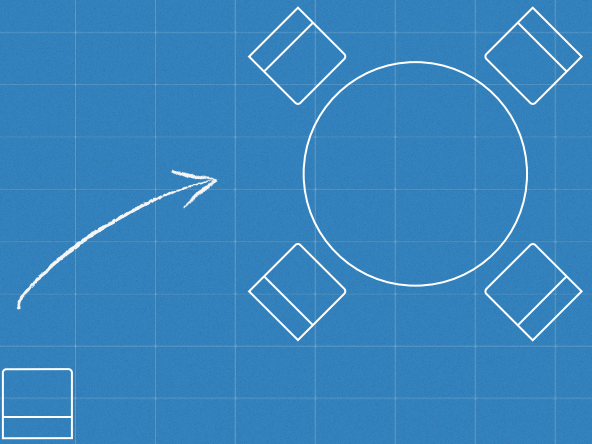
1. Dec 12 1-4pm Park 230
2. Dec 15 9:30-12:30 Park 159

Agenda

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

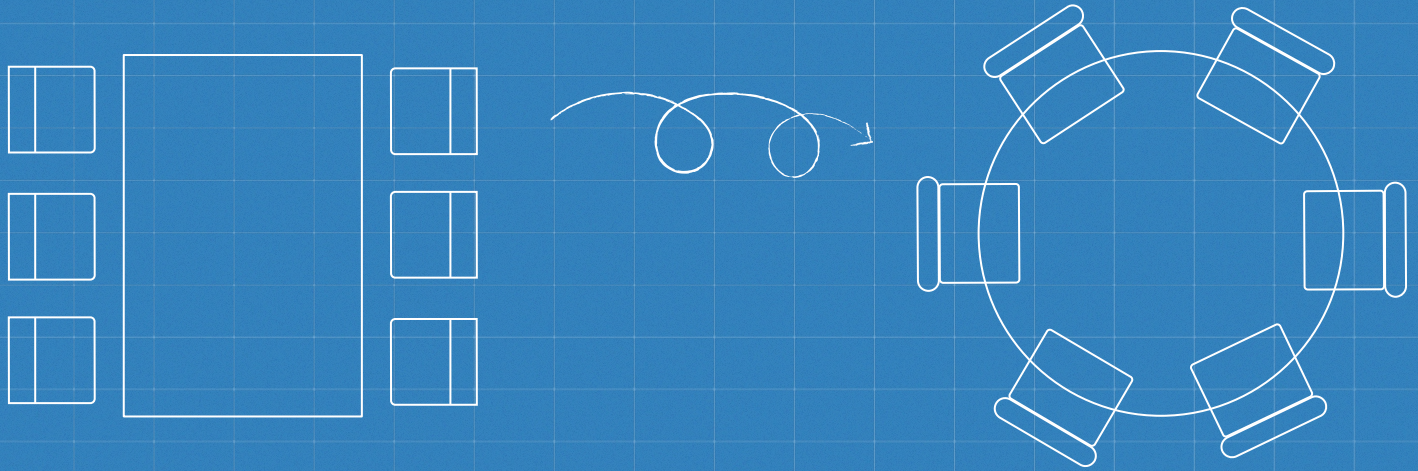
The Registrar's Problem

Project by Camille and Olivia



01

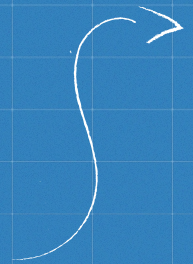
Initial Algorithm



Algorithm Description

Greedy Criteria: Most Popular Class is Scheduled in the Largest Room

- File containing the classes c, class times t, rooms r, and teachers p is read and parsed into respective data structures
- Class popularity is calculated; Classes are sorted from most to least popular and rooms are sorted in descending order based on capacity
- The most popular class is accessed and placed in that room for the first slot, provided the teacher is not scheduled to teach another class at that time.
- Then, c will represent the next class, which will be matched to the next t for that room



- If all classes are scheduled, the loop terminates and if a teacher is busy during the time a class is trying to be scheduled, the class is skipped and c will move on to the next class and assign it the current time slot
- The unassigned class is assigned during the next time slot
- Once all classes are assigned a room and time slot, students are placed into their preferred classes as availability allows
- Completed schedule is written to a file

Time Analysis

Let s = |S|, where S is the set of all students, c = |C| be the set of all classes, r = |R| be the set of all rooms, t = |T | be the set of all time slots, and p = |P | be the set of all professors.

- Algorithm contains four loops: a for-loop iterating through rooms, a nested for-loop iterating through time slots, a further nested while loop iterating through professors, and a separate for-loop iterating through students.
- The nested for-loop of rooms and times will have a complexity of O(rt).
- Since each student has exactly four classes in their preference list, the inner loop of the students loop runs in constant time of O(s).
- Since each professor teaches only two classes, there can only be one time conflict per time-slot already filled. Thus, the number of iterations of the while-loop is O(t).
- Factoring in the time to sort the classes by popularity (clogc) and sort rooms by capacity (rlogr), we get O(r · t + rlogr + s + clogc).

