ネットワークフローとその代表的な問題

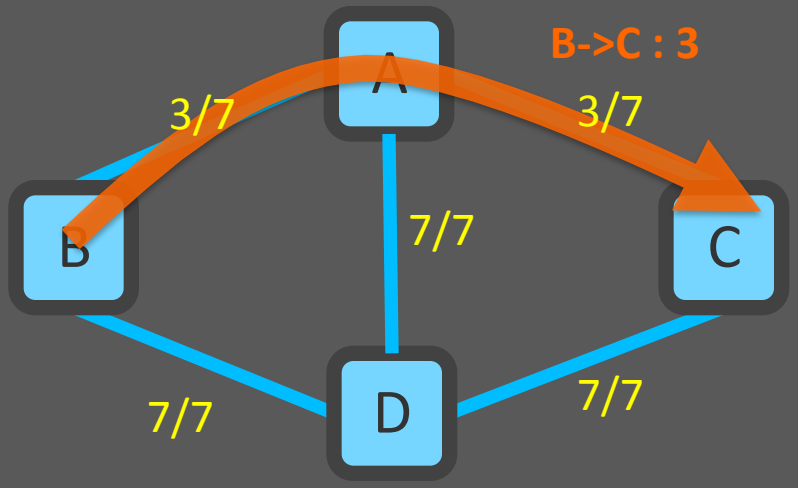
金子紘也（日本電気株式会社 情報ナレッジ研）

Internet Week 2013
S8 SDN時代を生き抜く為のグラフ理論とネットワークのアルゴリズム入門

ネットワークフローとは？

グラフ上の流れを扱う分野

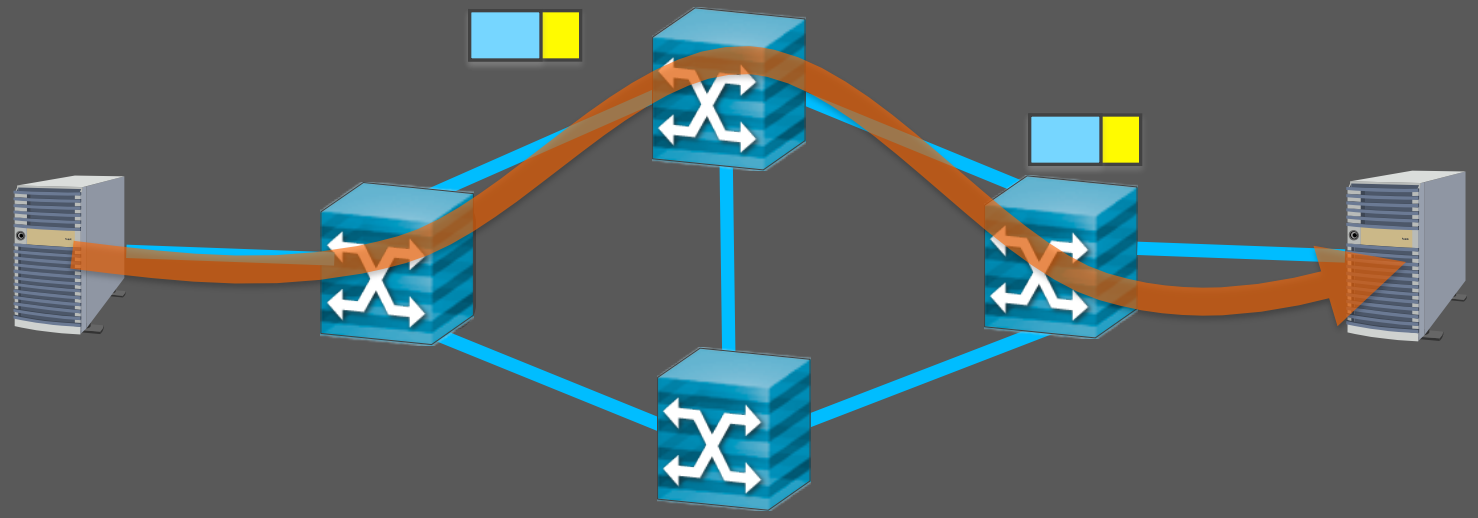
- グラフ構造にエッジ容量の概念を導入
- フローは容量を消費する
- 定義：flowはedgeを実数に写像する関数



ネットワークフローの例１

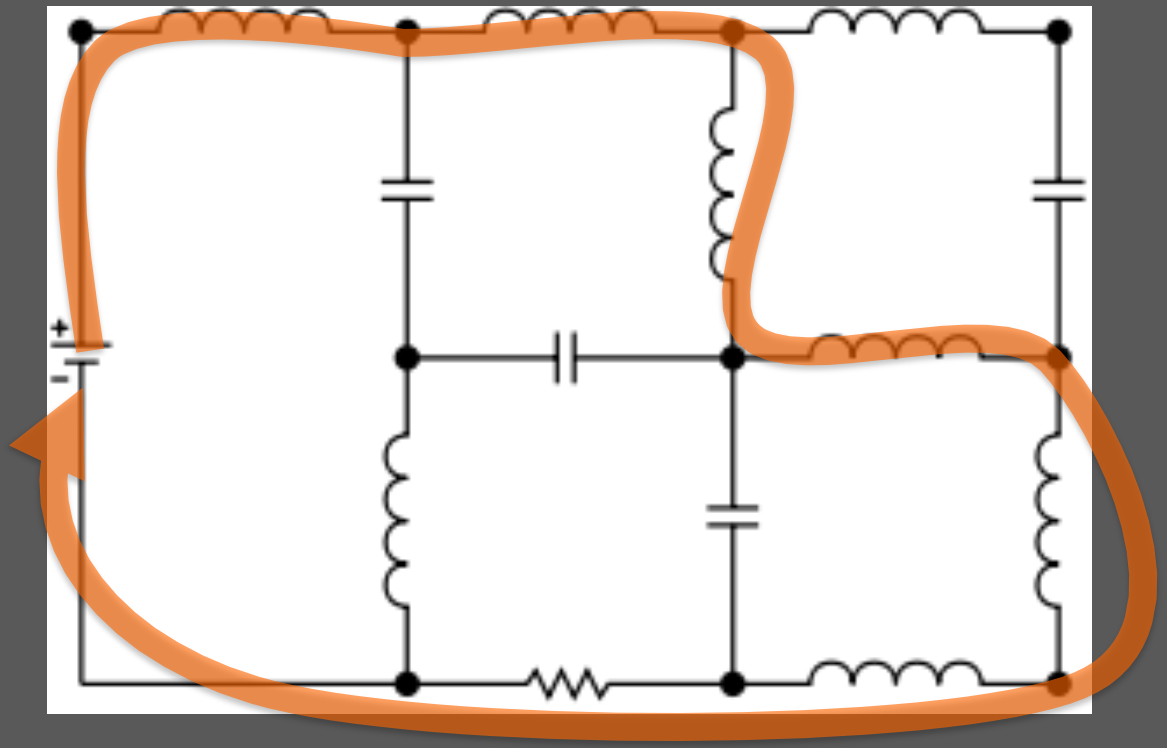
通信ネットワーク

交通システム



ネットワークフローの例２

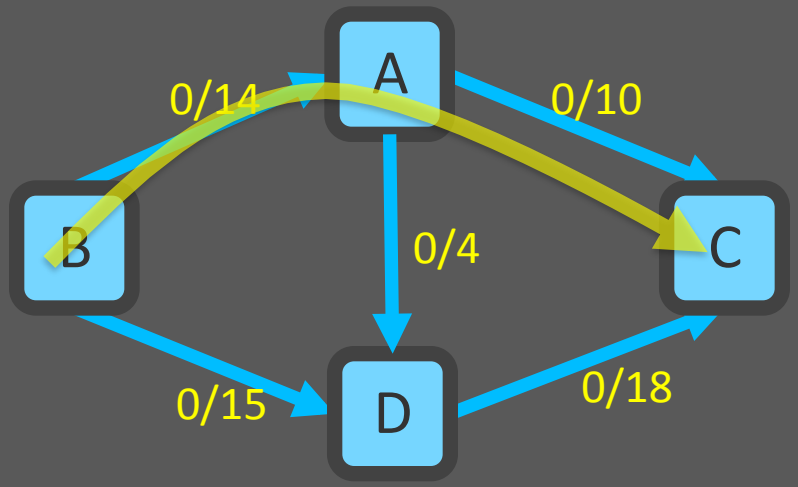
電気回路



そもそも最適化とは？

- BからCに行きたい…
- 黄色は最適な経路？

  ->確かに最短ではあるけど...

