

# Optimization

Spring 2010

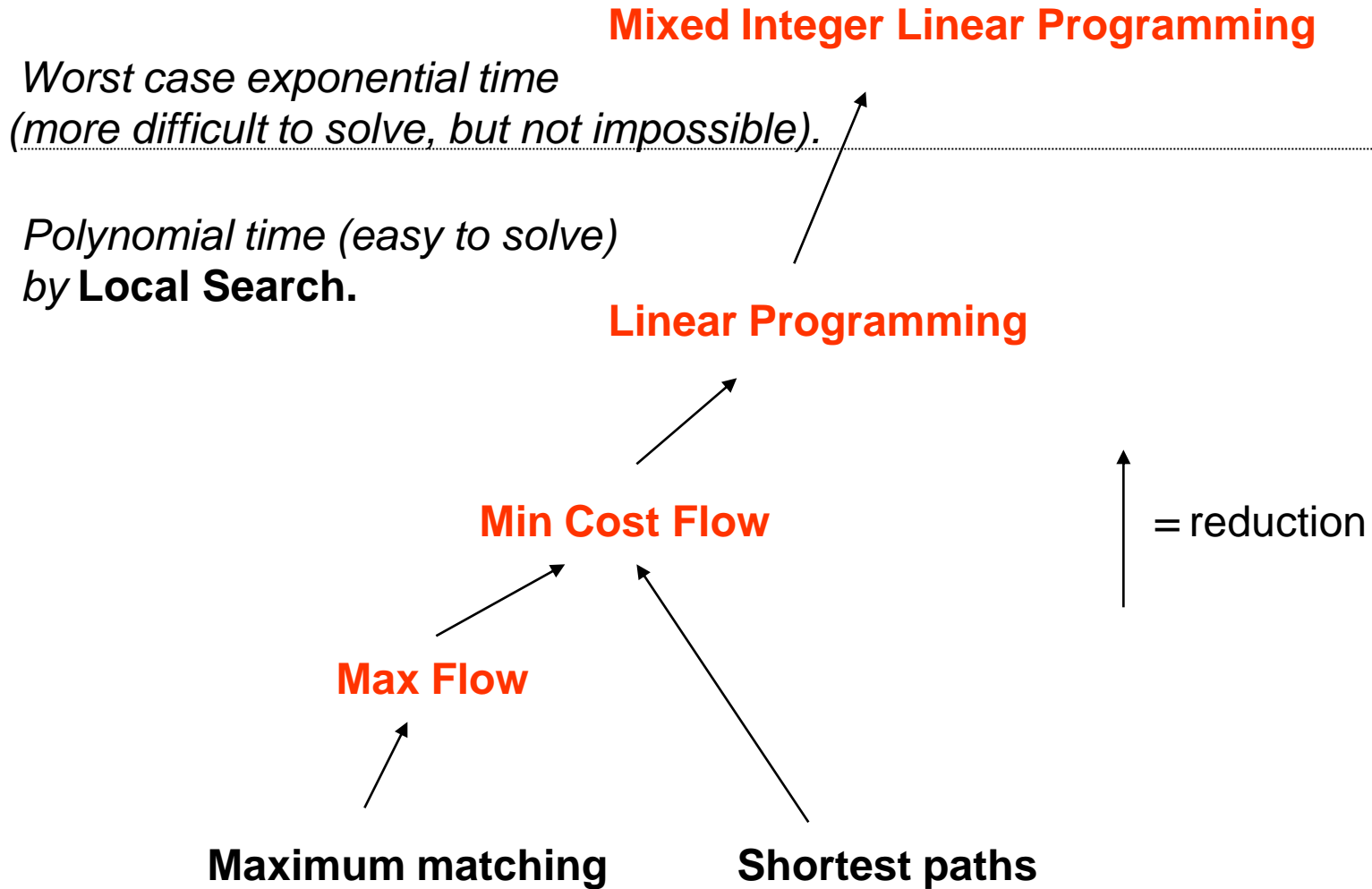
# Some practical remarks

- Lecturers: Kristoffer Arnsfelt Hansen and Peter Bro Miltersen.
- Homepage: [www.daimi.au.dk/dOpt](http://www.daimi.au.dk/dOpt)
- Exam: Written, 3 hours.
- There will be three compulsory assignments. If you want to transfer credit from last year, let me know as soon as possible *and before March 1*.
- The solution to the compulsory assignments should be handed in at specific tutorial and given to the instructor *in person*.
- Text: “Kompendium” available at GAD.

# FAQ about compulsory assignments

- **Q:** Do I really have to hand in all three assignments?
- **A:** **YES!**
  
- **Q:** Do I really have to hand in all three assignments *on time*?
- **A:** **YES!**
  
- **Q:** What if I can't figure out how to solve them?
- **A:** **Ask your instructor.** Start solving them early, so that you will have sufficient time.
  
- **Q:** What if I get sick or my girlfriend breaks up or my hamster dies?
- **A:** Start solving them early, so that you will have sufficient time in case of emergencies.
  
- **Q:** Do I really have to hand in all three assignments?
- **A:** **YES!**

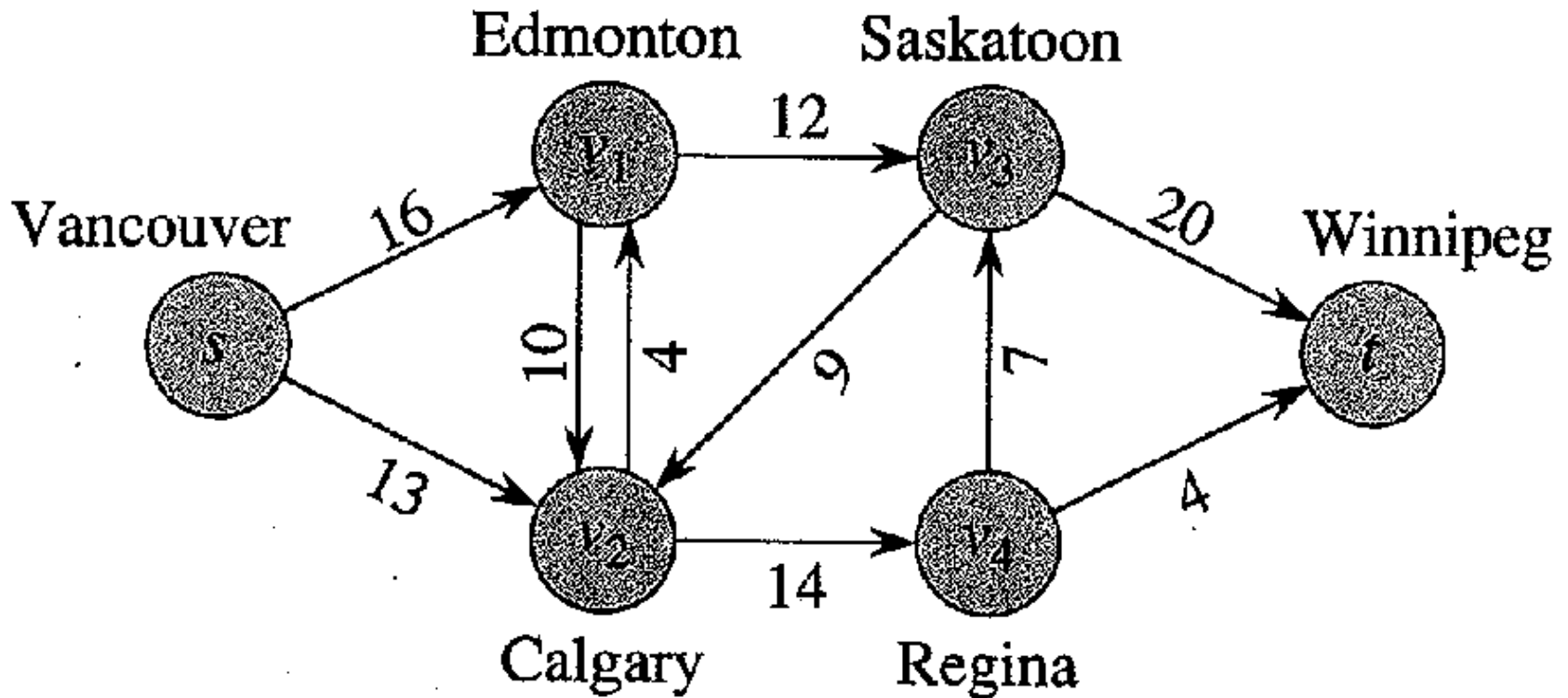
# The course in one slide



# Max Flow

- Max Flow was already covered in dADS
- The Ford-Fulkerson and Edmonds-Karp algorithms for solving max flow instances are ***not*** exam curriculum.
- Modelling concrete optimization problem using max flow ***is still*** exam curriculum.

# The Max Flow Problem



# Flow networks

- ***Flow networks*** are the ***problem instances*** of the max flow problem.
- A flow network is given by
  - 1) a ***directed graph***  $G = (V, E)$
  - 2) ***capacities***  $c: E \rightarrow \mathbf{R}^+$ .
  - 3) The ***source***  $s \in V$  and the ***sink***  $t \in V$ .
- **Convention:**  $c(u, v) = 0$  for  $(u, v)$  not in  $E$ .