Data Structures

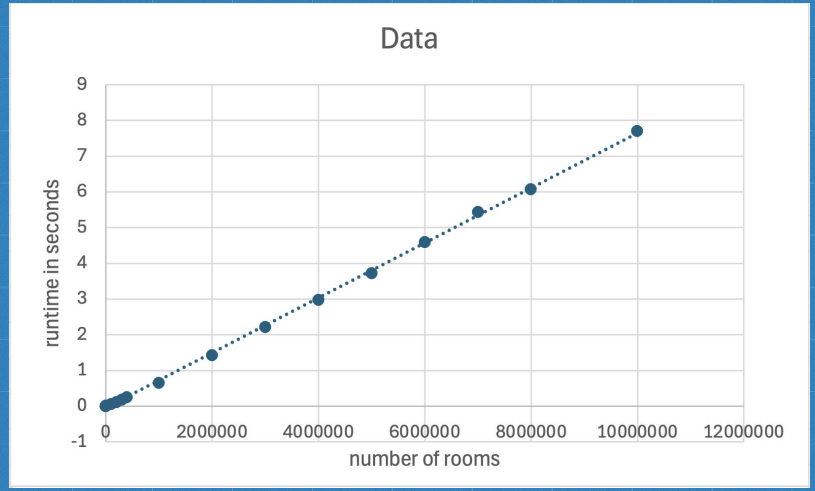
- Class Object Type
- Students stored in a 2D array
  - Building array: O(s)
  - Accessing array: O(1)
- Classes sorted and stored in an array
  - Building sorted array: O(clogc)
  - Accessing array: O(1)
- Rooms sorted and stored in an array
  - Building sorted array: O(rlogr)
- Class Times stored in an array
  - Building array: O(t)
- Final Class Schedule stored in a 2D array
  - Building array: O(r · t)
  - Accessing array: O(1)

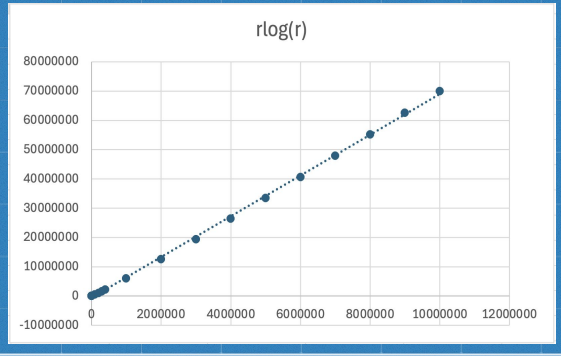
Experimental Analysis

Our Overall Time Complexity was the following:

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

- Dominant Variable is the Number of Rooms
- Tested using make_random_input.pl file
- Increased Room Size and kept the number of courses, class times, and students constant
- Over our many tests, using the best-case student value of 1/1 (100%) as our upper bound, our algorithm is able to achieve at least 3/5th's (60%) of the best-case student value

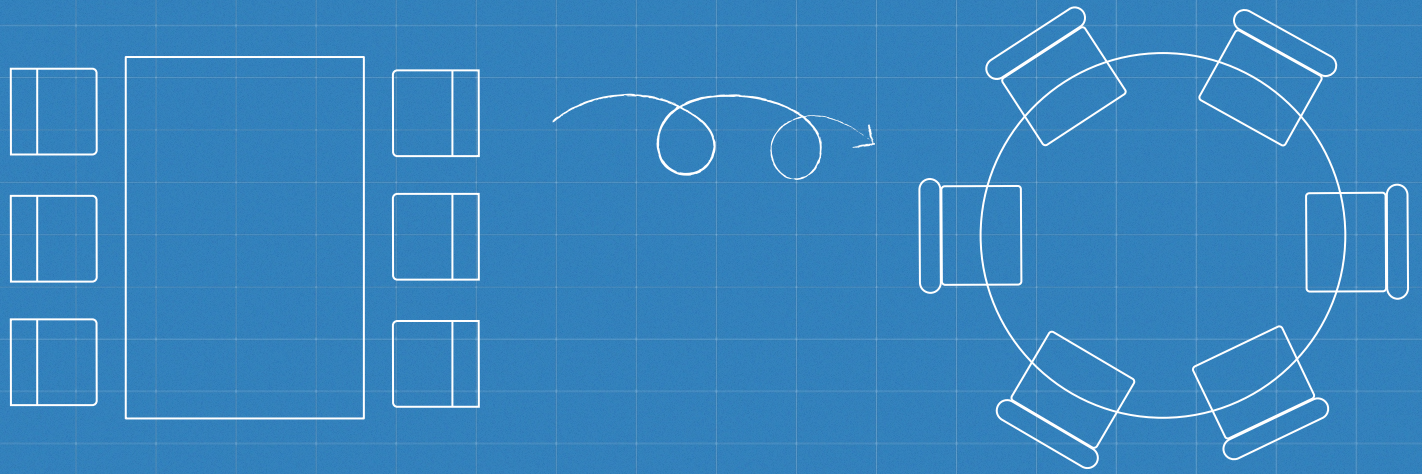




02

Bryn Mawr Scheduling

Creating Schedules Utilizing Real Bryn Mawr Data



Data Handling

BMC Real-World Data

- Used BMC Fall 2000-2004 and Spring 2001-2005 Data to Test
- Times had different formats and specified days
- Times could now conflict
- Length of Student Preference List Varied
- Classes, students, and teachers had id numbers that weren't easily indexable to an array