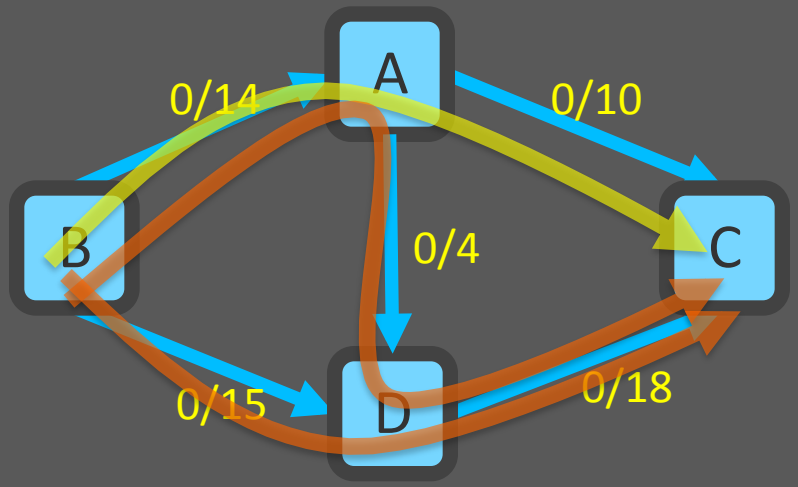


そもそも最適化とは？

- BからCに行きたい…
- 黄色は最適な経路？

  ->確かに最短ではあるけど...
- “何が最適なのか”は目的次第

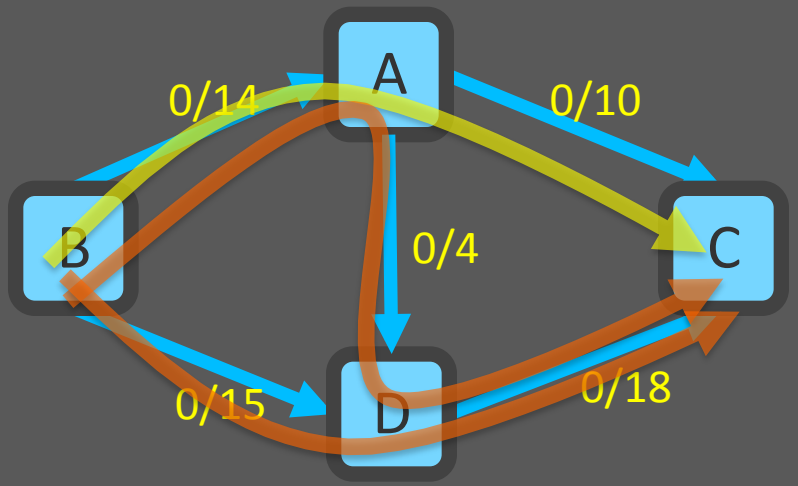
  ここからは流量最大化にフォーカス！



最大フロー問題

最大どれだけ流せるのか？を知りたい

問：BからCまでの流量を最大化せよ



最大フロー問題

最大どれだけ流せるのか？を知りたい

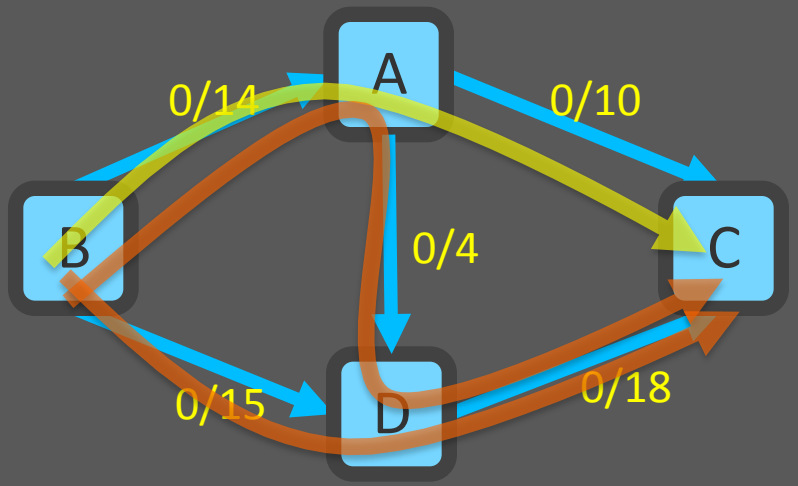
問：BからCまでの流量を最大化せよ

解：２８

B->A->C　　:10

B->A->D->C:4

B->D->C　　:14



Ford-Fulkerson法

最大フローを求めるアルゴリズム

　フロー増加法と呼ばれる

基本的な流れ

1. 残余NWから増加路を探索
2. 増加路にフローを流し，残余NW作成

増加路が発見できなくなったら終了

増加路は幅優先探索などで発見

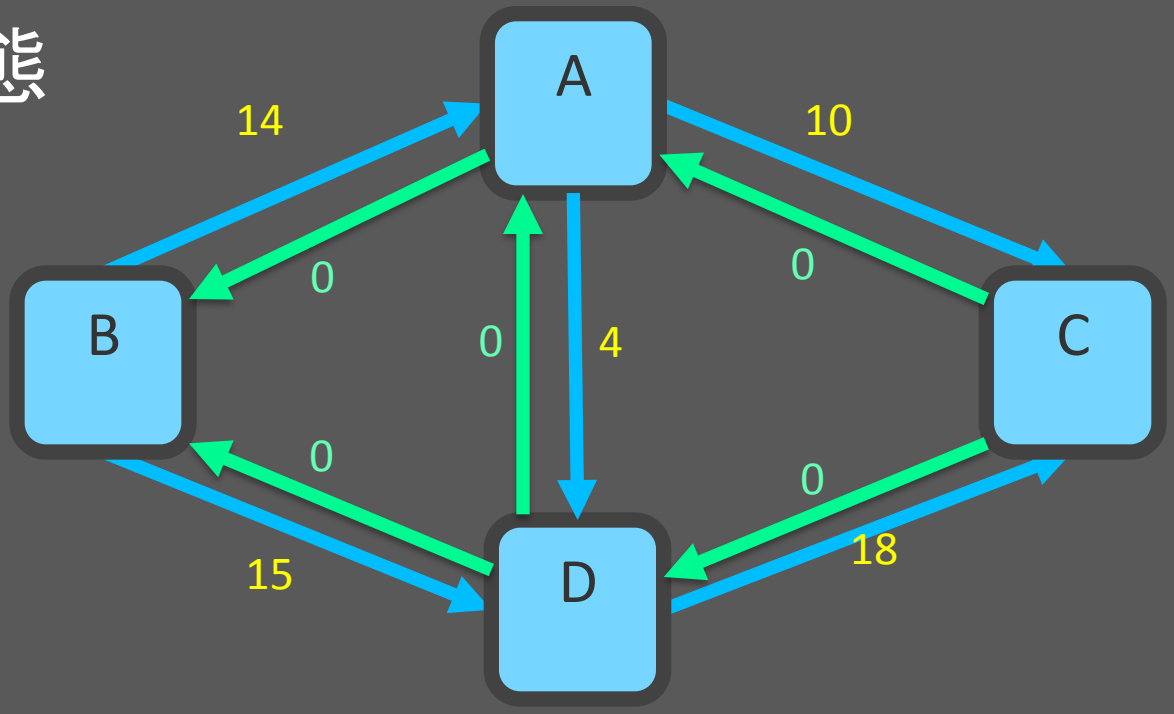
Ford-Fulkerson法

残余NWとは？

-> 現時点で利用可能な容量を示したNW