# Flows

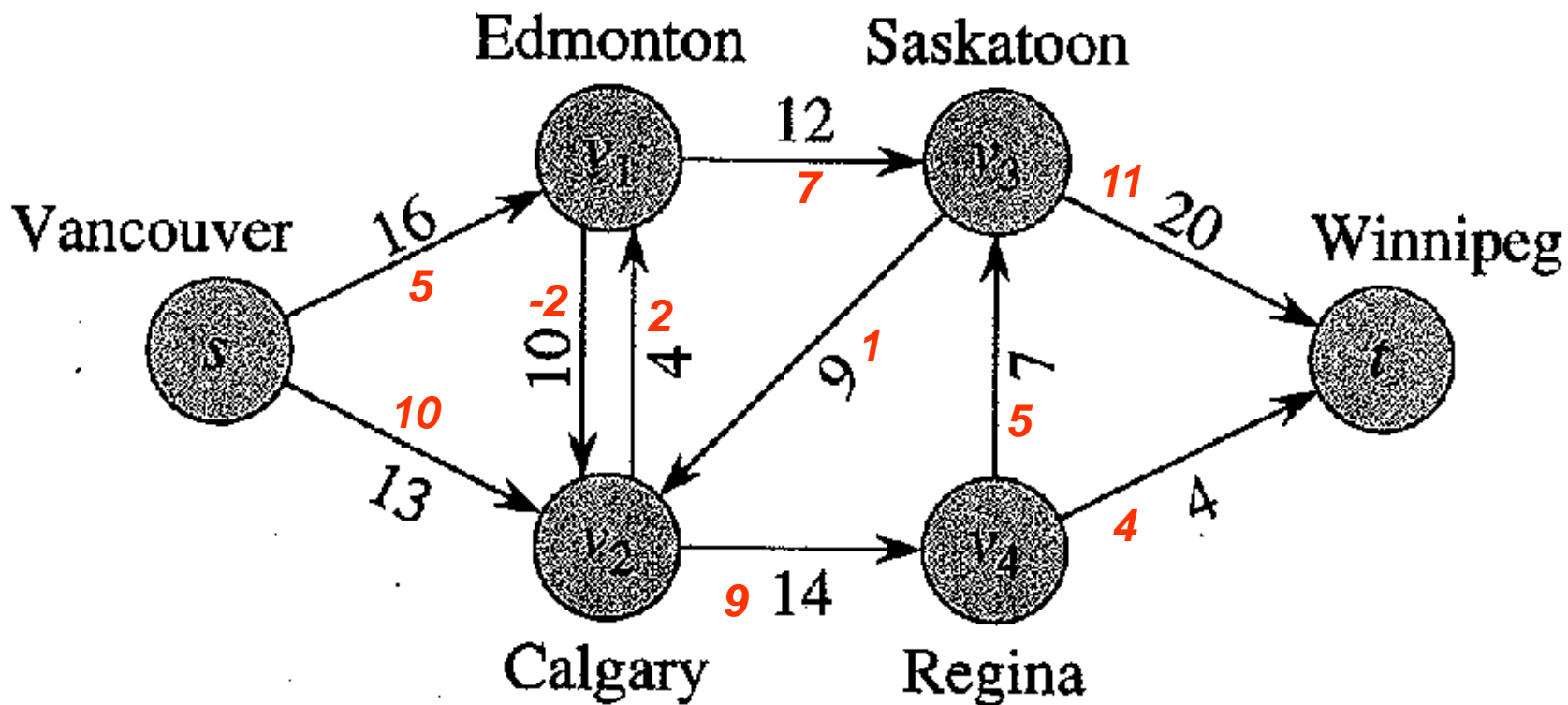
- Given flow network, a **flow** is a **feasible solution** to the max flow problem.
- A flow is a map  $f: V \times V \rightarrow \mathbf{R}$  satisfying

**capacity constraints:**  $\forall (u, v): f(u, v) \leq c(u, v)$ .

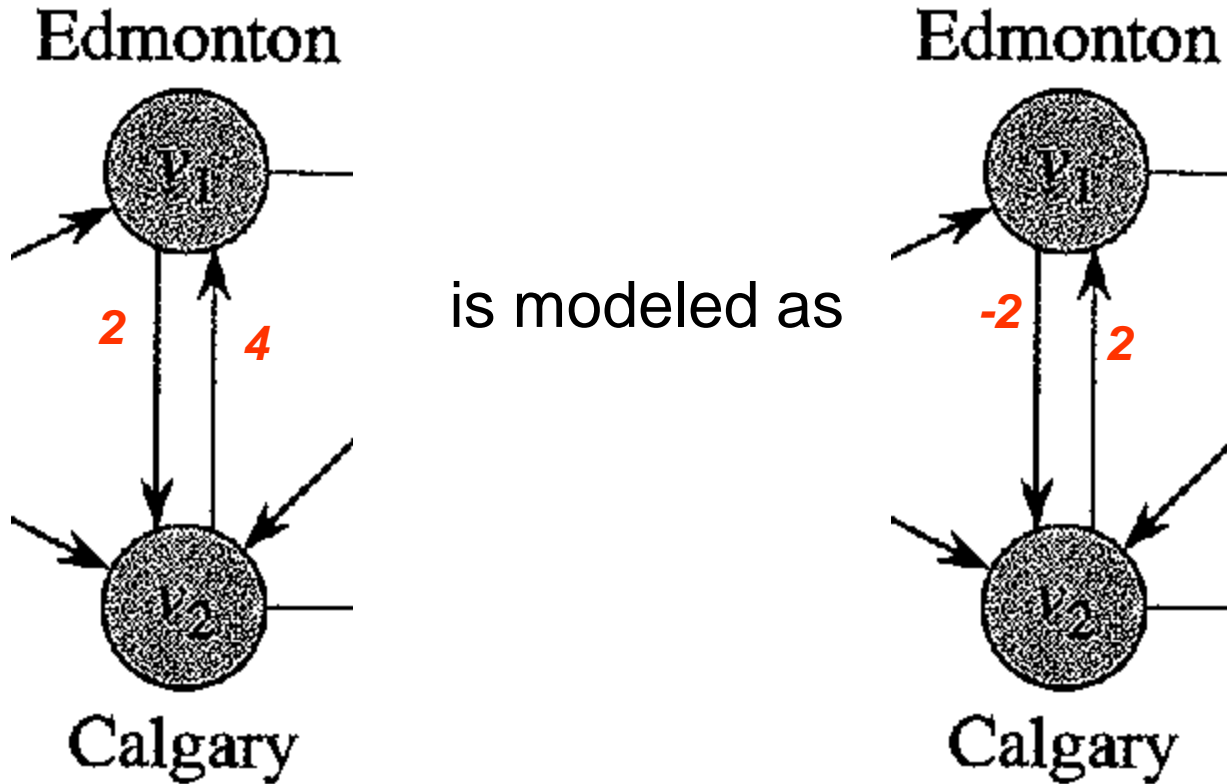
**Skew symmetry:**  $\forall (u, v): f(u, v) = -f(v, u)$ .