Changes to Initial Algorithm

- Handles Student Preference List being different sizes
- Added isConflict Function that handles checking if two class times conflict
- Modifying mark array to account for teachers teaching more than two classes

Recommendations, Constraints, and Scenarios

1. Same Major/Same Level Classes Cannot Be Taught at the Same Time

2. Randomly Generate Other Classes in Student's Preference List

3. Marking Missing Professors as TBD and including those courses in the schedule

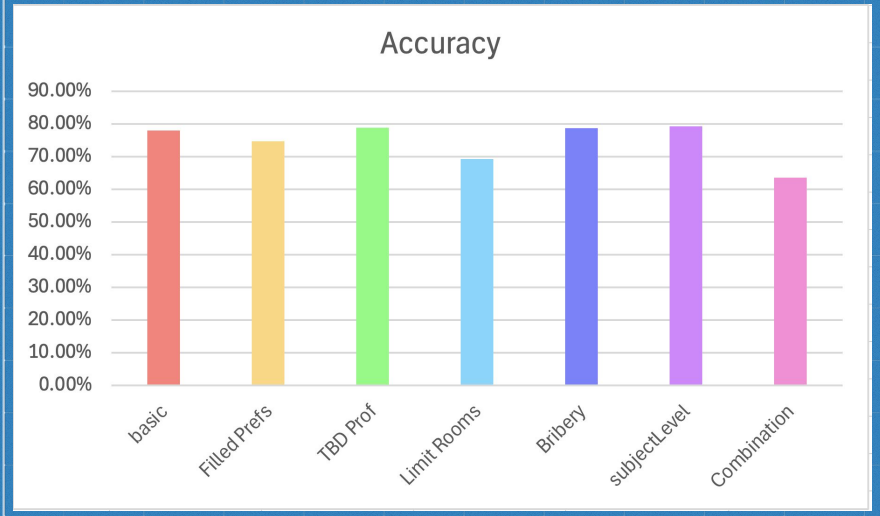
4. Limiting which classrooms a course can be taught it

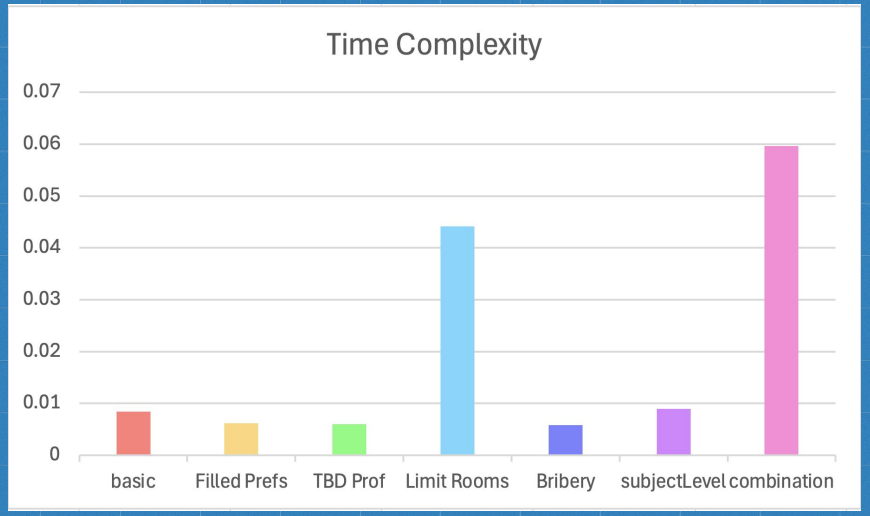
5. Allowing Students to Bribe their way into a Class

Algorithm Modifications

```
    teacher = class's teacher
    while teacher is busy do
        newClass = next class in classes
        if newClass == null then
            skip this time-room combination and go to next time
        end
        teacher = newClass's teacher
    end
    assign class to room and timeSlot
    mark class.teacher as busy
end
```

Experimental Analysis



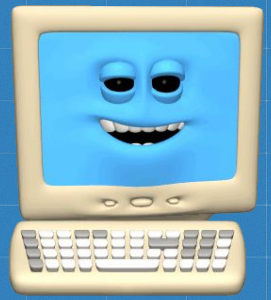


Live Demo of Code

Takeaways and Future Implementations

- Real-World Data was Incomplete
- Better results if we allow Classes to be held in a Wider Variety of Rooms
- Consider factoring in Blue Bus Times and Haverford Schedule when Scheduling Classes

Thank You



The Registrar’s Problem

Lindsey Turner and Isabella Taylor

Problem Overview

Goal: schedule courses to maximize Student Preference Value (SPV)

Constraints:

- One class per (room, time) pair
- Professors can’t teach two classes at once
- Students can’t have overlapping classes
- Room capacity respected

Our Algorithm

Phase A: Assign Classes

- **Compute demand (#students requesting each course)**
- **Sort courses by demand (high→low)**
- **Sort rooms by size (large→small)**
- **Assign each course to first feasible (room, time) slot where room and professor are free**

Phase B: Enroll Students

- For each student (file order):
  - Enroll if course exists, has space, and no time conflict
  - Update capacity & SPV

Pseudocode - Phase A

```
function parse constraints(file) {

    Read rooms, capacities, courses, professors, times

}

function parse prefs(file, courses, times)

    for each student s do {

        Read 4 (v requests for the real data)
        requests, update course demand and
        course requests

        }

}
```

Pseudocode - Phase A

```
function assign classes(courses, rooms, times) {
     Sort courses by demand, rooms by capacity
     for each course c {
          for each room r {
               for each time t {
                    if roomFree[(r, t)] and profFree[(prof (c), t)] {
                         Assign c to (r, t), mark room/prof busy
                         break
                    }
               }
          }
     }
}
```

Our Algorithm

Phase A: Assign Classes

- Compute demand (#students requesting each course)
- Sort courses by demand (high→low)
- Sort rooms by size (large→small)
- Assign each course to first feasible (room, time) slot where room and professor are free

Phase B: Enroll Students

- **For each student (file order):**
  - **Enroll if course exists, has space, and no time conflict**
  - **Update capacity & SPV**

Pseudocode

```
function enroll students(students){
     SPV ← 0
     for each student s {
          for each requested course c(4)
               if c has space and no conflict {
                    Enroll s, update capacity and conflicts, SPV + +
               }
          }
     }
}
```

Pseudocode

```
function write schedule(courses, students, SPV){
    for each course c do {
        Write (course, room, prof, time, enrolled)
    }
    Print SPV , BestCase, Fit
}
```

Validity & Data Structures

Guarantees:

- Room conflict prevented via `roomFree[(r,t)]`
- Professor conflict via `profFree[(p,t)]`
- Student conflict via `hasClassAt[t]`
- Capacity tracked with `roomRemCap[(r,t)]`

Data Structures

- `courses[c]`: {prof, demand, room, time, enrolled[]}
- `roomFree`, `profFree`, `roomRemCap`
- `students`: list of dicts with name, requests & time conflicts

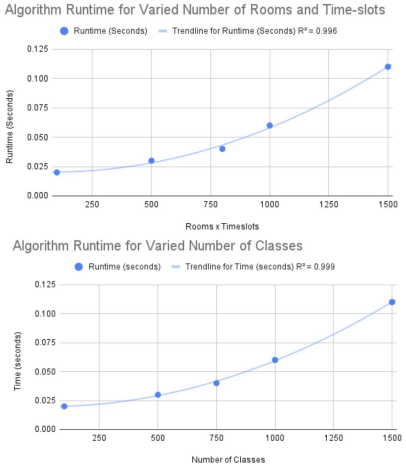
Runtime Analysis

Algorithm runtime

- $T(n) = O(c \cdot r \cdot t + c\log c + r\log r + s)$
  - Scales quadratically with #classes, not #students
  - $\approx O(c \cdot r \cdot t) \approx \mathbf{O(c^2)}$ since $c \approx r \times t$

On Haverford data:

- Runtime ~0.03s



Experimental Results

On random data:

- Fit typically between 0.85–0.90, but hard to say a specific average as there are many different combinations of random inputs/possible constraints.

On real HC data:

- SPV = 2725
- Best Case = 3139
- Fit = 0.868
- Runtime = 0.03s

Real Data Scenarios

We created new script files to modify/create new student preference input files simulating different registrar policy scenarios:

1. Single building per student - **0.913 fit**
2. Same professor per student - **0.811 fit**
3. All students request ≥1 ECON course - **0.866 fit**
4. ≤1 300-level course per student - **0.921 fit**
5. ≥1 auditorium course per student - **0.796 fit**

Key Insights

Highest fit: limiting students to ≤1 300-level course (Scenario 4)

Lowest fit: forcing ≥1 auditorium course (Scenario 5)

Takeaways:

Greedy algorithm still performs well (fit ≥ 0.8 in all cases).

Fit depends more on student request distribution than algorithm complexity.

Structural bottlenecks (large lectures, seminar overloads) are main conflict sources.