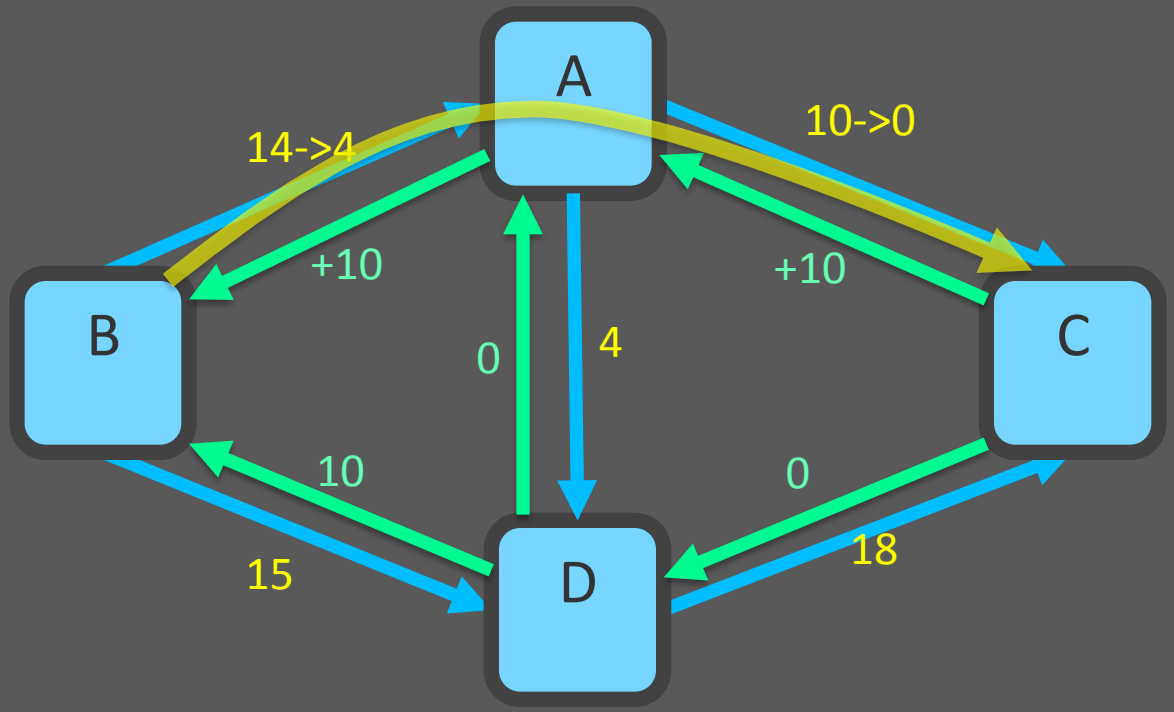
利用済みの後進辺も記述

初期状態



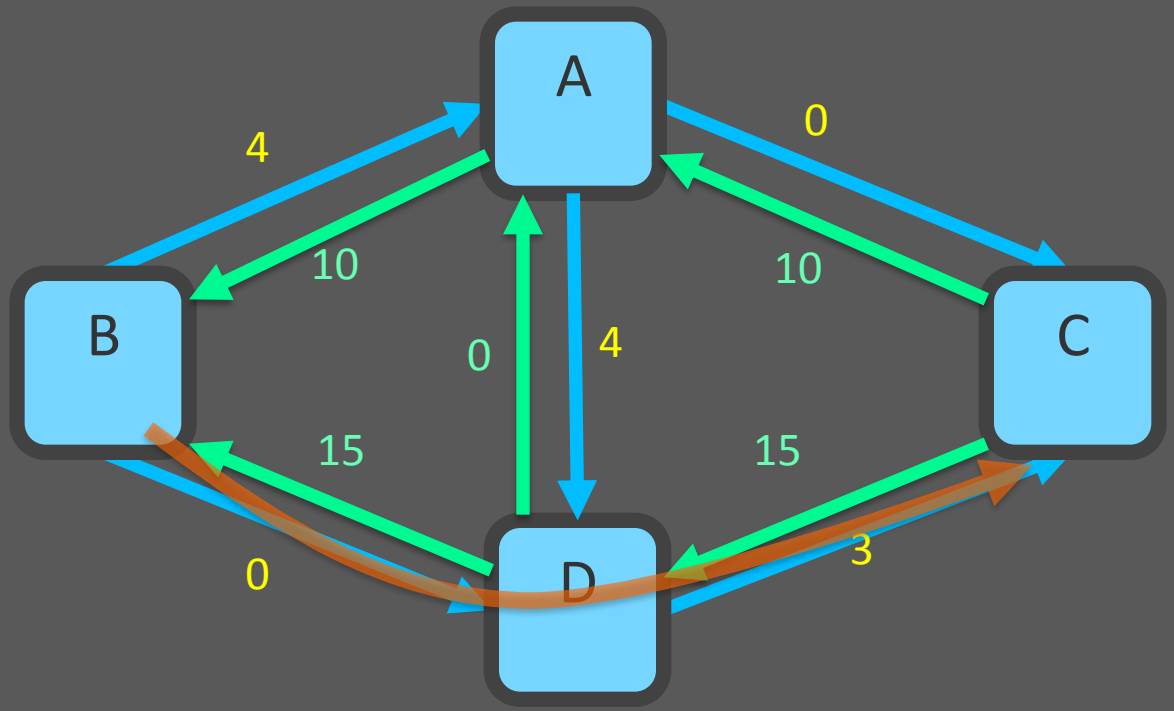
Ford-Fulkerson法

1.B->A->Cのパスに流量を10追加



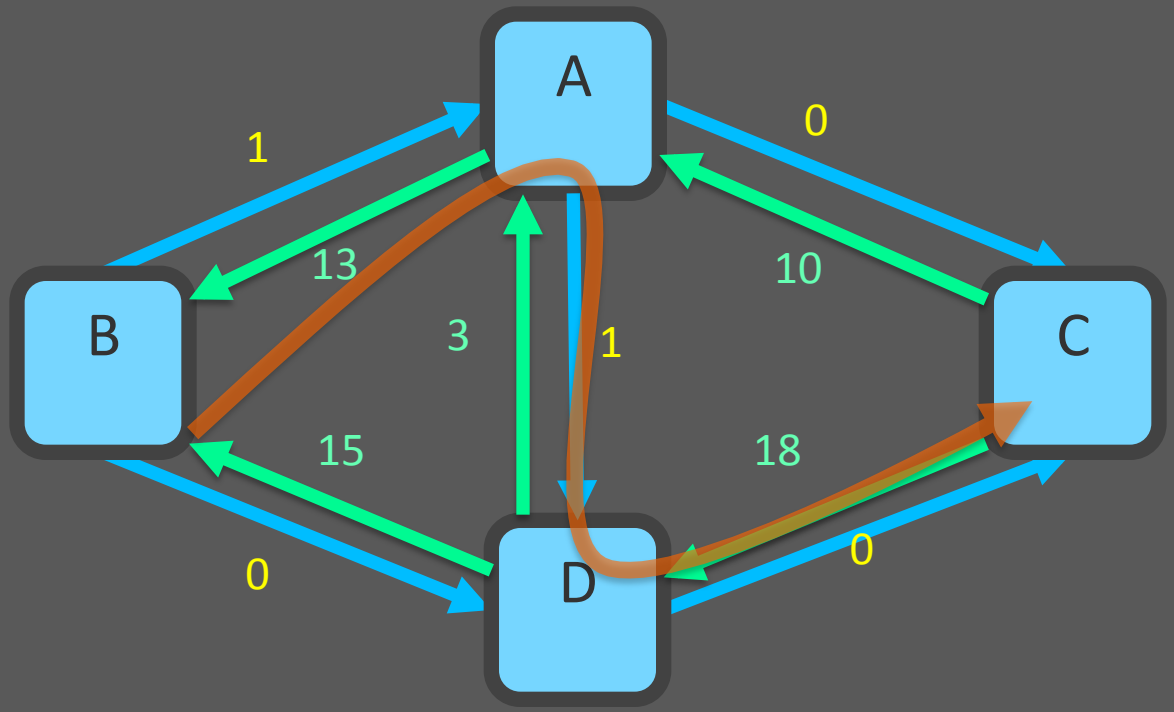
Ford-Fulkerson法

2.B->D->Cのパスに流量を15追加



Ford-Fulkerson法

3.B->A->D->Cのパスに流量を3追加



Ford-Fulkerson法

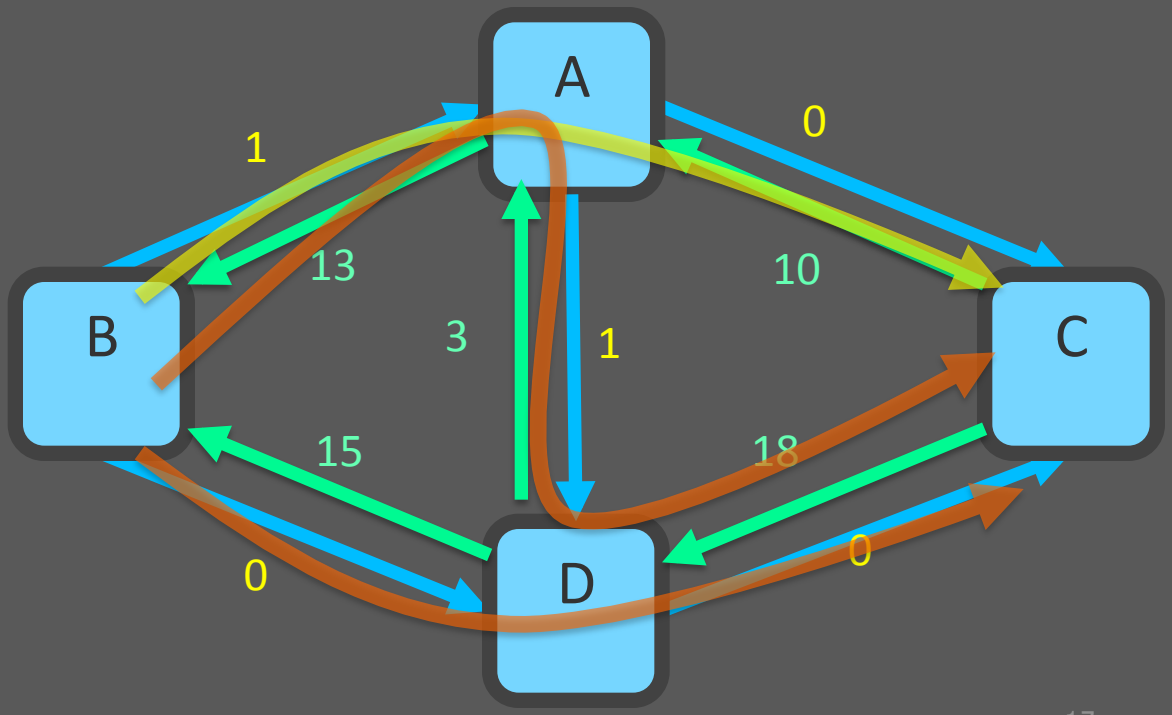
4.B->Cの間に増加路なし，終了

解：28

B->A->C　:10

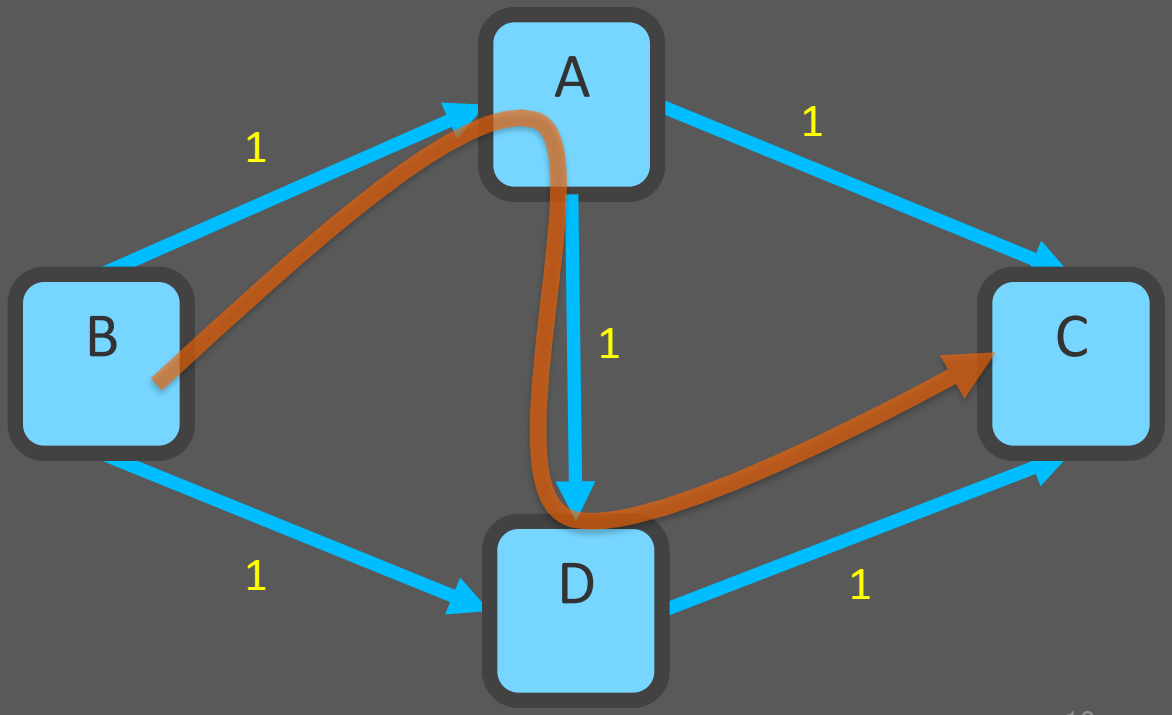
B->A->D->C:4

B->D->C　:14



Ford-Fulkerson法

後進辺を利用しない場合のコーナーケース
一回目の増加路にB->A->D->Cを選んだ場合



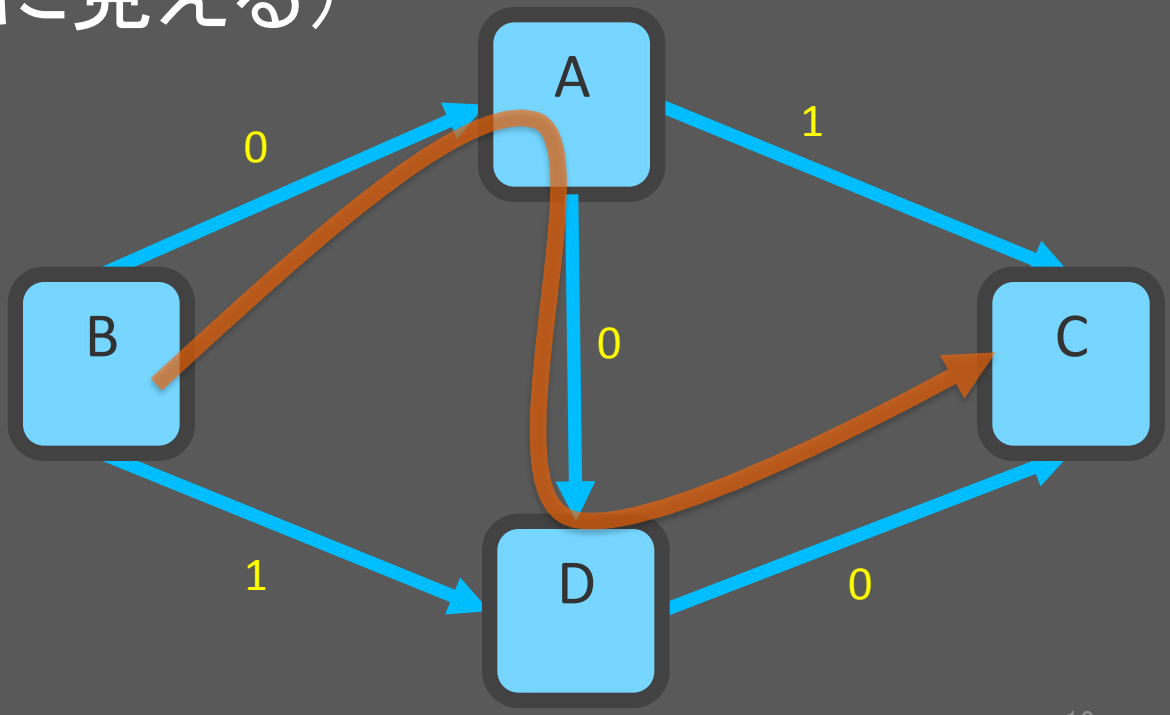
Ford-Fulkerson法

後進辺を利用しない場合のコーナーケース

一回目の増加路にB->A->D->Cを選んだ場合

=>増加道なし(に見える)