**Flow conservation:**  $\forall u \in V - \{s, t\}: \sum_{v \in V} f(u, v) = 0$

# A Flow

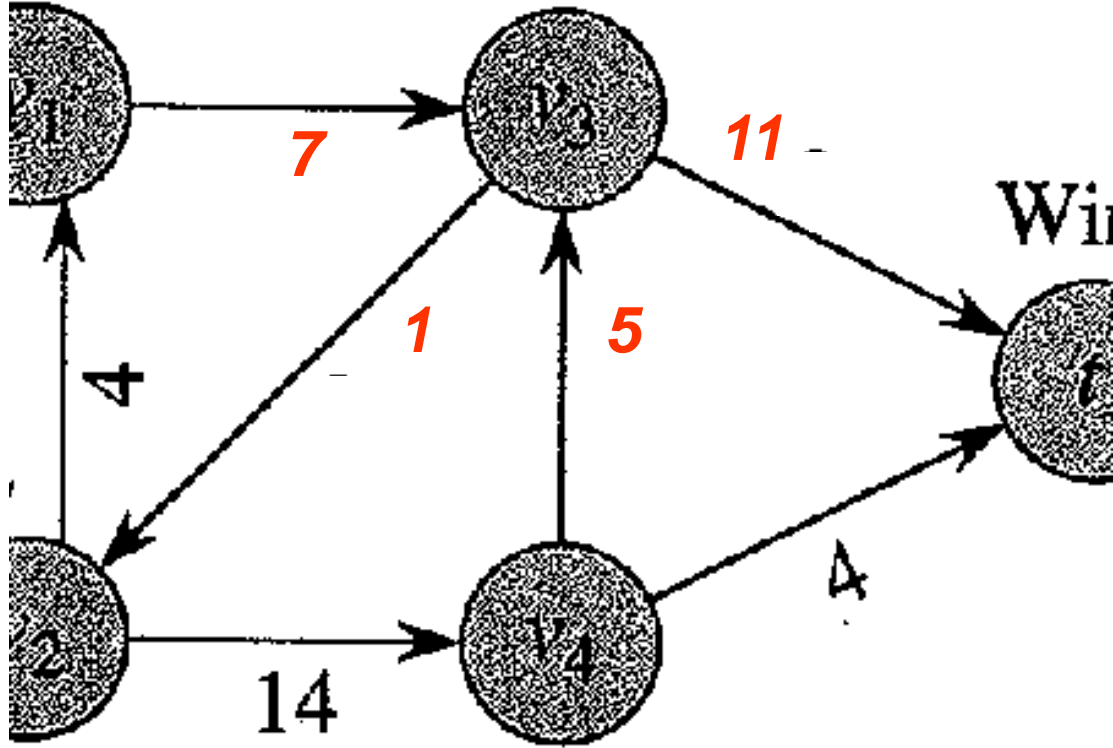


# Skew Symmetry



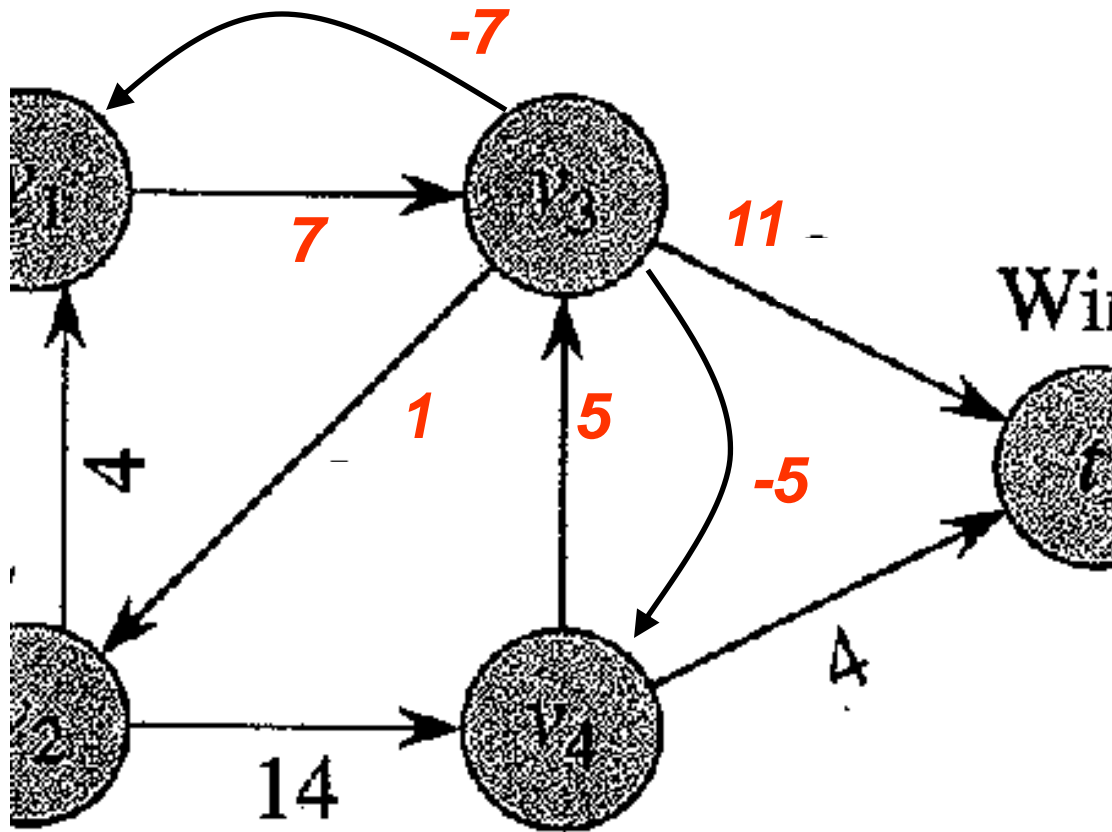
Our flows are *net flows*.

# Flow Conservation

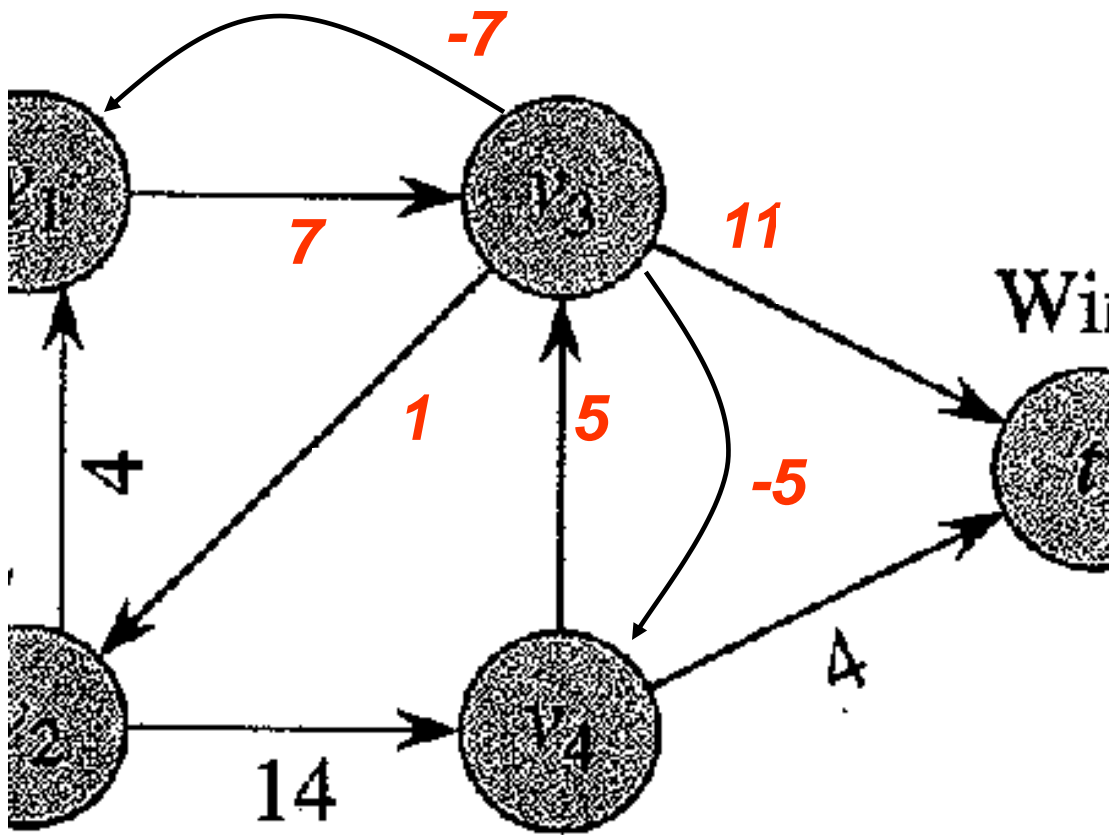


$$\sum_u f(v_3, u) = 0?$$

# Flow Conservation



$$\sum_u f(v_3, u) = (-7) + 1 + (-5) + 11 = 0$$



Flow **entering**  $v_3 = 12$

Flow **leaving**  $v_3 = 12$

Flow conservation expresses that  $v_3$  is in **balance**.

# The max flow problem

- $f(X, Y) := \sum_{u \in X, v \in Y} f(u, v)$ .
- **Value** of  $f$ :  $|f| := f(s, V)$ .

## The Max Flow Problem:

Given a flow network  $(V, E, c, s, t)$ , find the feasible flow  $f$  maximizing  $|f|$ .

# Algorithms for solving the Max Flow problem

- Ford-Fulkerson algorithm
- Edmonds-Karp algorithm
- Examples of the ***local search*** algorithmic pattern.

# Local Search Pattern

LocalSearch(ProblemInstance  $x$ )

$y :=$  feasible solution to  $x$ ;

**while**  $\exists z \in N(y): v(z) > v(y)$  **do**

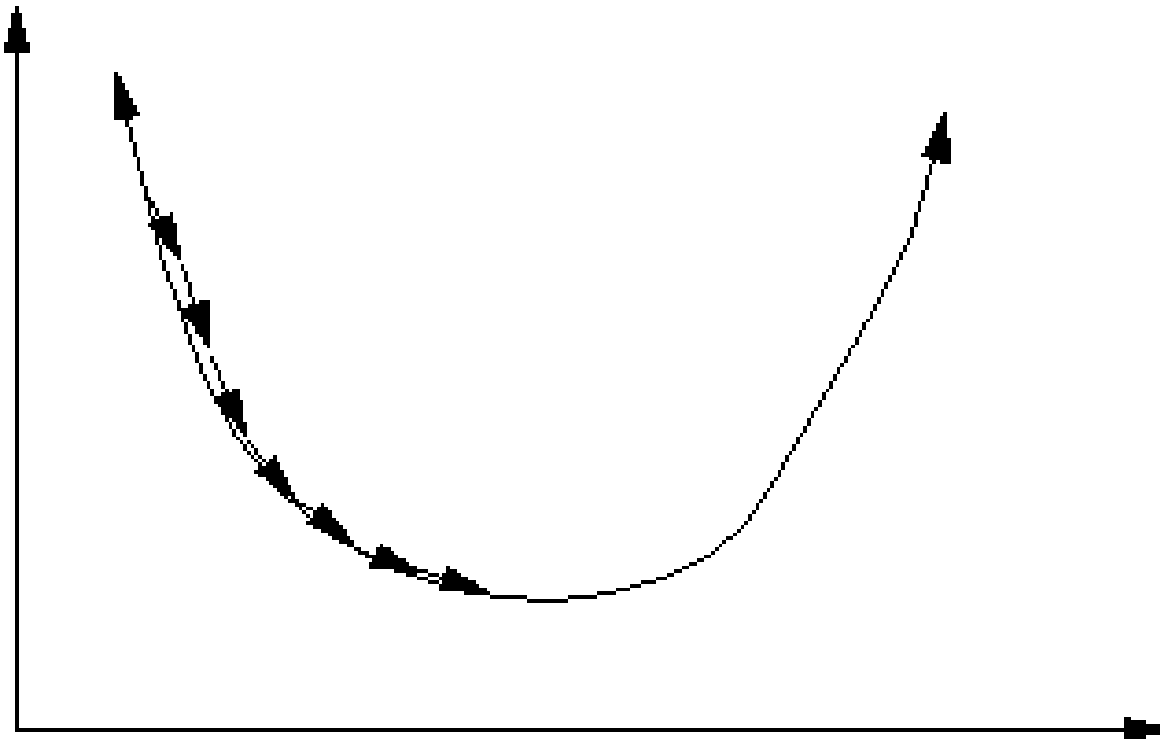
$y := z$ ;

**od**;

**return**  $y$ ;

$N(y)$  is a *neighborhood* of  $y$ .