Recommendations for the Registrar

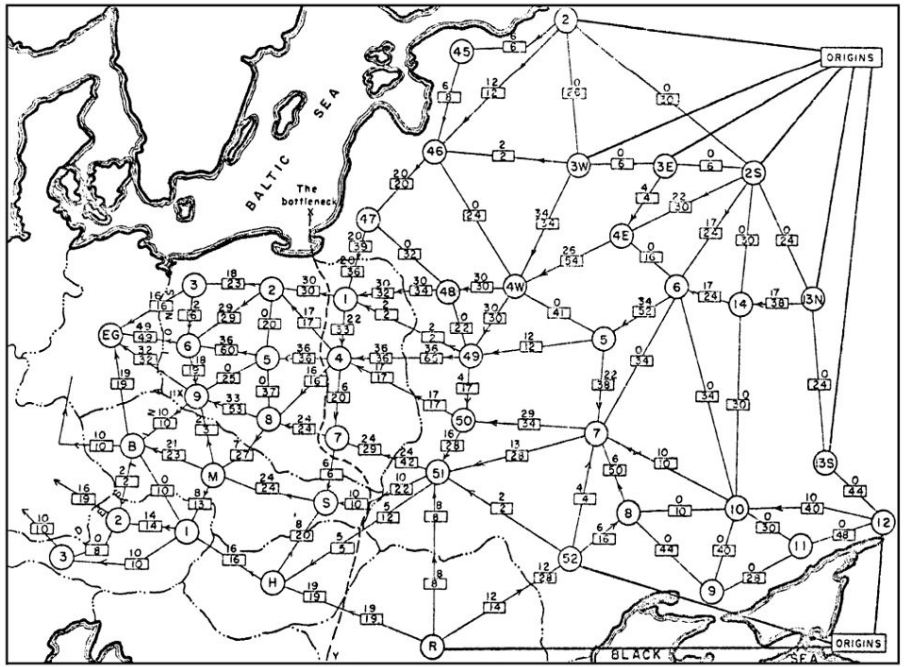
1. **Coordinate 300-level seminars across departments.**
   → Reduce overlap between upper-level seminars.

2. **Diversify large-room scheduling.**
   → Reduce overlap of popular courses

3. **Create “conflict-free bundles.”**
   → Pre-plan common course combinations (e.g., orgo + bio).

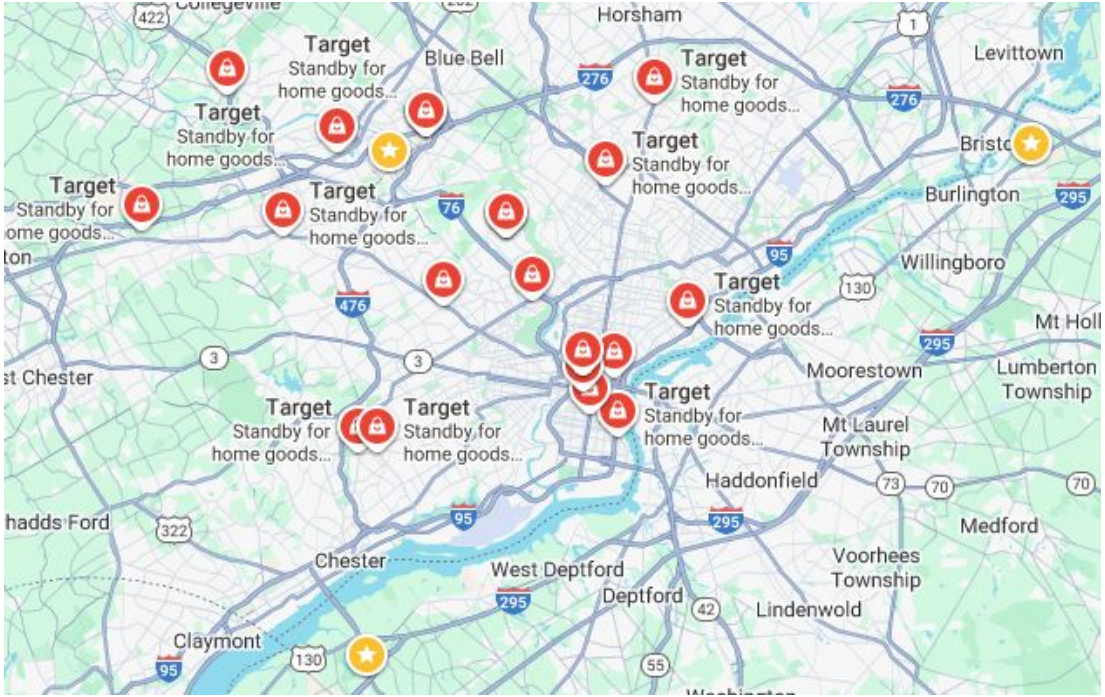
Lab 8 Review

Network Flow and the Cold War



Harris and Ross's map of the Warsaw Pact rail network

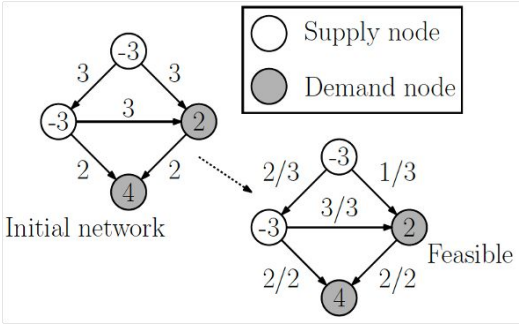
Circulations with Demands



Circulations with Demands

- An extension of the maximum flow problem with multiple sources and sinks
- As before, there is an integer capacity on each edge
- We will not be seeking to maximize an s-t flow value
- Sources have fixed supply values
- Sinks have fixed demand values
- Goal: ship flow from nodes with available supply to those with given demands
- Decision / feasibility problem not a maximization / optimization problem
- Is it possible to get the products from the supply nodes to the demand nodes, subject to the capacity constraints?