最大流1？

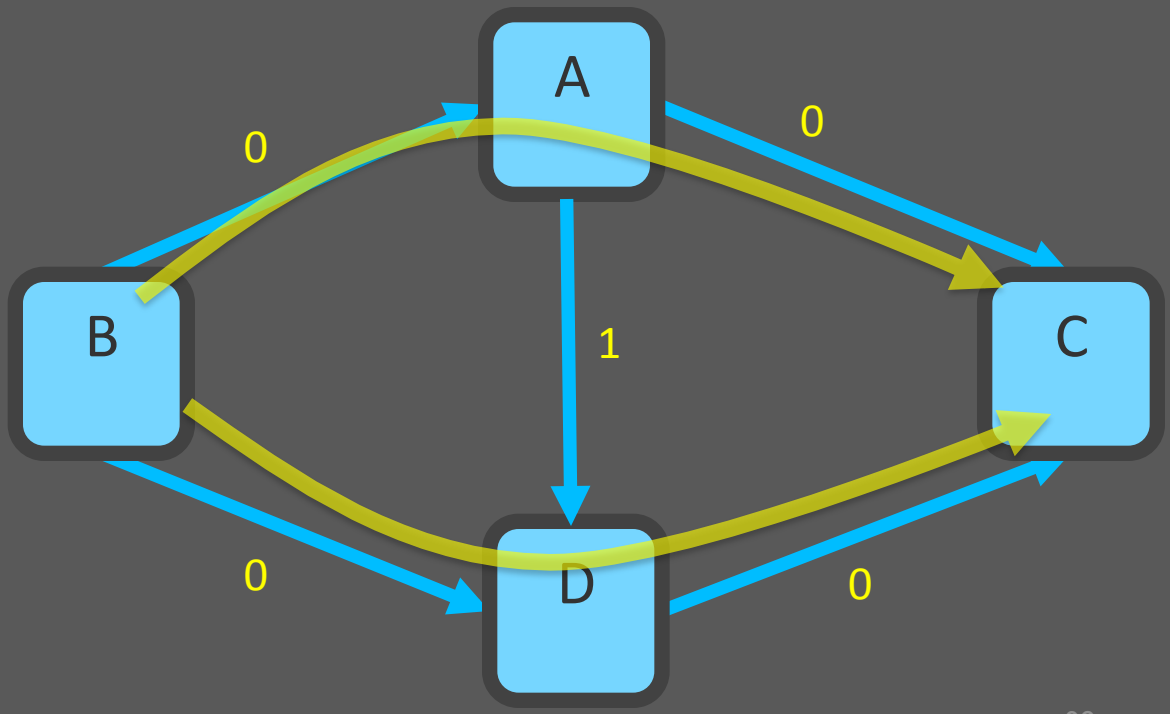


Ford-Fulkerson法

正しい最大流フロー

B->A->C

B->D->C　の計2

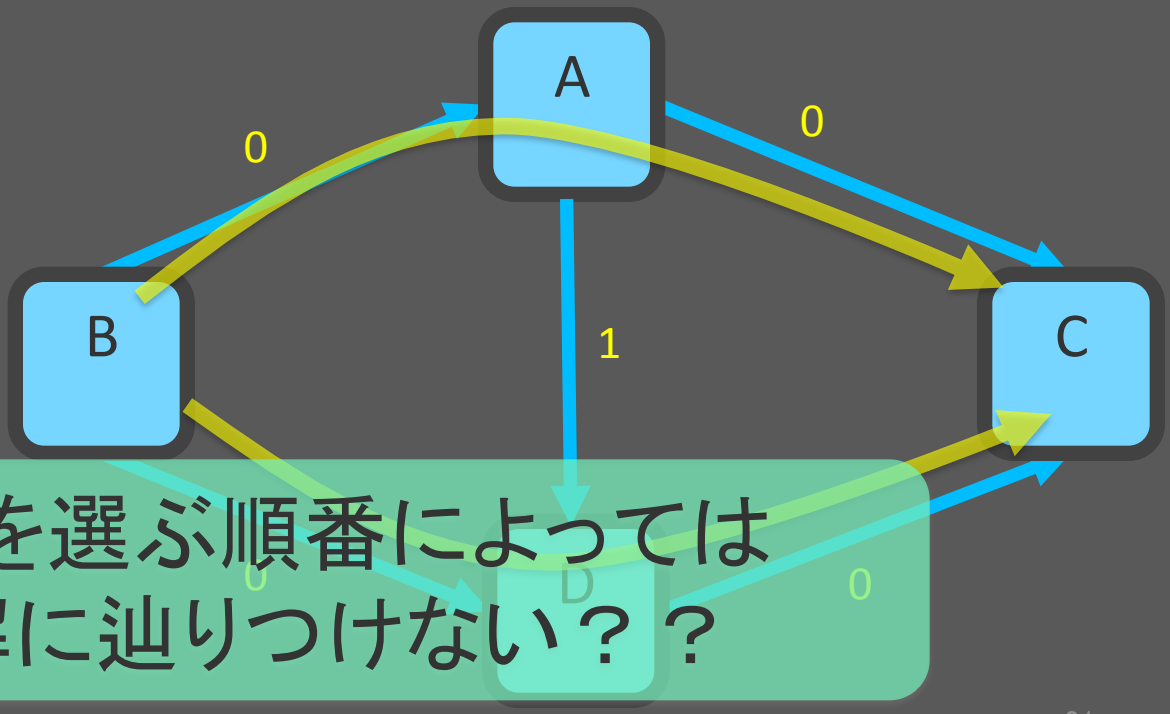


Ford-Fulkerson法

正しい最大流フロー

B->A->C

B->D->C　の計2

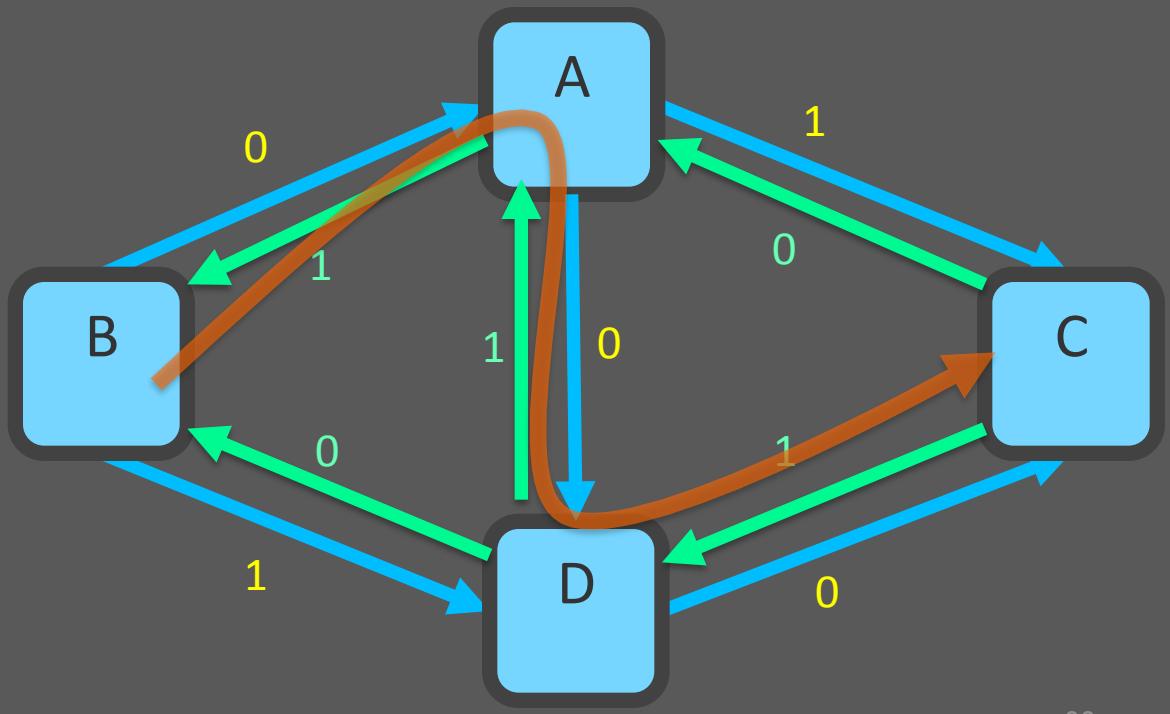


増加路を選ぶ順番によっては
最適解に辿りつけない？？

Ford-Fulkerson法

後進辺の役割

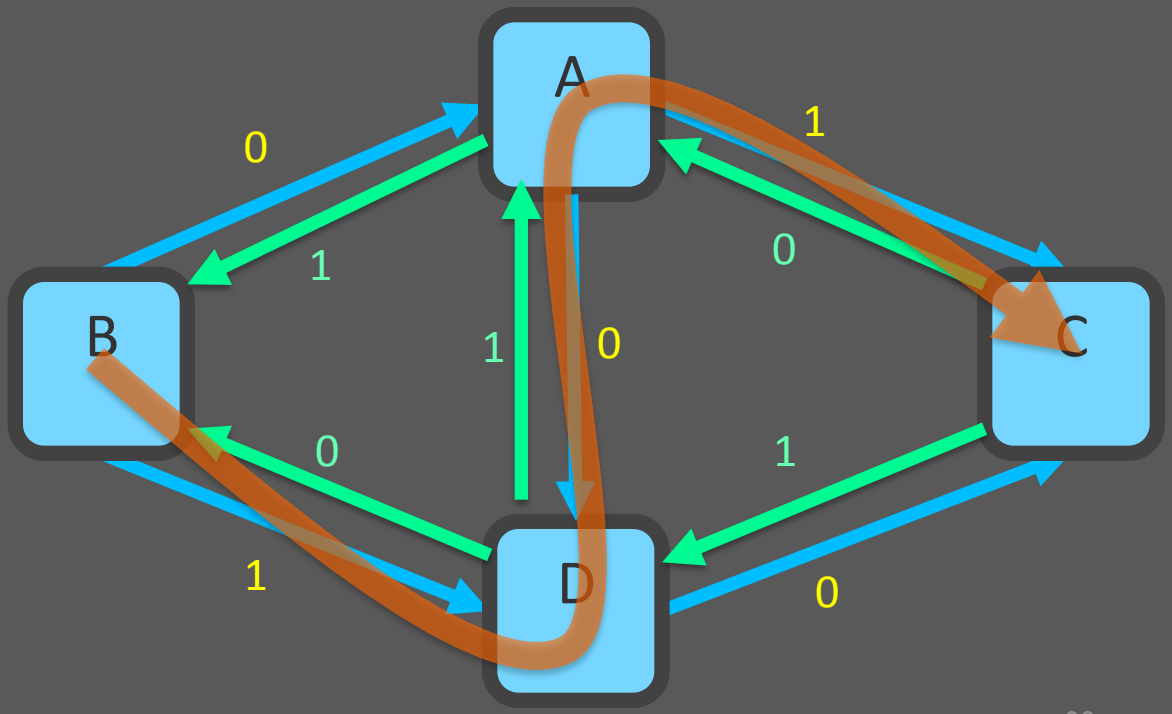
1.最初にB->A->D->Cを選択してしまった場合



Ford-Fulkerson法

後進辺の役割

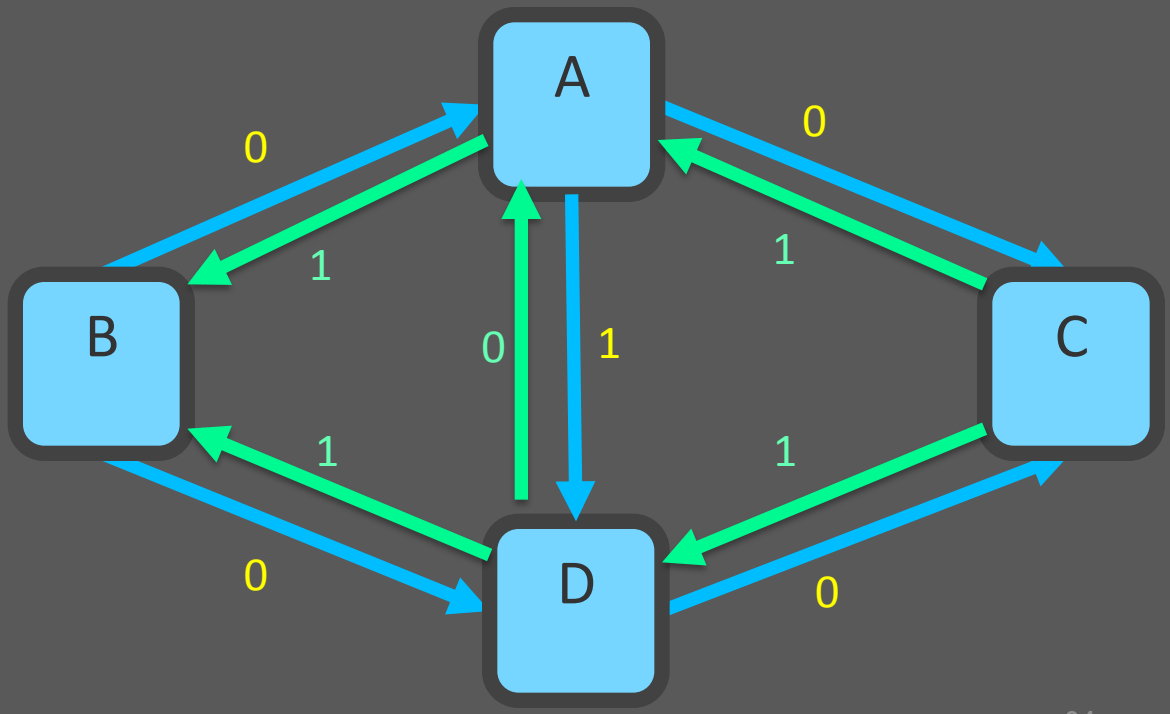
2.後進辺を含めると増加路が存在する！



Ford-Fulkerson法

後進辺の役割

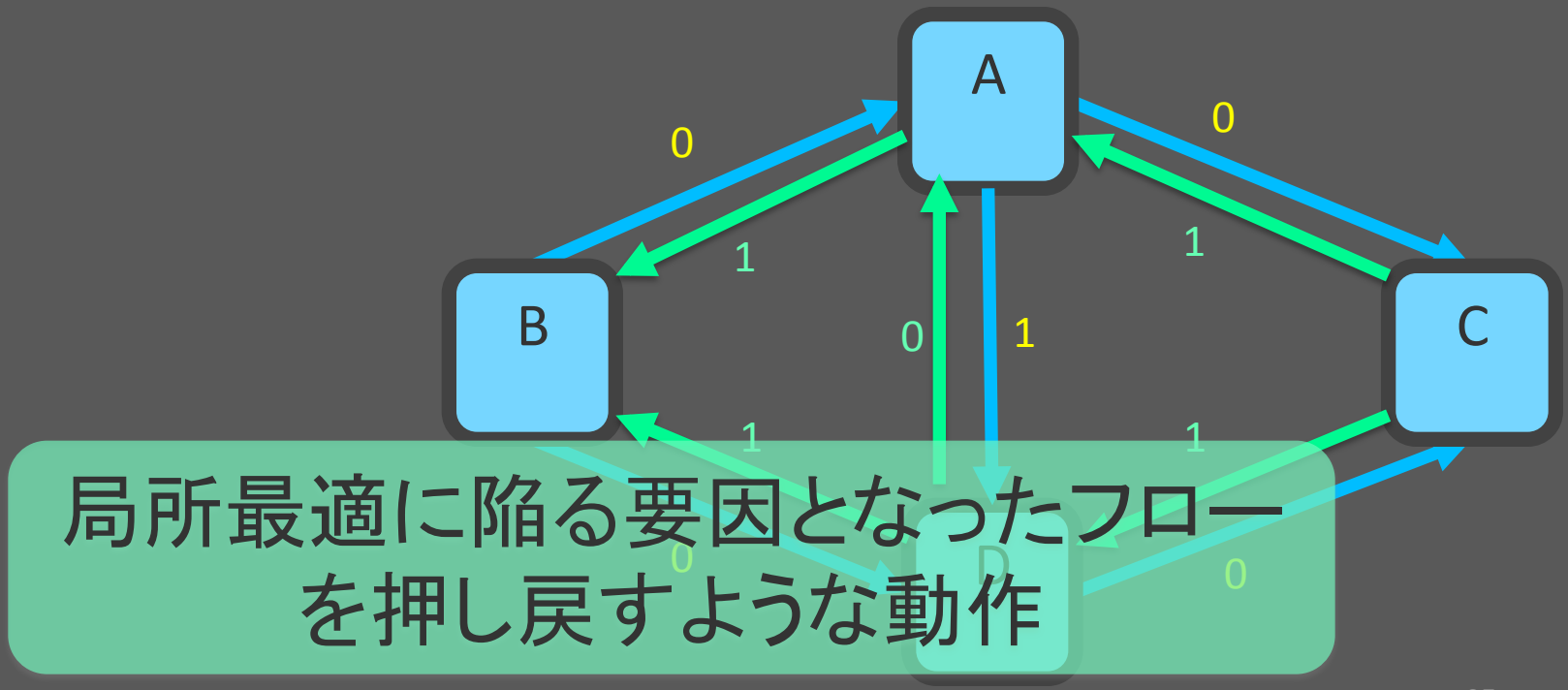
3.後進路を辿ってフローを増加->最大フロー



Ford-Fulkerson法

後進辺の役割

3.後進路を辿ってフローを増加->最大フロー



線形計画法とは

ここまで例はグラフの構造を利用していた

派生問題も全てアルゴリズムを考える？

disjoint path,minimum cost,etc...

Linear Programming(LP)

制約条件群（一次式）を満たす
目的関数（一次式）を最大/最小化する
変数を探す手法！

線形計画法 例

- 生産計画
  - 利益を最大化したい！
- 定式化すると
  - 制約
    1. x + 2y <=10
    2. 4x <= 16
    3. x >= 0 , y >= 0
  - 目的関数
    - maximize 2x+y