# Local Search



# Local search checklist

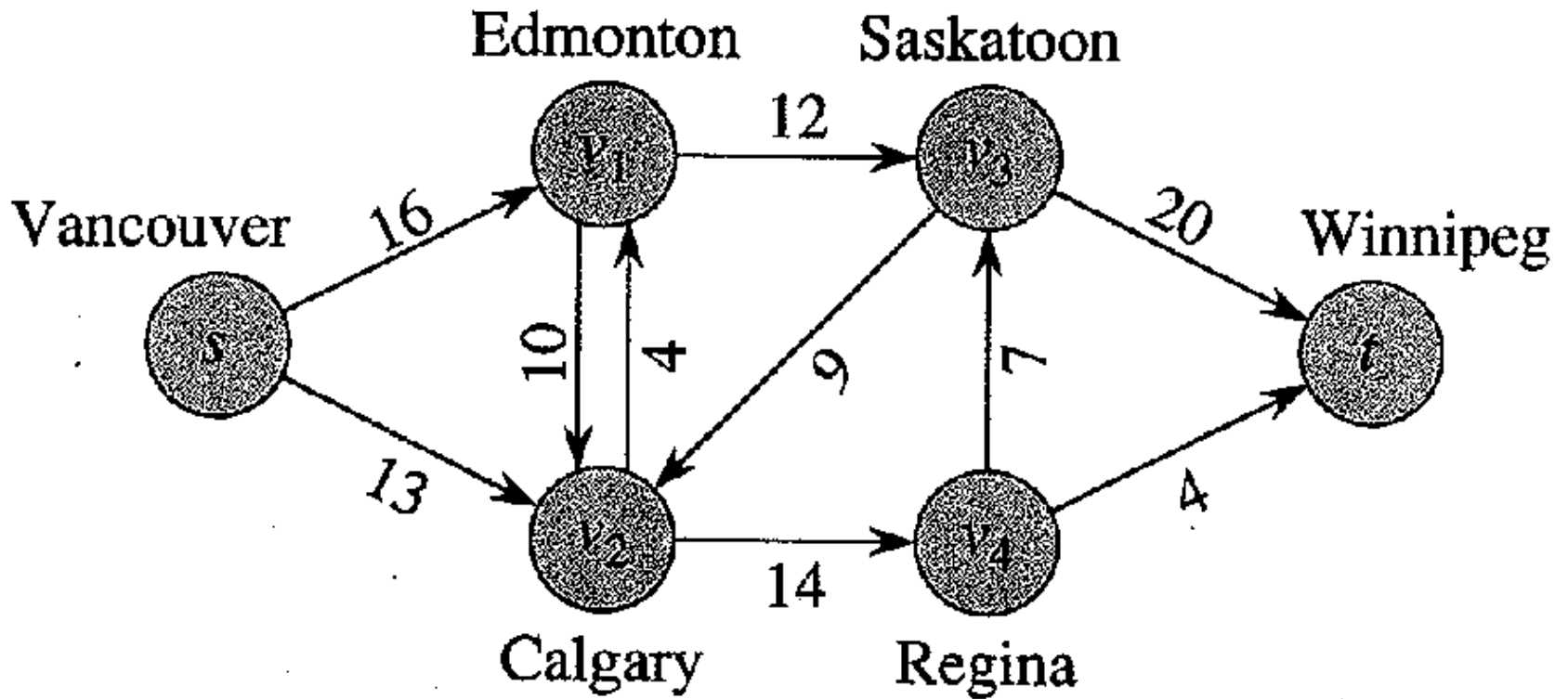
## **Design:**

- How do we find the first feasible solution?
- Neighborhood design?
- Which neighbor to choose?

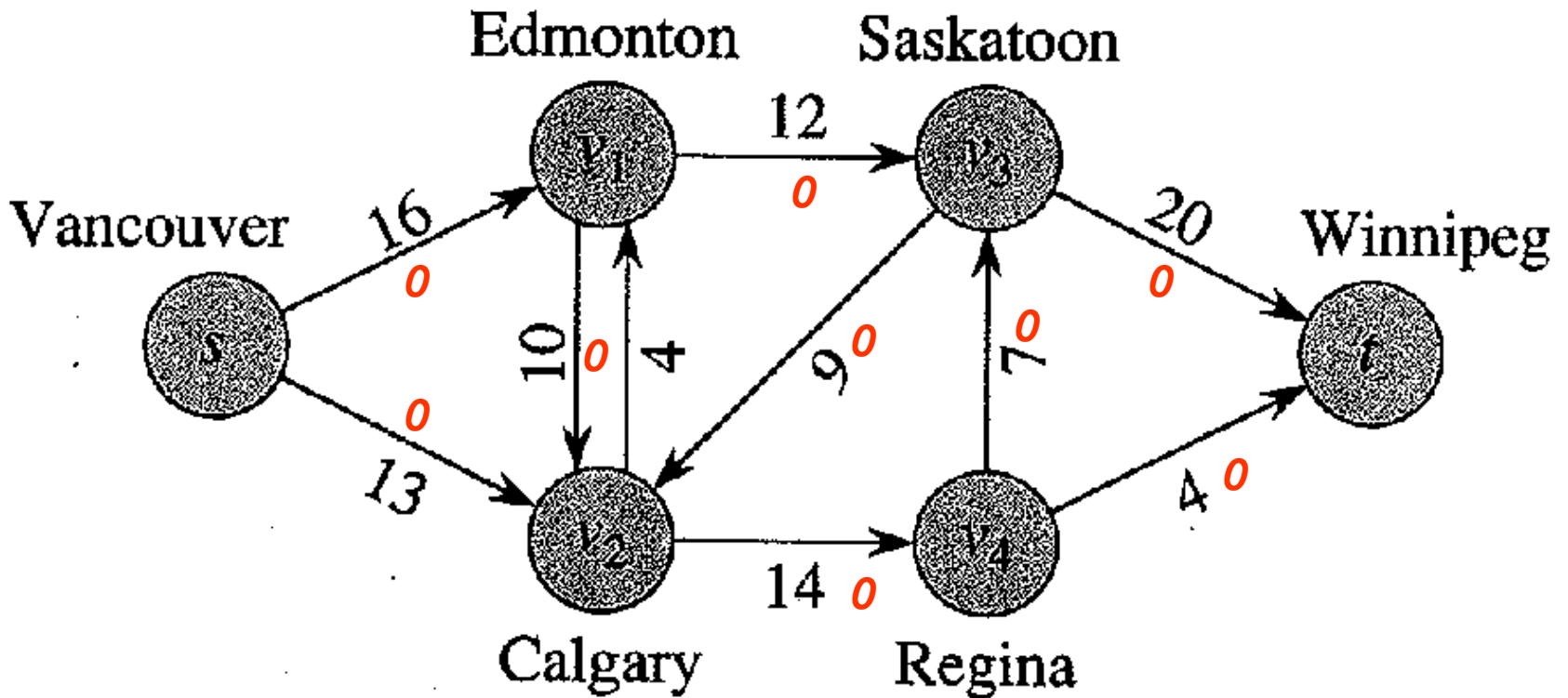
## **Analysis:**

- Partial correctness? (termination  $\Rightarrow$  correctness)
- Termination?
- Complexity?

# The first flow?



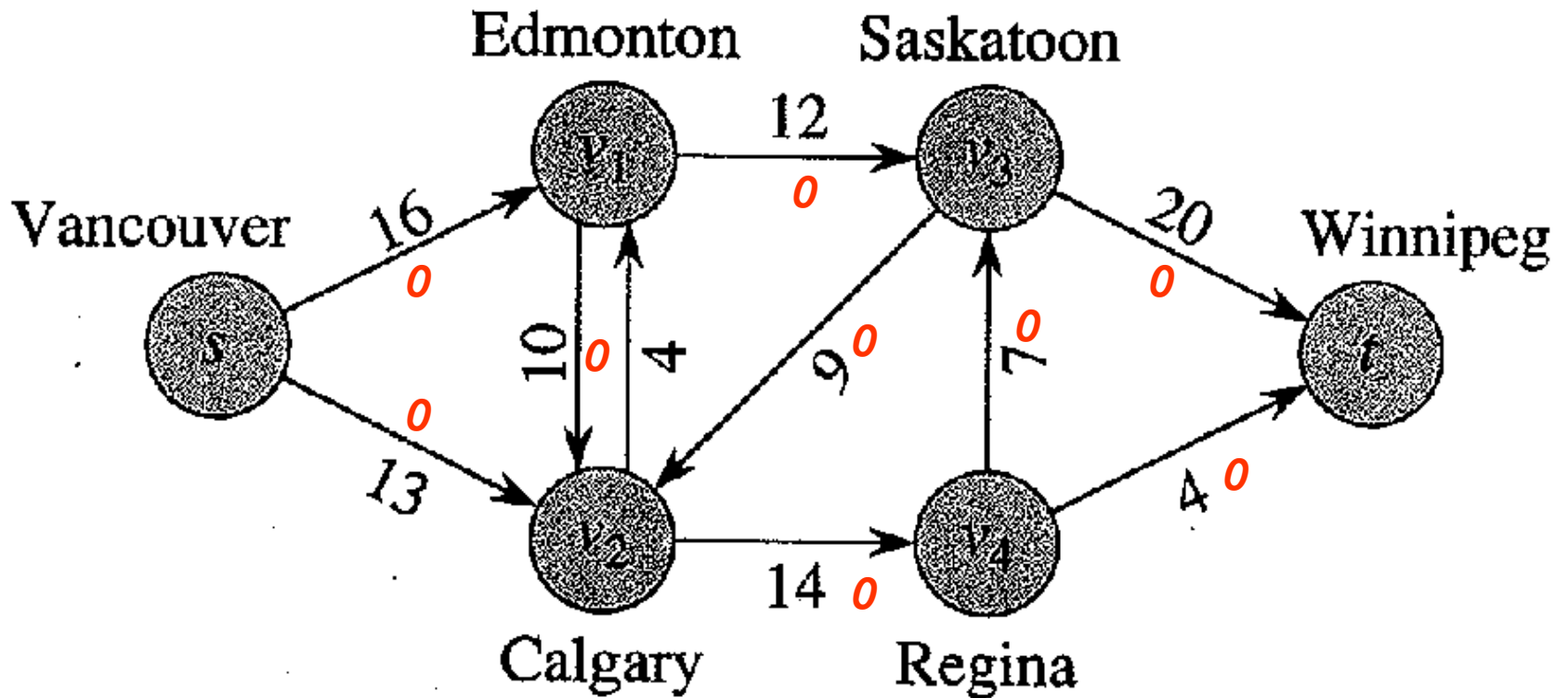
# The first flow?