Circulations with Demands

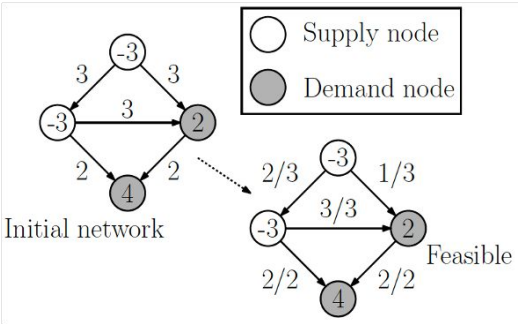


- Flow network G = (V E) with capacities on the edges
- Set of supply nodes S
- Set of demand nodes T
- Each node $v \in V$ has an associated demand value d_v
- If $d_v < 0$, v is a supply node with supply $-d_v$
  - A supply node may have incoming edges, but it must be compensated by the flow that leaves 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
- Decision / feasibility problem not a maximization / optimization problem

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

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



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 $\in$ S, add a new edge (s*, v) of capacity $-d_v$
- For each demand node u $\in$ T, add a new edge (u, t*) of capacity d_v

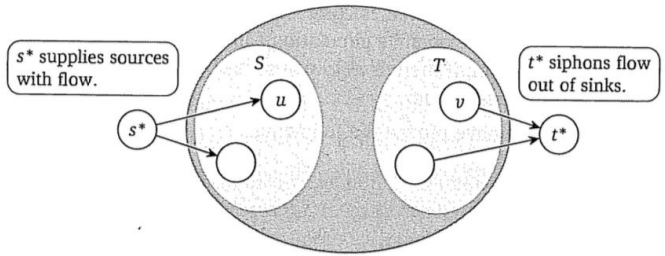
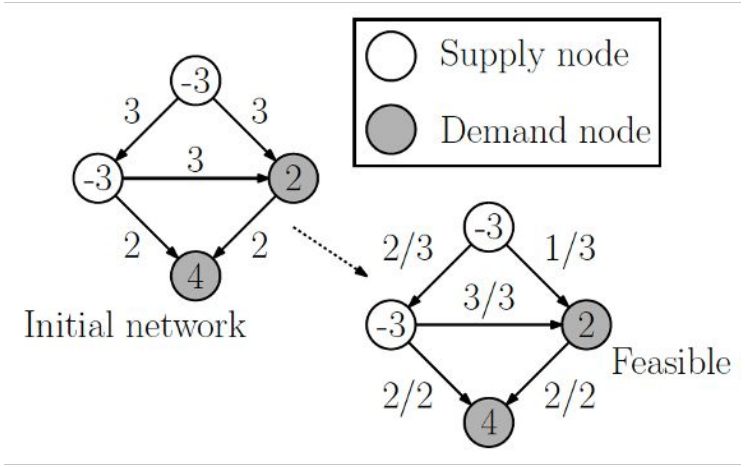
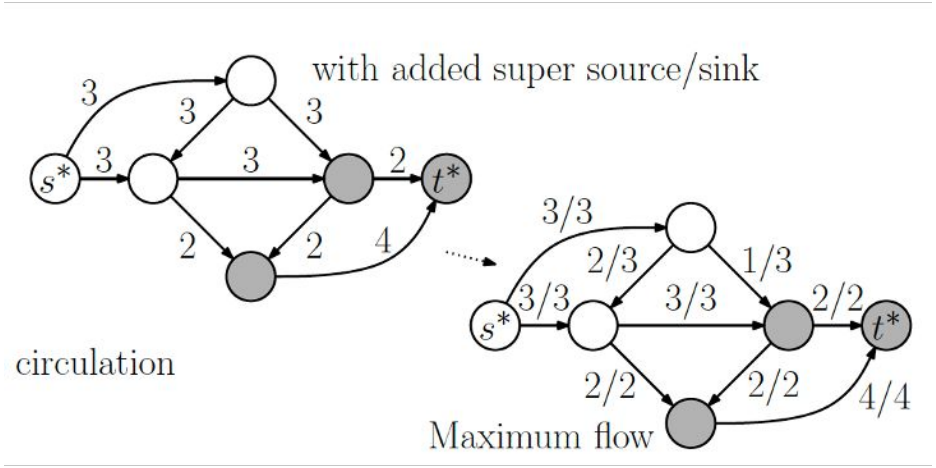


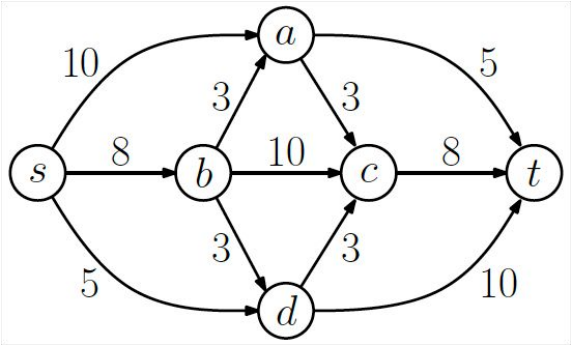
Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.