	製品X	製品Y	使える量
原料A	1	2	10
原料B	4	0	16
利益	2	1	

AとBを何個ずつ作ろう？

線形計画法 例

- 生産計画
  - 利益を最大化したい！
- 定式化すると
  - 制約
    1. x + 2y <=10
    2. 4x <= 16
    3. x >= 0 , y >= 0
  - 目的関数
    - maximize 2x+y

	製品X	製品Y	使える量
原料A	1	2	10
原料B	4	0	16
利益	2	1	

AとBを何個ずつ作ろう？

この形に落とし込めばsolverで解ける！

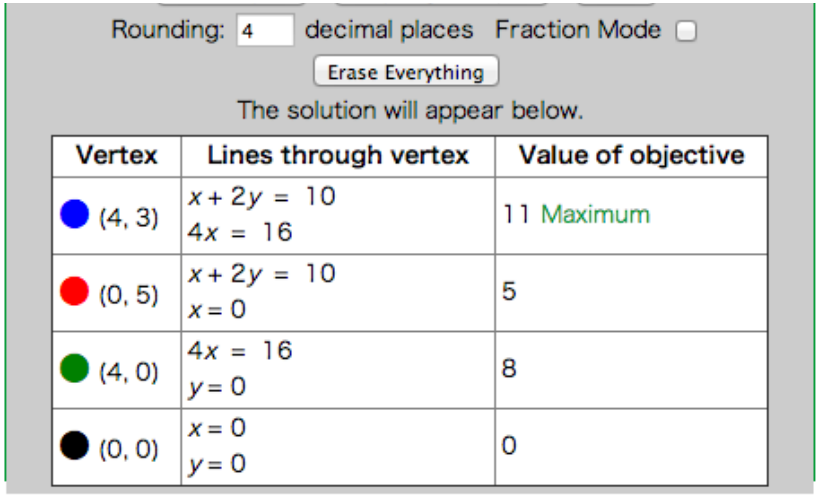
線形計画法 例

制約

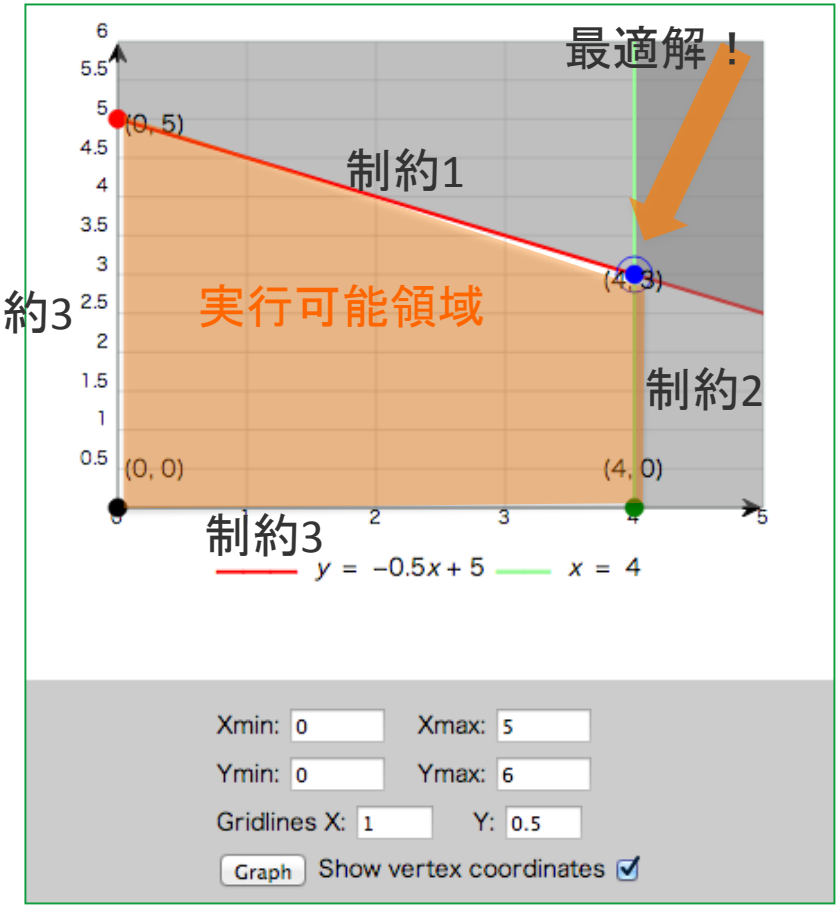
1. x + 2y <=10
2. 4x <= 16
3. x >= 0 , y >= 0

目的関数

maximize 2x+y



Vertex	Lines through vertex	Value of objective
(4, 3)	$x + 2y = 10$ $4x = 16$	11 Maximum
(0, 5)	$x + 2y = 10$ $x = 0$	5
(4, 0)	$4x = 16$ $y = 0$	8
(0, 0)	$x = 0$ $y = 0$	0



http://www.zweigmedia.com/utilities/lpg/index.html

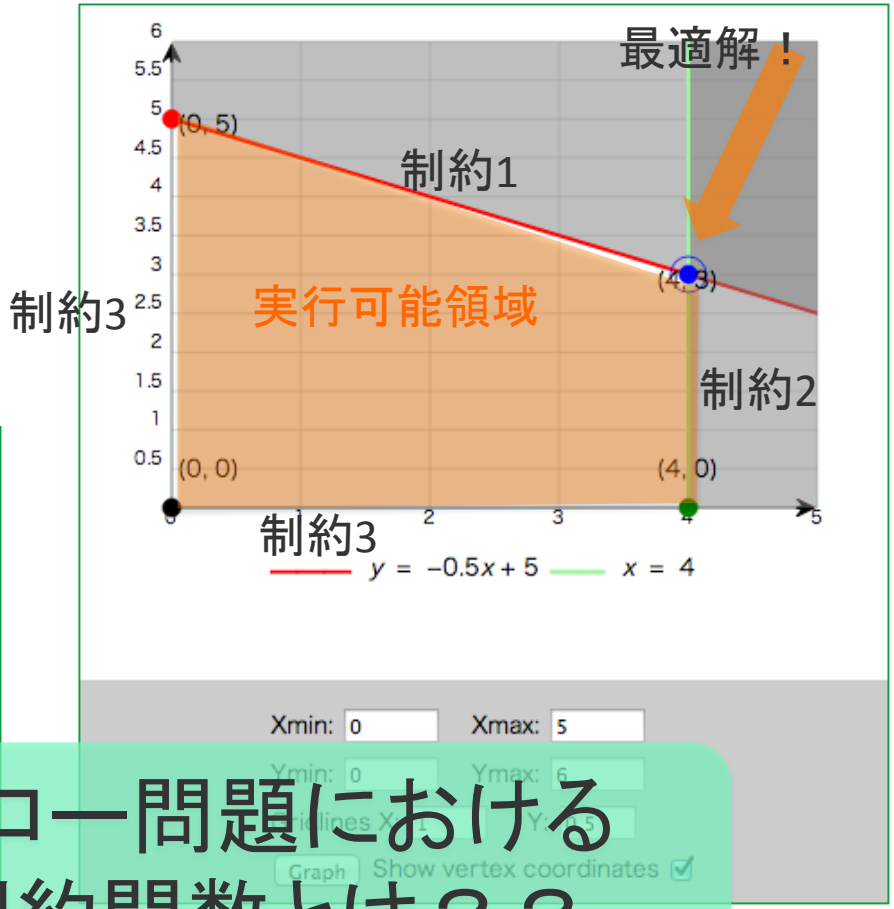
線形計画法 例

制約

1. x + 2y <=10
2. 4x <= 16
3. x >= 0 , y >= 0

目的関数

maximize 2x+y



Rounding: 4 decimal places Fraction Mode

Erase Everything

The solution will appear below.