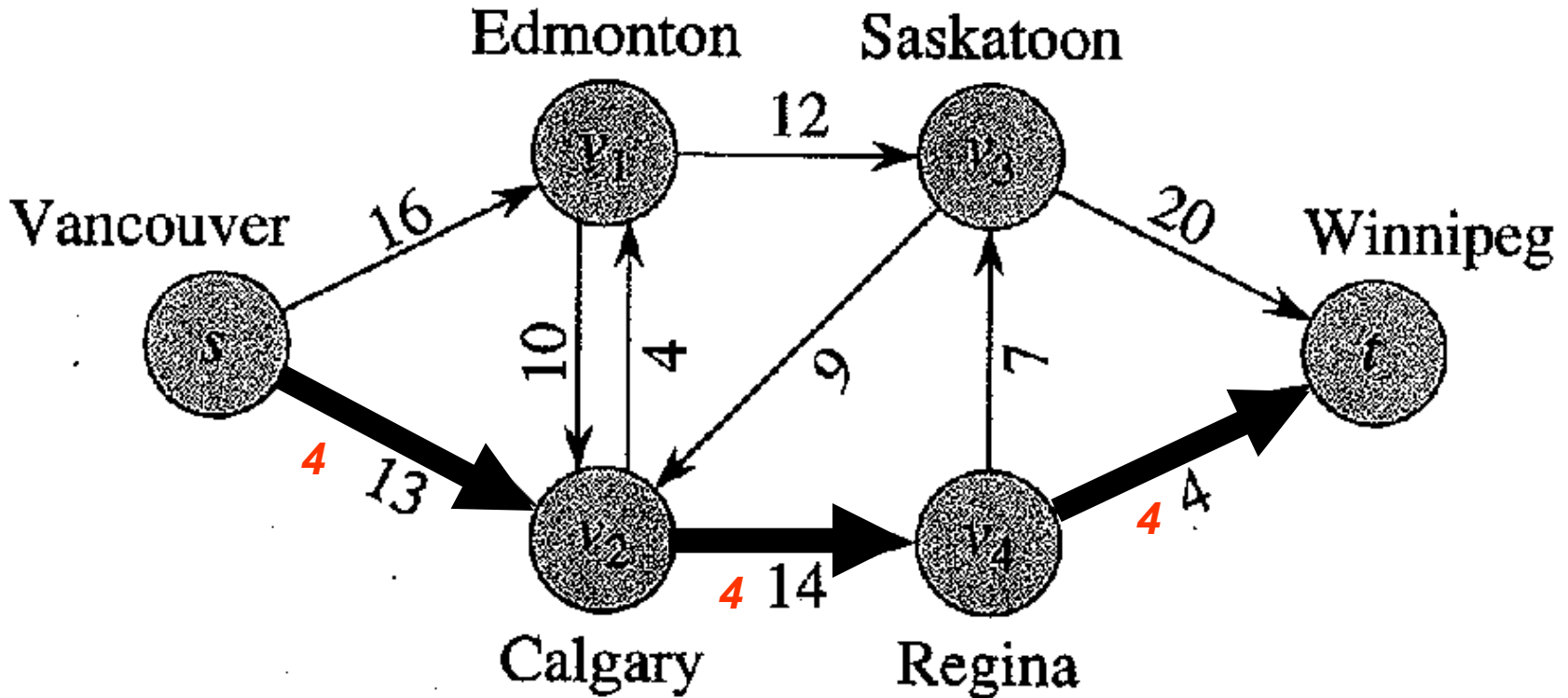
# Neighborhood design

- Given a flow, how can we find a slightly different (and hopefully slightly better) flow?

# The first flow?



# Path Flows

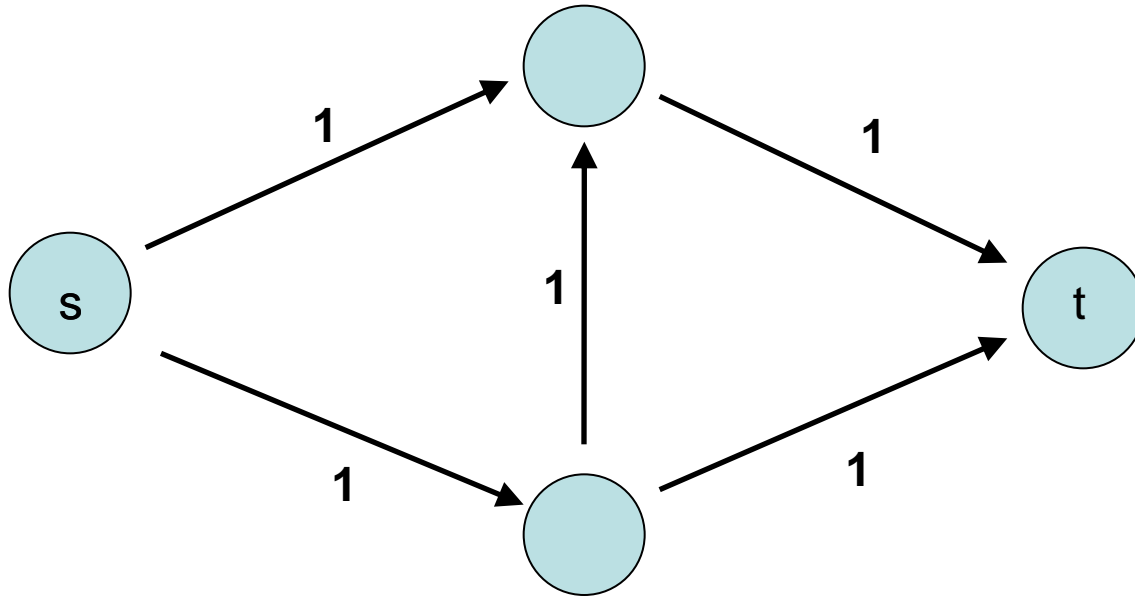


Path Flow = flow with positive values only on a simple path from  $s$  to  $t$

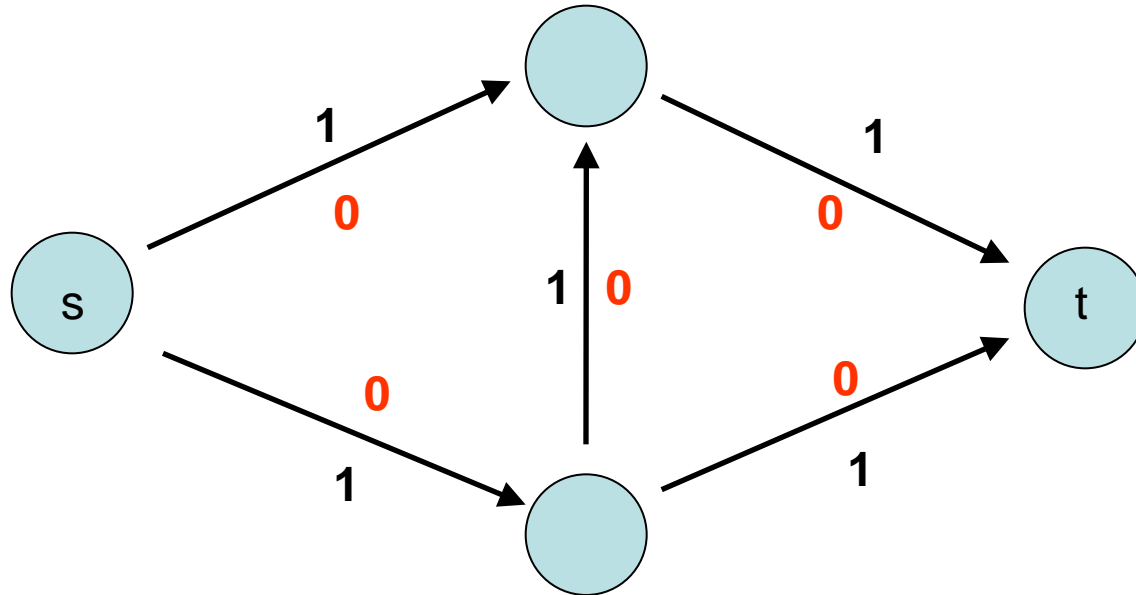
# Idea

- Let  $N(y)$  be the flows that can be obtained from  $y$  by adding a path flow (without violating the capacity bounds).
- The path flow we add should use some path in  $(V, E)$  along which every edge has some unused capacity.