Reduction to Network Flow





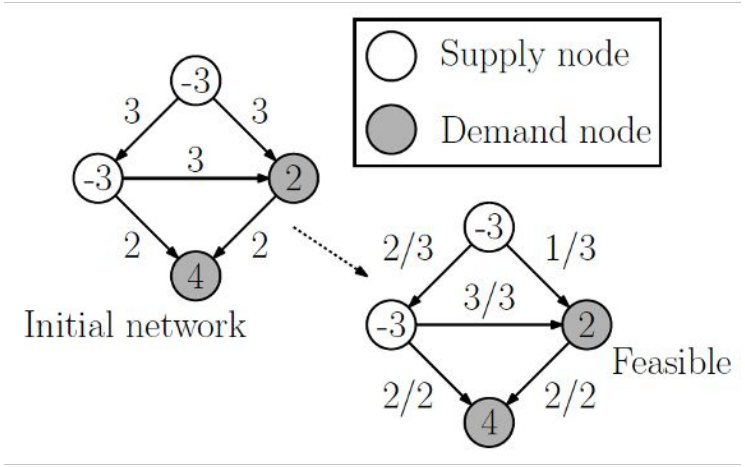
Ford-Fulkerson

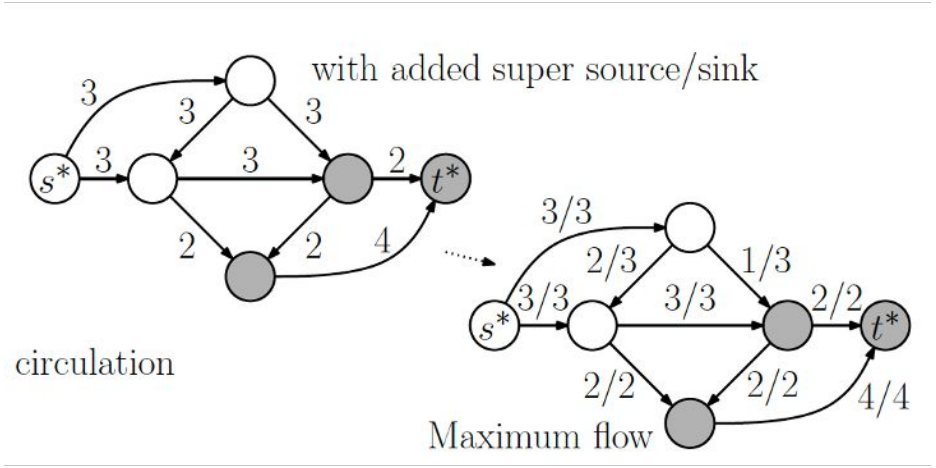



```
ff(G=(V, E, s, t)) {
  f = 0 (all edges carry 0 flow)
  while (true) {
    G' = residual-network of G for f
    if (G' has no s-t augmenting path) break
    P = any-augmenting-path of G'
    c = min capacity of P
    augment by adding c to every edge of P in G
    update f
  }
  return f
}
```

Reduction to Network Flow

- Compute the maximum flow value in G’
- If the |f| ≥ total demand D, then we have found a feasible circulation
- Is there an upper bound on |f| ?

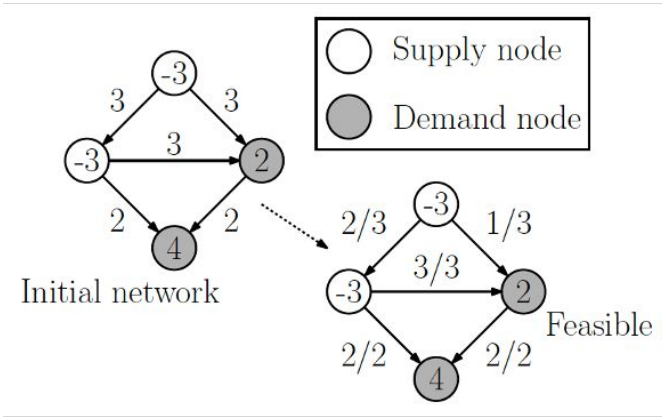


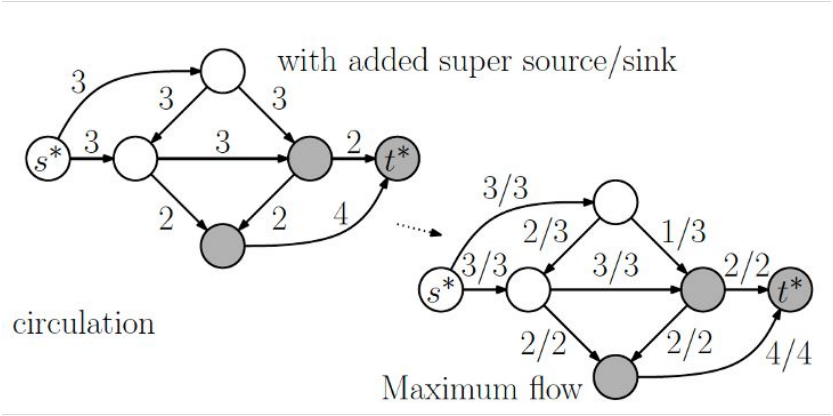


Reduction to Network Flow

Lemma: 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





Reduction to Network Flow

1. If the |f| ≥ total demand D, then we have found a feasible circulation
2. Lemma: There cannot be an s*-t* flow in G' of value greater than D
3. **If |f| = total_demand D, then we have found a feasible connection**