Vertex	Lines through vertex	Value of objective
(4, 3)	$x + 2y = 10$ $4x = 16$	11 Maximum
(0, 5)	$x + 2y = 10$ $x = 0$	5
(4, 0)	$4x = 16$ $y = 0$	0
(0, 0)	$x = 0$ $y = 0$	0

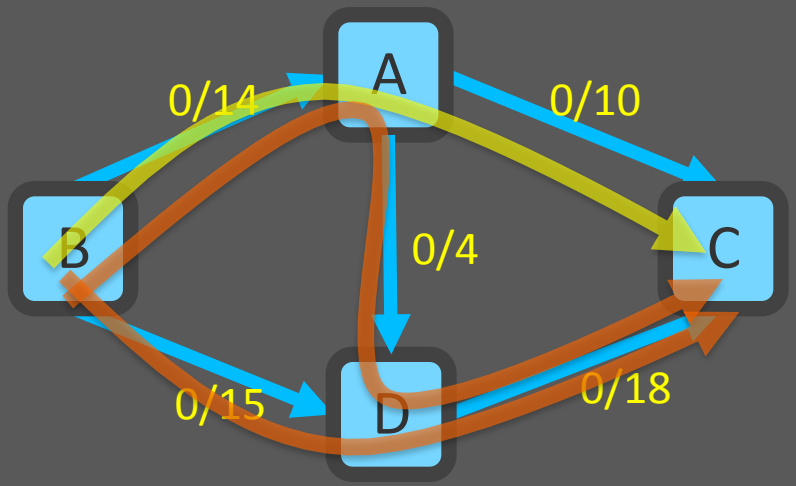
ネットワークフロー問題における
目的関数／制約関数とは？？

http://www.zweigmedia.com/utilities/lpg/index.html

ネットワークフロー問題の定式化

BからCまでの最大流問題

- 最大化したいもの：B->Cのフロー流量
- 制約:ネットワークフローであること



ネットワークフロー問題の定式化

目的関数

SourceNodeにおける出力フロー最大

制約条件

1. エッジ上のフローはエッジの容量以下
2. 入出力フロー量の一致

ネットワークフロー問題の定式化

目的関数

SourceNodeにおける出力フロー最大

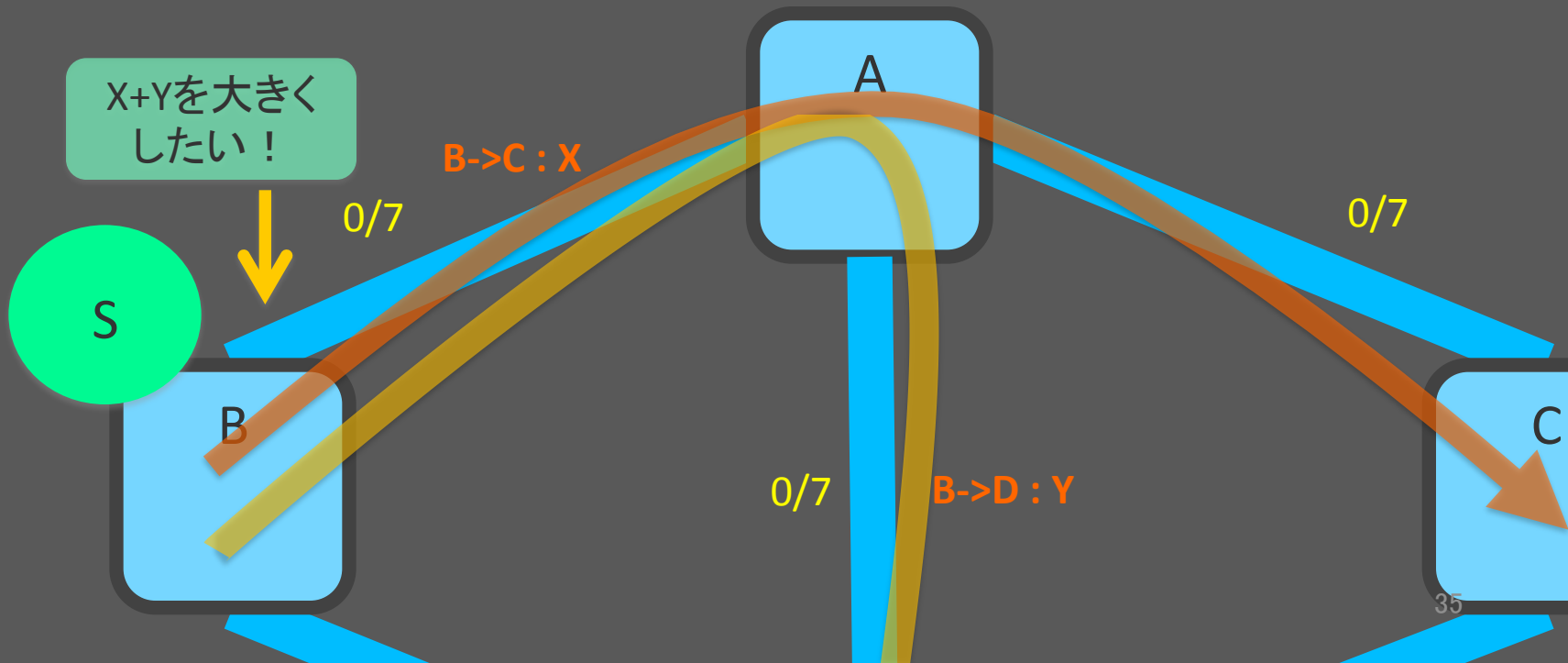
制約条件

1. エッジ上のフローはエッジの容量以下
2. 入出力フロー量の一致

ネットワークフロー問題の定式化

目的関数：SourceNodeにおいて

出次エッジのフロー量最大



ネットワークフロー問題の定式化

x_e edge e上のフロー量

$\delta+(v)$ vから出るエッジeの集合

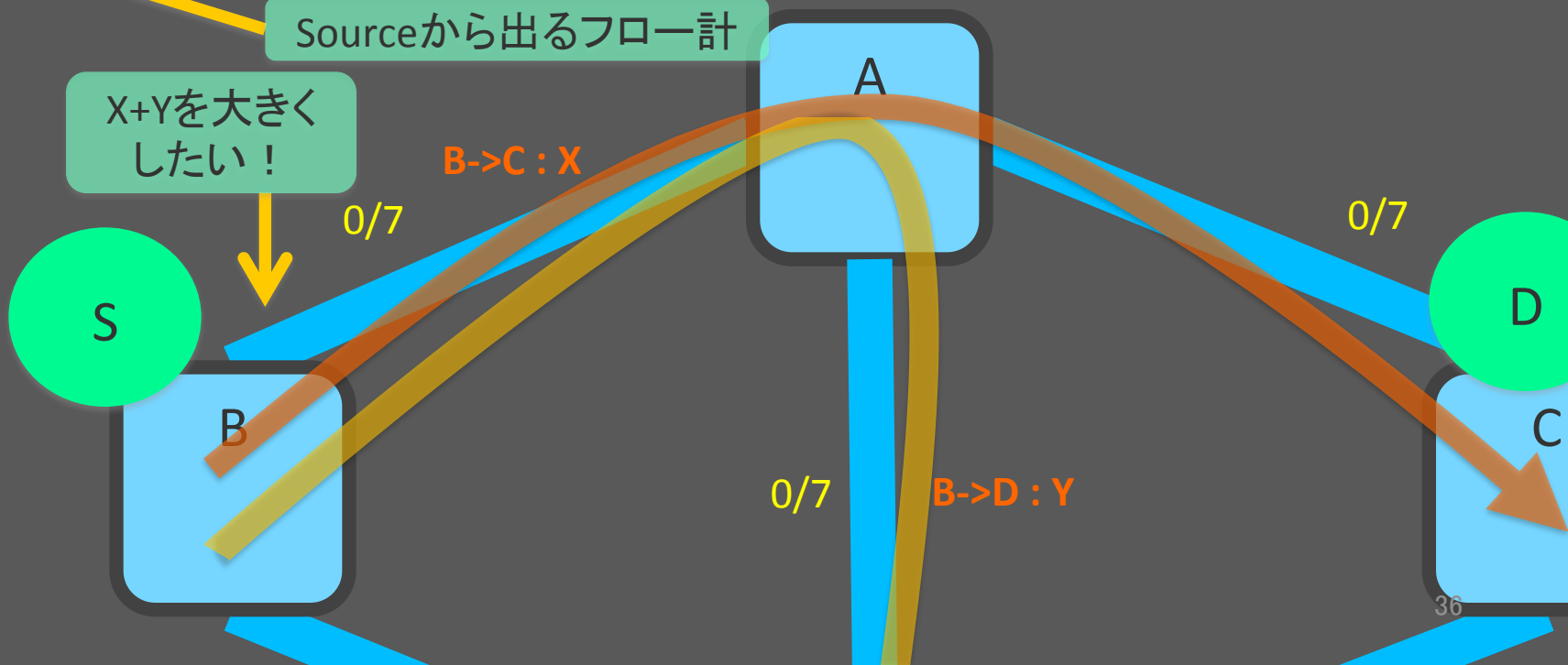
目的関数

$$\sum_{e\in\delta+(v)} x_e - \sum_{e\in\delta-(v)} x_e$$

Sourceに入るフロー計

この差がSourceからの直接的な流出量

Sourceから出るフロー計



ネットワークフロー問題の定式化

目的関数

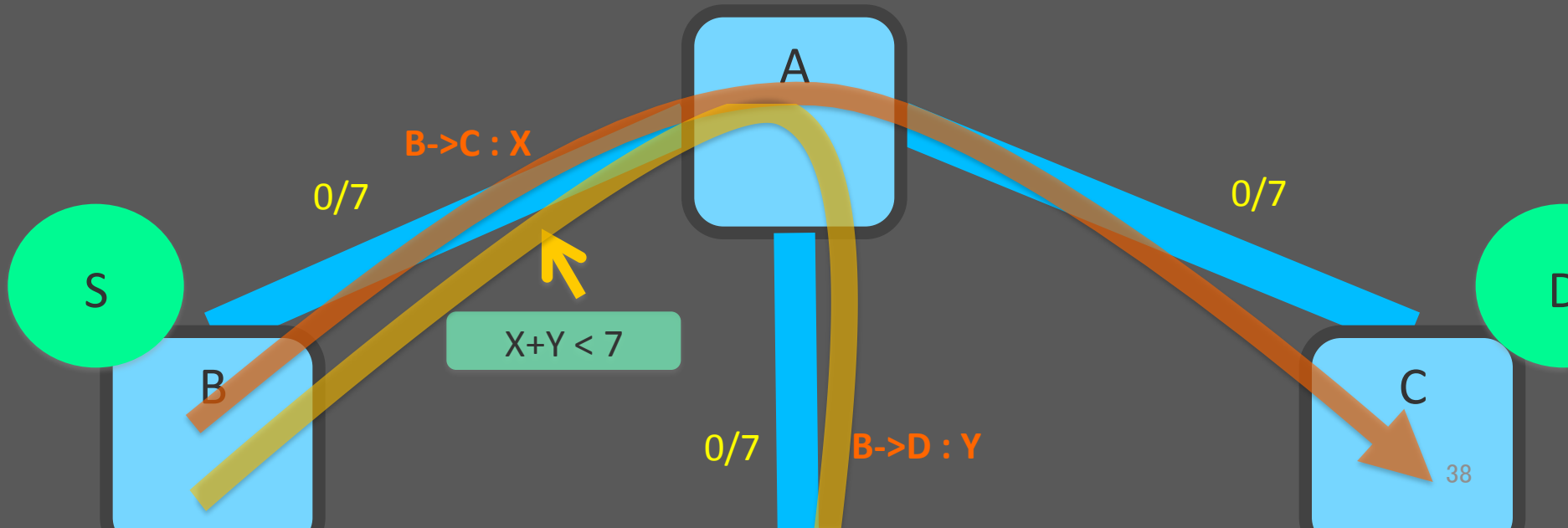
SourceNodeにおける出力フロー最大

制約条件

1. エッジ上のフローはエッジの容量以下
2. 入出力フロー量の一致

ネットワークフロー問題の定式化

制約1.全てのエッジ上について
その上を流れるフローの合計はは
エッジの容量を超えない
容量制約則と呼ぶ



ネットワークフロー問題の定式化

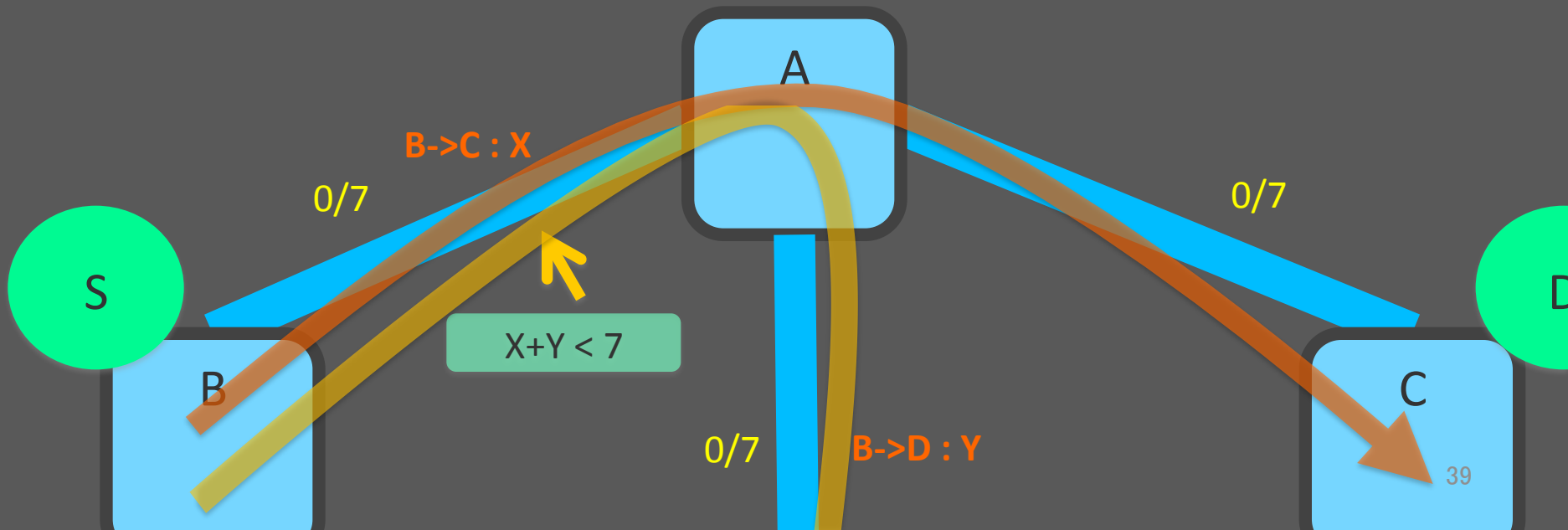