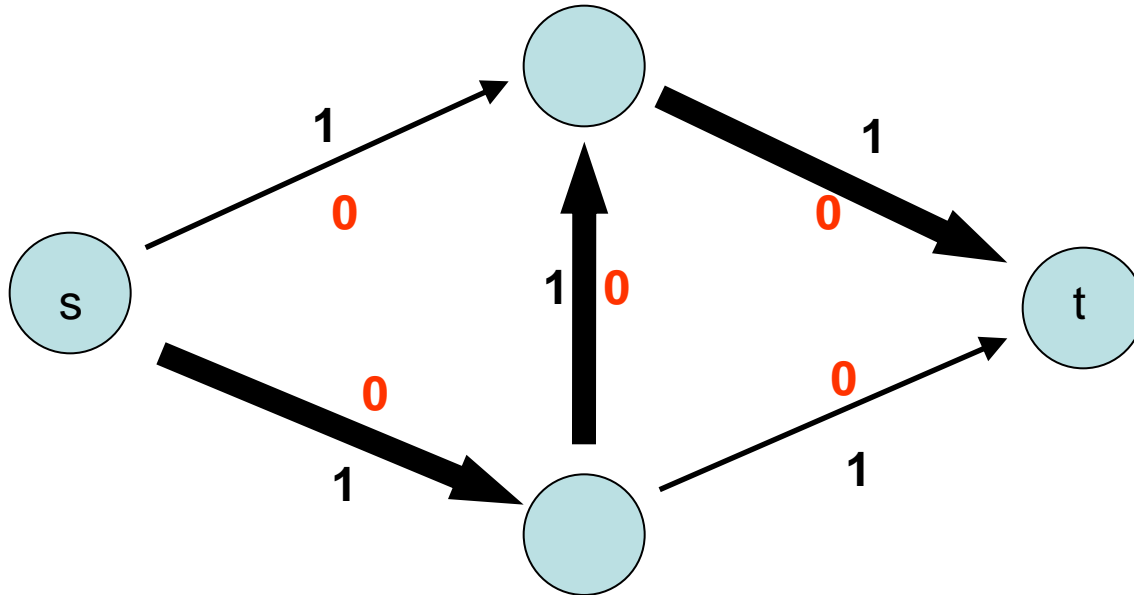
# Example



# Example

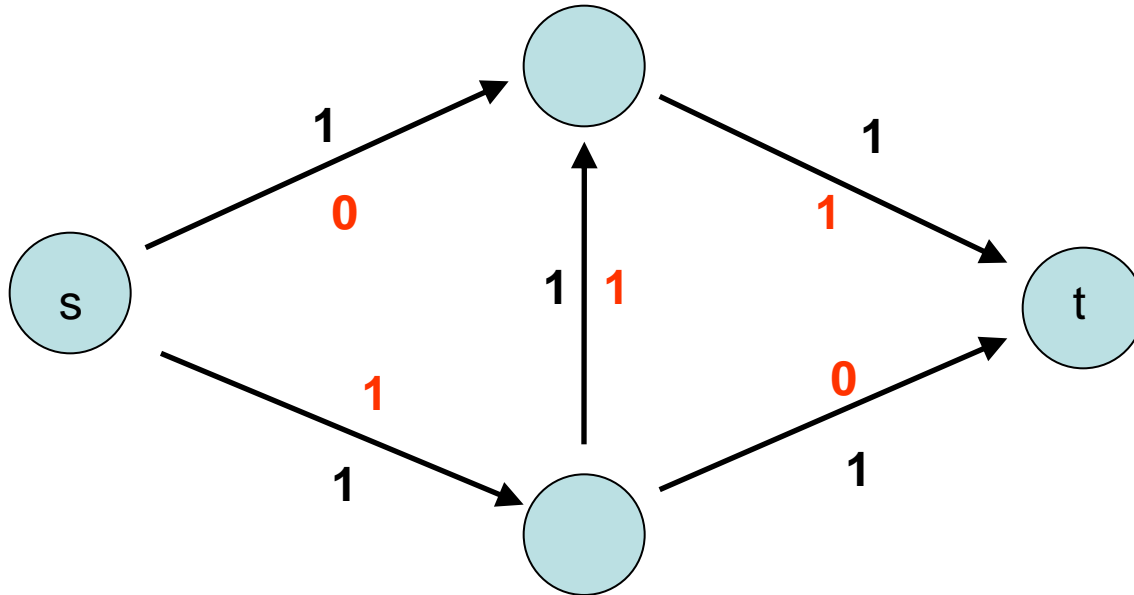


# Example

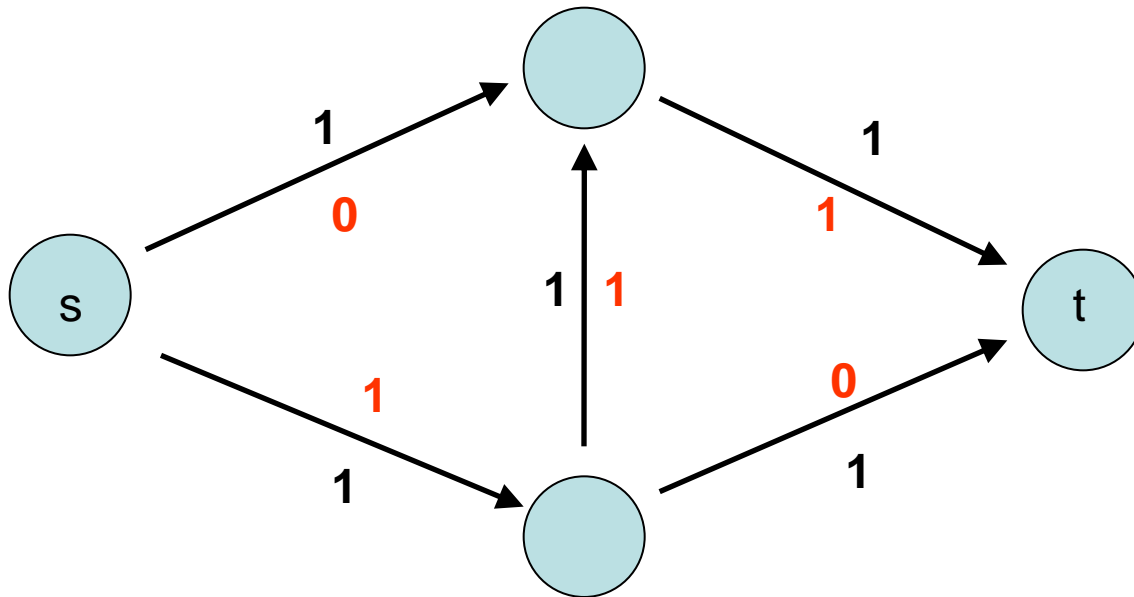


# Another Path?

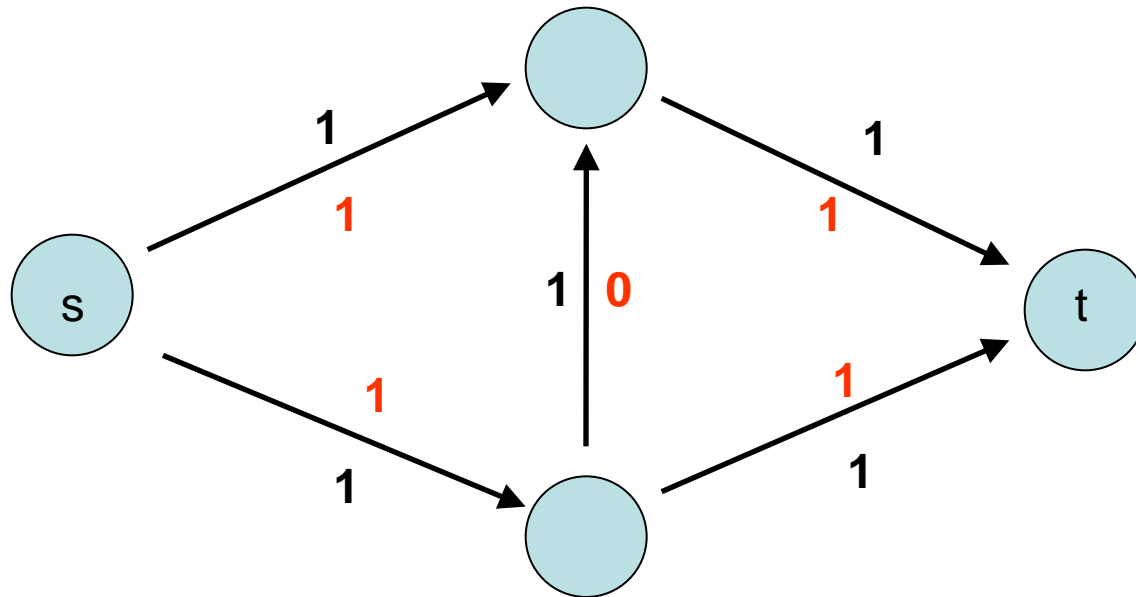
**No.**



# Optimal?



# No!



# Some remarks

- When designing a local search algorithm, the most obvious neighborhood relation is not necessarily the right one.
- That a solution cannot be improved by using some specified set of changes does **not** necessarily mean it is globally optimal.

# Better Idea

- The graph  $(V, E)$  is not really the right one to find paths in.
- The path flow we add should use some path in  $(V, E_f)$  where  $E_f$  is the set of edges that has unused capacity under the current flow  $f$ .
- $E_f$  may include edges  $(u, v) \in E$  as well as **back-edges**  $(u, v)$  for which  $(v, u) \in E$ .

# The residual network

- Let  $G=(V,E,c,s,t)$  be a flow network and let  $f$  be a flow in  $G$ .
- The ***residual network*** is the flow network with edges and capacities

$$E_f = \{(u,v) \in V \times V \mid f(u,v) < c(u,v)\}$$

$$c_f(u,v) = c(u,v) - f(u,v)$$

# Augmenting Paths

- A simple path  $p$  from  $s$  to  $t$  in  $G_f$  is called an **augmenting path**.
- Let  $c_f(p) = \min\{ c_f(u,v) : (u,v) \text{ is on } p \}$
- Let  $f_p(u,v)$  be
  - $c_f(p)$  if  $(u,v)$  is on  $p$
  - $-c_f(p)$  if  $(v,u)$  is on  $p$
  - 0 otherwise
- Then  $f_p$  is a path flow in  $G_f$  with value  $c_f(p)$

# Ford-Fulkerson method

Ford-Fulkerson( $G$ )

$f = \mathbf{0}$

**while**( $\exists$  simple path  $p$  from  $s$  to  $t$  in  $G_f$ )

$f := f + f_p$

**output**  $f$

# Local search checklist

## **Design:**

- How do we find the first feasible solution?
- Neighborhood design?
- Which neighbor to choose?