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' ≥ solving original problem
   c. |f| on G' ≤ solving original problem

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' ≥ solving original problem
   c. |f| on G' ≤ solving original problem

Correctness of the Reduction

Claim: 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

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

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

1. |f| on G' = D
2. D = |f| on G'

Correctness of the Reduction

Claim: 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

Claim: 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 $(u,v) \in E$, $f(u,v) = f'(u,v)$

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.

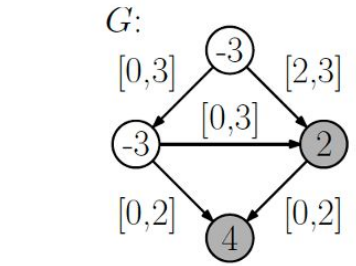
A flow of size |f'| = D can be transformed to a feasible circulation in G.

Extension #2

Capacity Bounds

Circulations with Demands and **Lower Bounds**

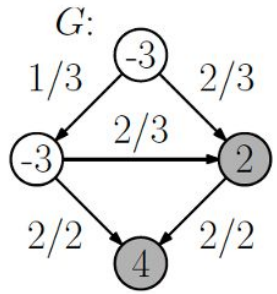
- Add minimum capacity constraints *l(u, v)*
- Given a network G = (V, E)



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

- Each edge (u, v) ∈ E, $l(u,v) \le f(u,v) \le c(u,v)$

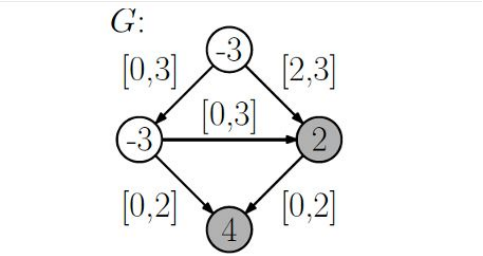
- For each vertex u ∈ V, $f^{in}(v) - f^{out}(v) = d_v$
- Is there a feasible circulation?



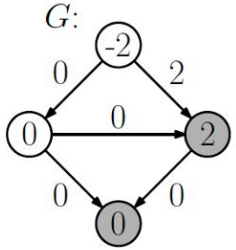
A valid flow

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
- For each $v \in V$, let L_v denote the excess flow coming into v in f_0



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



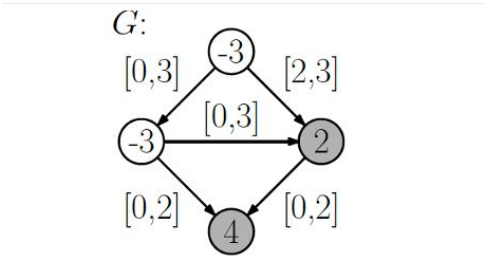
Circulation f_0
(Nodes labeled
with L values)

- 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

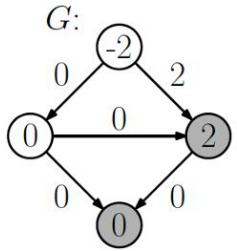
$$L_v = f_0^{in}(v) - f_0^{out}(v) = \sum_{(u,v)\in E} l(u,v) - \sum_{(v,w)\in E} l(v,w)$$

Reduction to Circulations with Demands

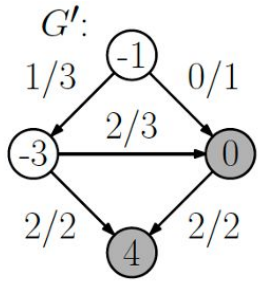
- f_0 satisfies exactly the lower bounds $f_0(u,v) = l(u,v)$
- L_v denotes the excess flow coming into v in f_0
- If $L_v = d_v$ then we have satisfied the demand condition at v
- If not, we need to add more flow
- Superimpose a circulation f_1 on top of f_0 that will clear the remaining “imbalance” at v



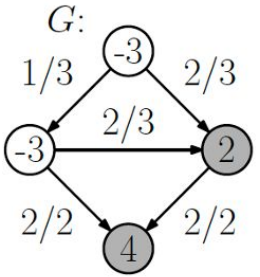
Initial network
(edges labeled with $[\ell, c]$ bounds)



Circulation f_0
(Nodes labeled
with L values)



Valid circulation
f_1



Final circulation
(for original network)

Correctness of the Reduction

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

Summary

- Next class: correctness and time analysis of circulation with demands with lower bounds
- Start HW8