x_e edge e上のフロー量

c_e edge eのcapacity

容量制約

$$0 \leq x_e \leq c_e$$

エッジを流れるフローが容量を超えないと言っているだけ



ネットワークフロー問題の定式化

目的関数

SourceNodeにおける出力フロー最大

制約条件

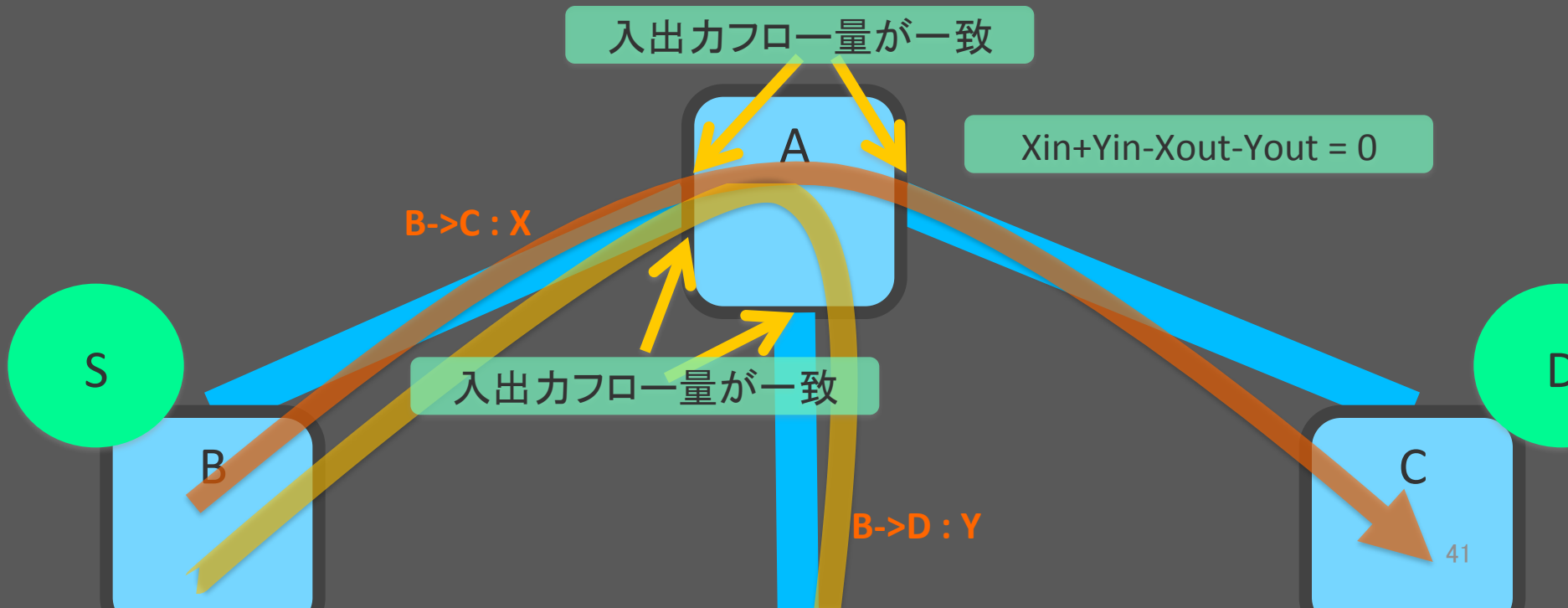
1. エッジ上のフローはエッジの容量以下
2. 入出力フロー量の一致

ネットワークフロー問題の定式化

制約2.S-Dでは無いノードについて

入出力フロー量が一致する

フロー保存則と呼ぶ



ネットワークフロー問題の定式化

x_e edge e上のフロー量

$\delta+(v)$ vから出るエッジeの集合

$\delta-(v)$ vに入るエッジeの集合

フロー保存則

$$\sum_{e\in\delta+(v)} x_e - \sum_{e\in\delta-(v)} x_e = 0$$

中継ノードについて
入力/出力の差分が0

入出力フロー量が一致

A

Xin+Yin-Xout-Yout = 0

B->C : X

S

入出力フロー量が一致

B

B->D : Y

C

D

ネットワークフロー問題の定式化

まとめ

目的関数

$$\sum_{e \in \delta+(v)} x_e - \sum_{e \in \delta-(v)} x_e$$

Sourceに入るフロー計

Sourceから出るフロー計

この差がSourceからの直接的な流出量

制約条件

1. $0 \leq x_e \leq c_e$

エッジを流れるフローが容量を超えないと言っているだけ

2. $\sum_{e \in \delta+(v)} x_e - \sum_{e \in \delta-(v)} x_e = 0$

目的関数と形同じS/Dノード以外はフローを中継するのみ

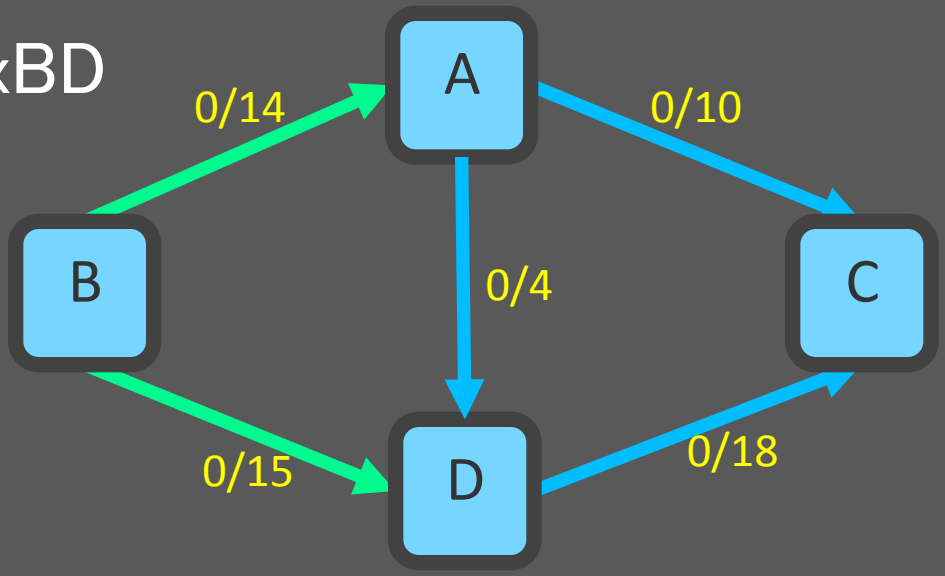
Glpkで動かしてみよう

GNU Linear Programming Kit

- 線形計画法をとくためのSolver Kit

目的関数

– Maximize：xBA + xBD



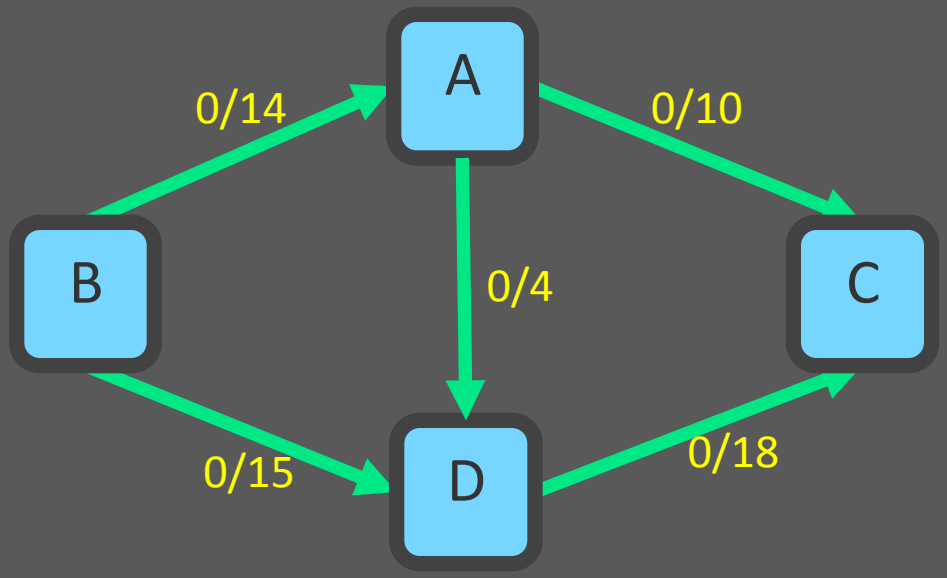
Glpkで動かしてみよう

容量制約

$$0 \leq x_e \leq c_e$$

xBA <= 10 , xBD <= 10,
xAD <= 10, xAC<=10 ,