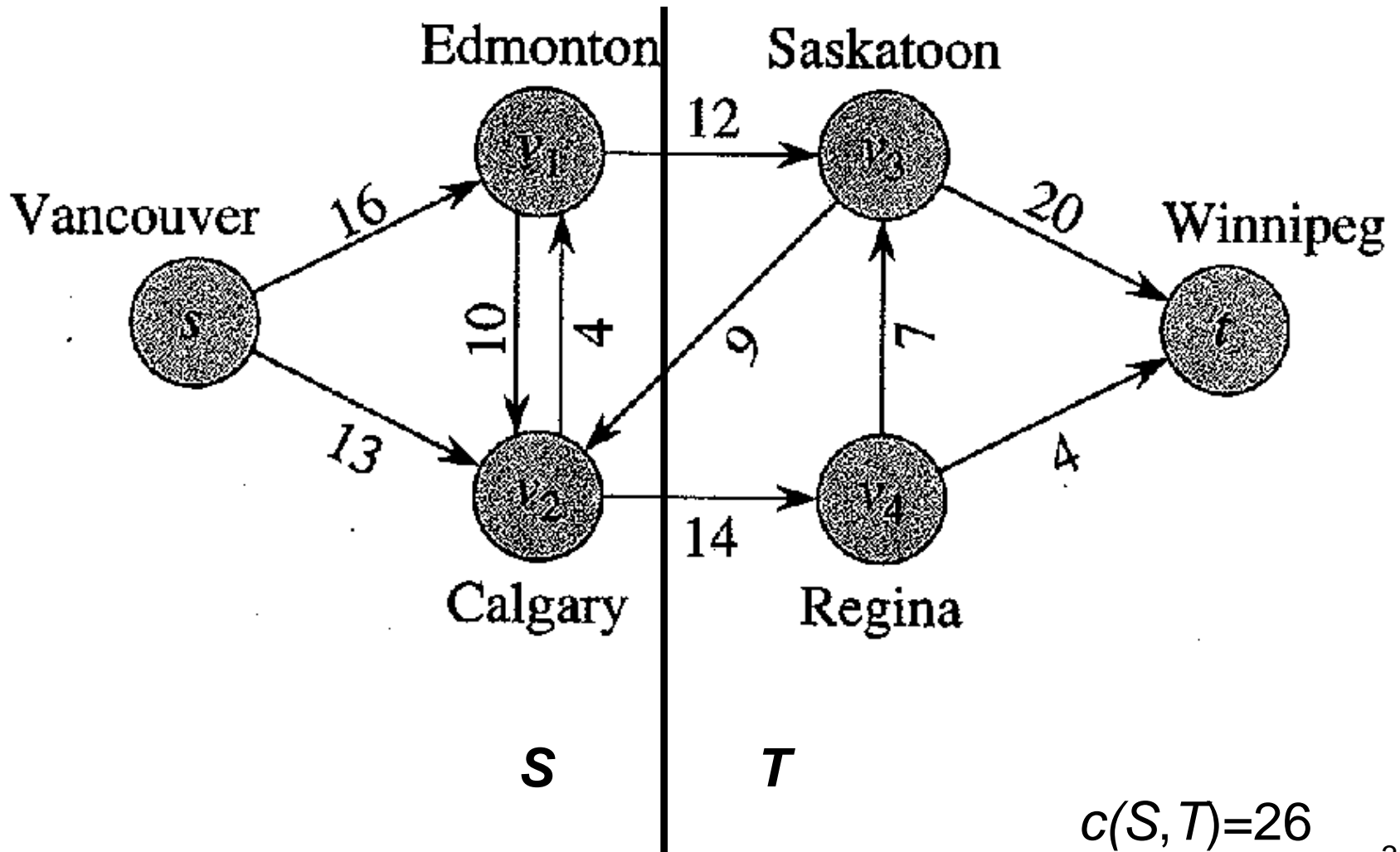
## **Analysis:**

- Partial correctness? (termination  $\Rightarrow$  correctness)
- Termination?
- Complexity?

# Cuts

- A **cut**  $(S, T)$  in  $G$  is a partition of  $V$  into  $S$  and  $T=V-S$  with  $s \in S$  and  $t \in T$ .
- Its **capacity** is
$$c(S, T) = \sum_{u \in S, v \in T} c(u, v)$$
- A **minimum cut** is a cut with smallest capacity among all cut.

# A cut



# Lemma 26.5

- Let  $f$  be a flow in  $G$  and let  $(S, T)$  be a cut in  $G$ . Then  $|f| = f(S, T)$ .

# Corollary 26.6

- Let  $f$  be a flow in  $G$  and let  $(S, T)$  be a cut in  $G$ . Then  $|f| \leq c(S, T)$ .
- This is a ***weak duality theorem***.

# Max Flow – Min Cut Theorem

Let  $f$  be a flow in  $G$ . The following three conditions are equivalent:

1.  $f$  is a maximum flow
2.  $G_f$  contains no augmenting path
3. There is a cut  $(S, T)$  so that  $|f|=c(S, T)$

# Max Flow – Min Cut Theorem

- The value of the maximum flow in  $G$  is equal to the capacity of the minimum cut in  $G$ .
- This is a ***strong duality theorem***.

# Flows vs Cuts

- The solution *values* agree, not the solutions themselves – flows and cuts are completely different objects.
- Given a max flow we can easily find a min cut (follows from proof of max flow-min cut theorem). Going the other way is less obvious.

# Consequence

- The Ford-Fulkerson method is ***partially correct***, i.e., *if* it terminates it produces the flow with the maximum value.

# Local search checklist

## **Design:**

- How do we find the first feasible solution?
- Neighborhood design?
- Which neighbor to choose?

## **Analysis:**

- Partial correctness? (termination  $\Rightarrow$  correctness)
- Termination?
- Complexity?

# Termination

- Suppose all capacities are integers.
- We start with a flow of value 0.
- In each iteration, we get a new flow with higher integer value.
- We always have a legal flow, i.e., one of value at most  $|f|$ .
- Hence we can have at most  $|f|$  iterations.

# Correctness of Ford-Fulkerson

- Since Ford-Fulkerson is *partially correct* and it *terminates* if capacities are integers it is a *correct* algorithm for finding the maximum flow *if* capacities are integers.

# Integrality Theorem (26.11)

If a flow network has integer valued capacities, there is a maximum flow with an integer value on every edge. The Ford-Fulkerson method will yield such a maximum flow.

The integrality theorem is often ***extremely*** important when “programming” and modeling using the max flow formalism.

# Local search checklist

## **Design:**

- How do we find the first feasible solution?
- Neighborhood design?
- Which neighbor to choose?

## **Analysis:**

- Partial correctness? (termination  $\Rightarrow$  correctness)
- Termination?
- Complexity?

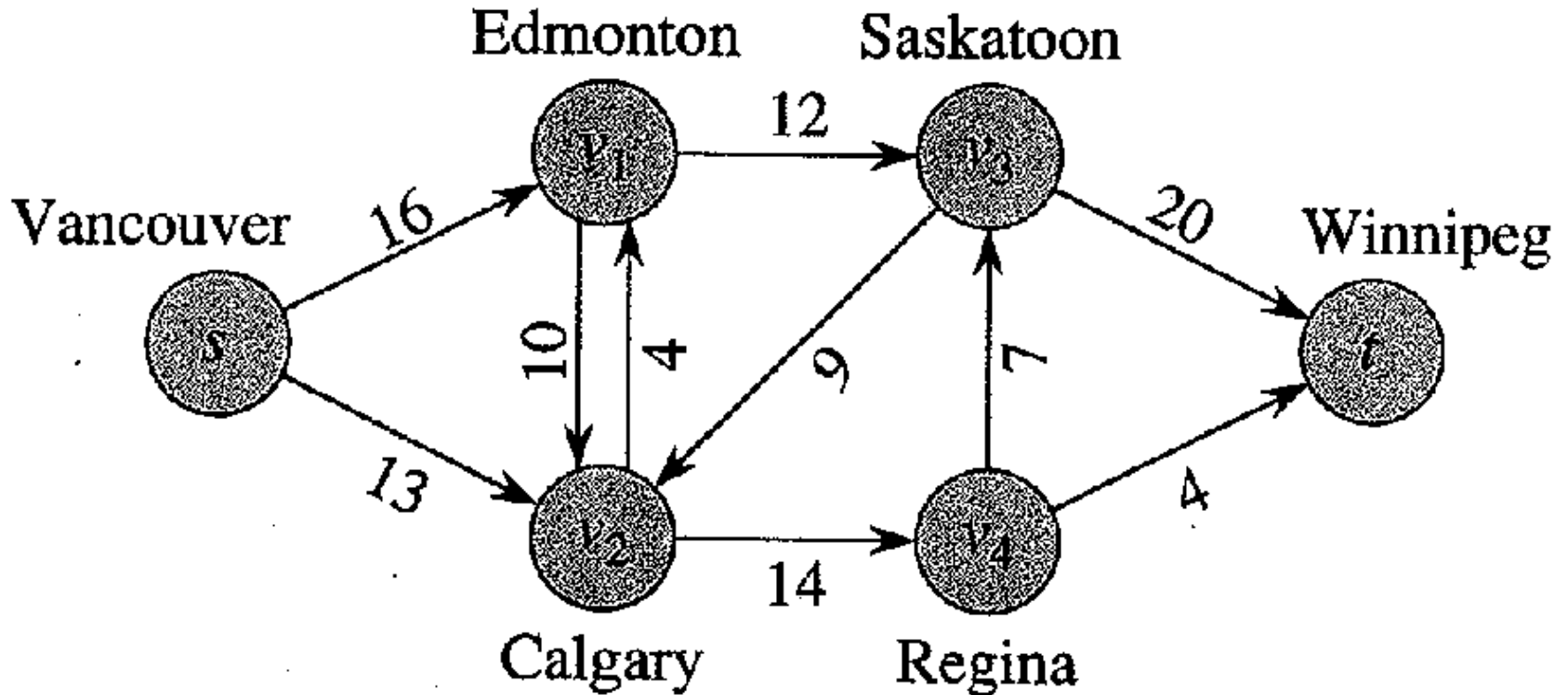
# Complexity

- The Ford-Fulkerson can be implemented to run in time at most  $O(|E| |f|)$ .
- Is this fast?

# Polynomial time algorithms

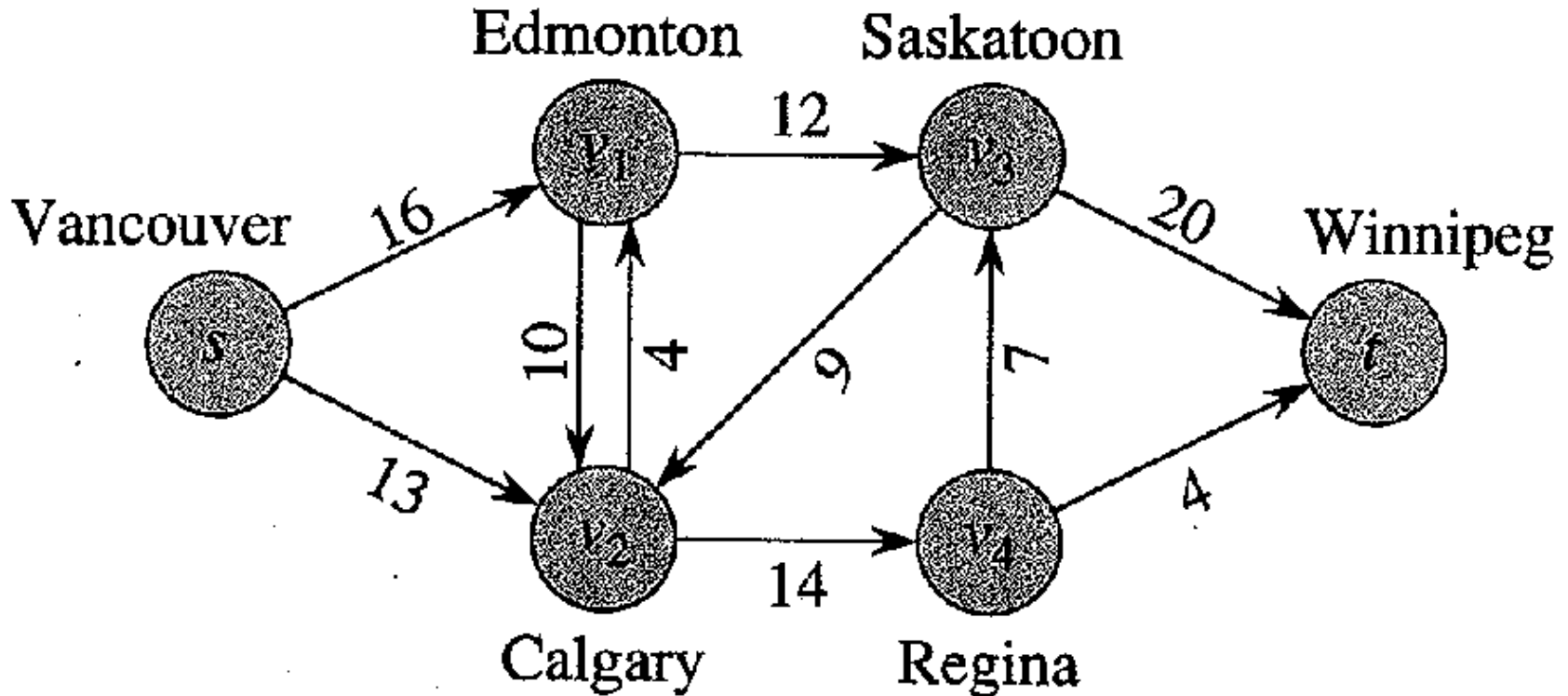
- **Defintion:** A *polynomial time algorithm* is an algorithm than runs in time polynomial in  $n$ , where  $n$  is the *number of bits* of the input.
- How we intend to encode the input influences if we have a polynomial algorithm or not. Usually, some “standard encoding” is implied.
- In this course: Polynomial  $\approx$  Fast  
Exponential  $\approx$  Slow

# How to encode max flow instance?



java MaxFlow ????????????

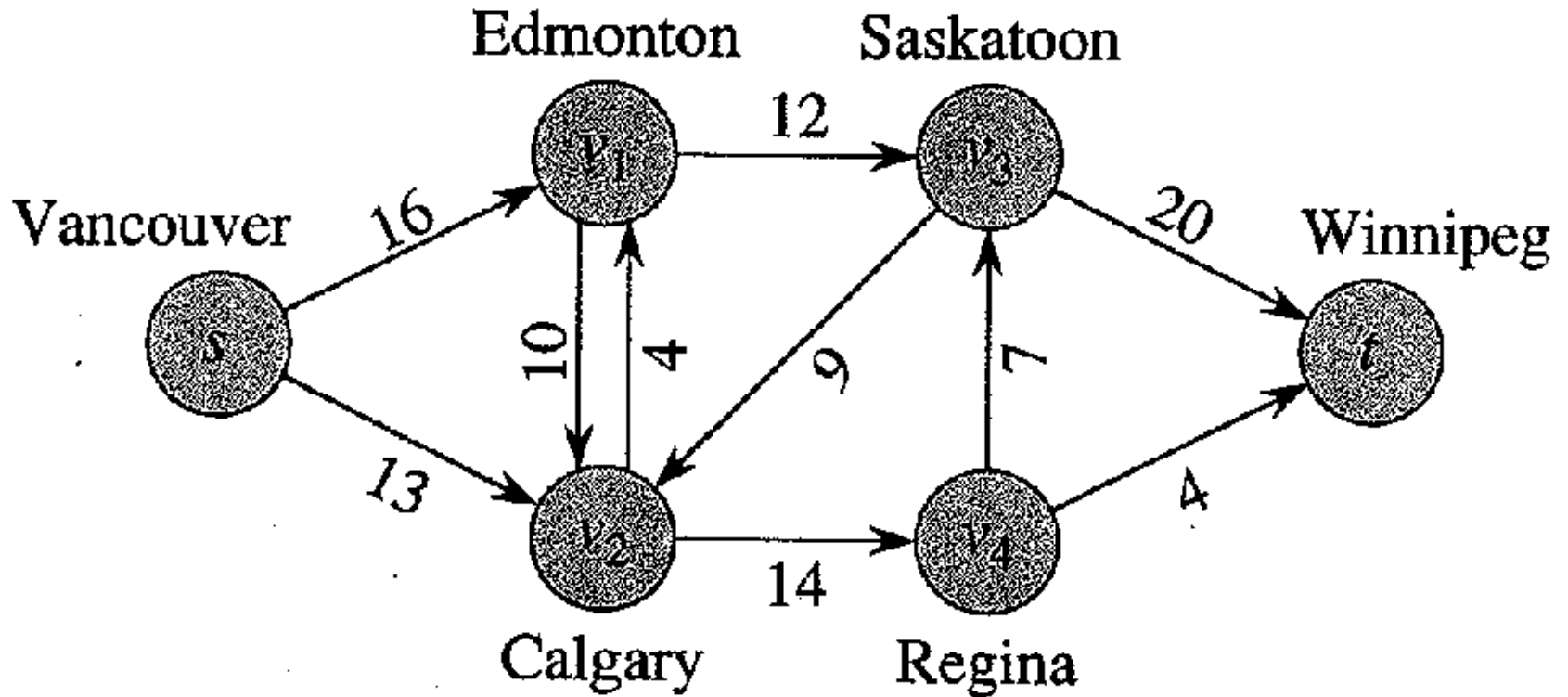
# How to encode max flow instance?



```
java MaxFlow 6#0|16|13|0|0|0#0|0|10|12|0|0  
#0|4|0|0|14|0#0|0|9|0|0|20  
#0|0|0|7|0|4|#0|0|0|0|0|0
```

# Complexity of Ford-Fulkerson

- With standard (decimal or binary) representation of integers, Ford-Fulkerson is an ***exponential*** time algorithm.



```

java MaxFlow 111111
#|111111111111111111|1111111111111111|||
#||1111111111|11111111111111||
#|1111|||1111111111111111|
#||1111111111|||111111111111111111111111
#|||11111111||1111 #|||||
  
```

# Complexity of Ford-Fulkerson

- With *unary* (4 ~ 1111) representation of integers, Ford-Fulkerson is a *polynomial* time algorithm.
- Intuition: When the input is longer it is easier to be polynomial time as a function of the input length.
- An algorithm which is polynomial if integer inputs are represented in unary is called a *pseudo-polynomial* algorithm.
- Intuitively, a pseudo-polynomial algorithm is an algorithm which is fast if all numbers in the input are small.

# Edmonds-Karp

Edmonds-Karp algorithm for Max Flow:

Implement Ford-Fulkerson by always choosing the ***shortest possible*** augmenting path, i.e., the one with fewest possible edges.

# Complexity of Edmonds-Karp

- The total running time is  $O(|V| |E|^2)$  and Edmonds-Karp is a ***polynomial time*** algorithm for Max Flow.