xDC <= 10

xBA >= 0 , xBD >= 0,
xAD >= 0, xAC >= 0 ,
xDC >= 0



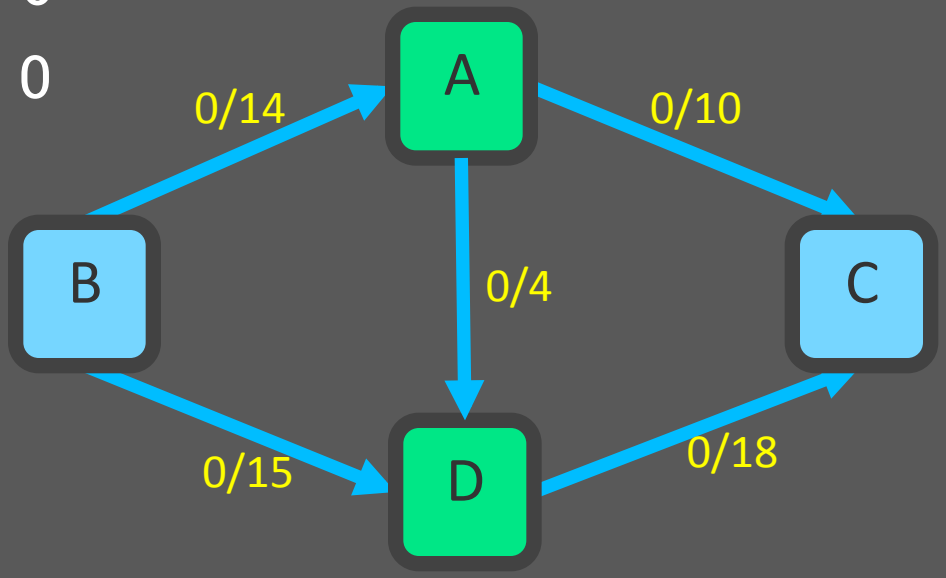
Glpkで動かしてみよう

フロー保存則

$$\sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0$$

xDC − xBD − xAD = 0

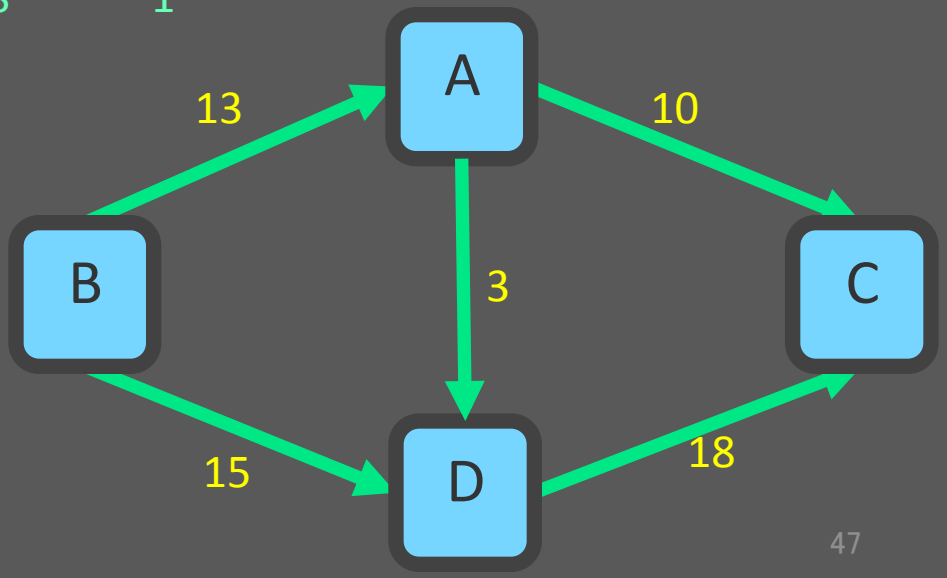
xAD + xAC − xBA = 0



実行例

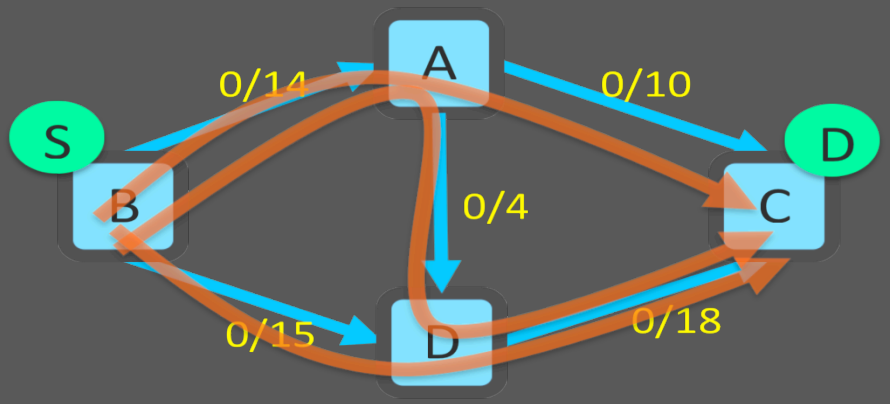
Objective: TRAFFIC = 28 (MAXimum)

No.	Column name	St	Activity	Lower bound	Upper bound	Marginal
1	k1	B	28			
2	uBA	B	13	0	14	
3	uBD	NU	15	0	15	< eps
4	uAD	B	3	0	4	
5	uAC	NU	10	0	10	1
6	uDC	NU	18	0	18	1
7	xBA	B	13	0		
8	xBD	B	15	0		
9	xAD	B	3	0		
10	xAC	B	10	0		
11	xDC	B	18	0		



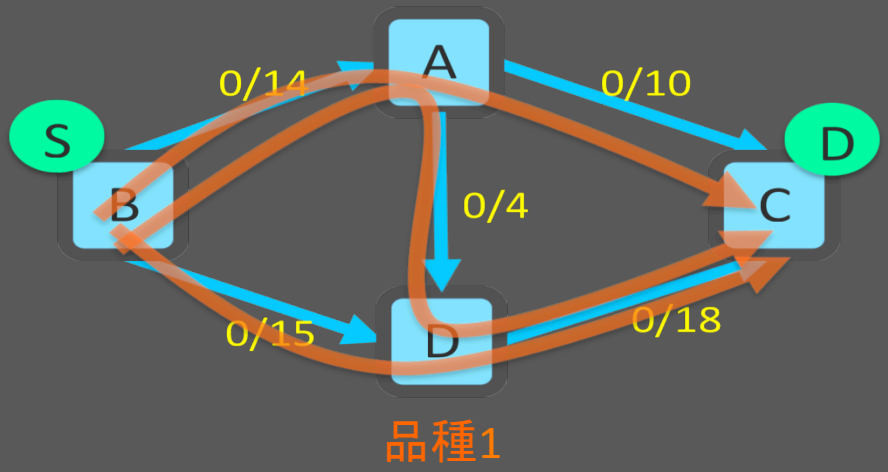
多品種フロー

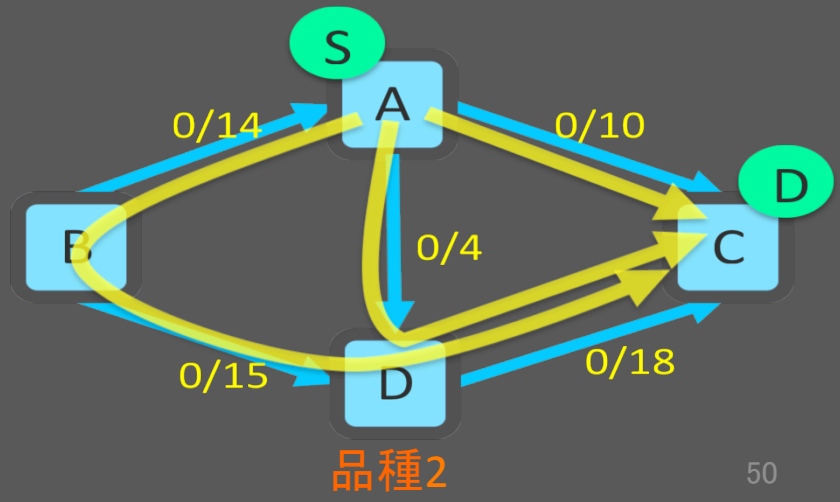
- 品種(commodity)とは発着地のペアの事
- さきほど解いた多品種問題は1品種
  - 品種(B->C)



多品種フロー

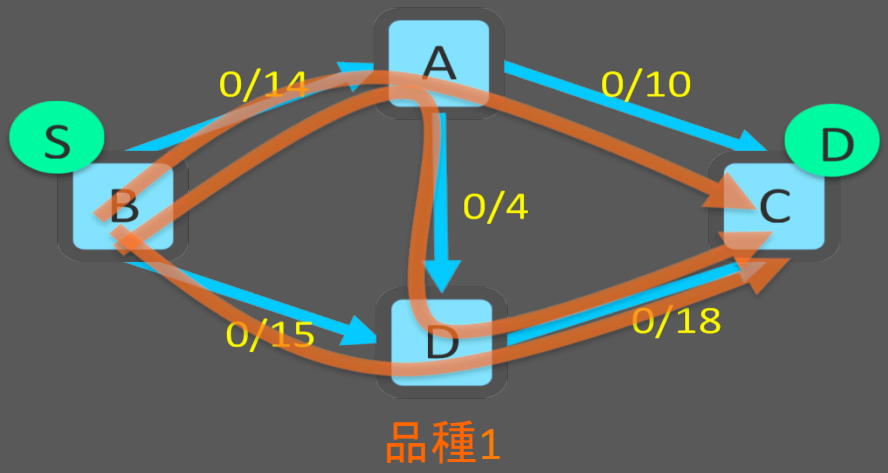
- 品種とは発地，着地のペアの事
- さきほど解いた多品種問題は1品種
  - 品種(B->C)
- 複数の品種に拡張(multi commodity)
  - 例えば(B->C)と(A->C)の2品種



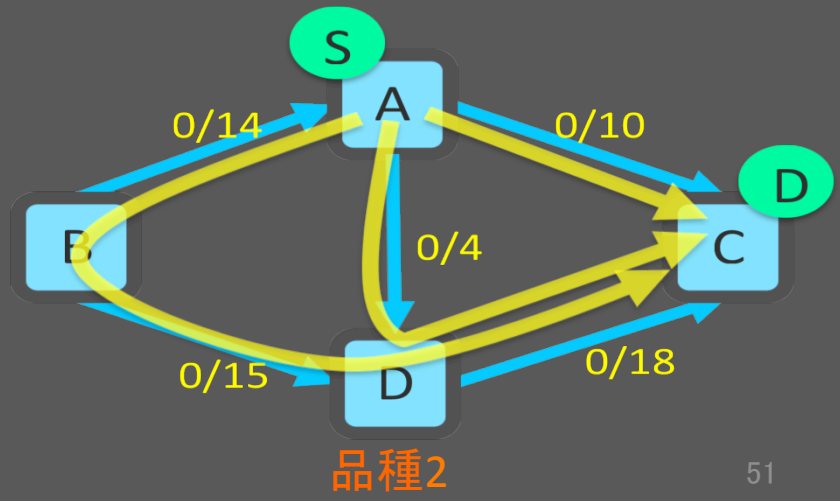


多品種フローの定式化

- 今回目指す最適性
  - 複数品種のフローの合計を最大化
- 目的関数
  - 1品種：Sourceにおける出力フロー最大
  - 多品種：Source群における出力フロー合計最大



品種1



品種2

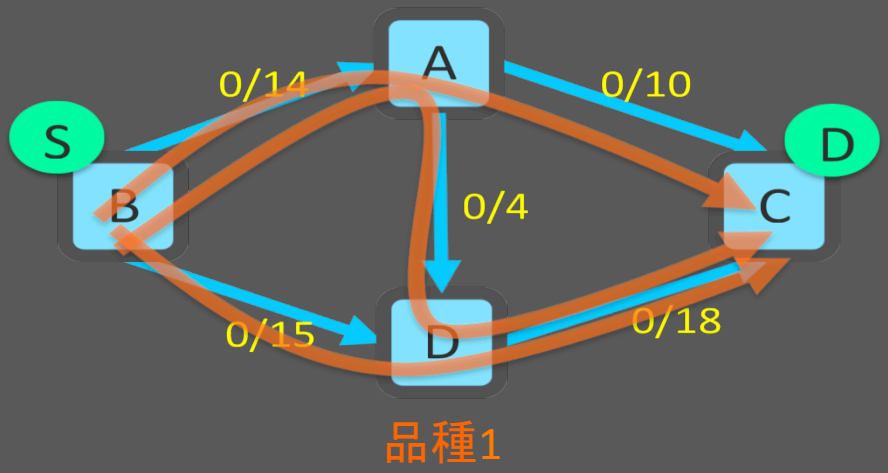
多品種フロー問題の定式化

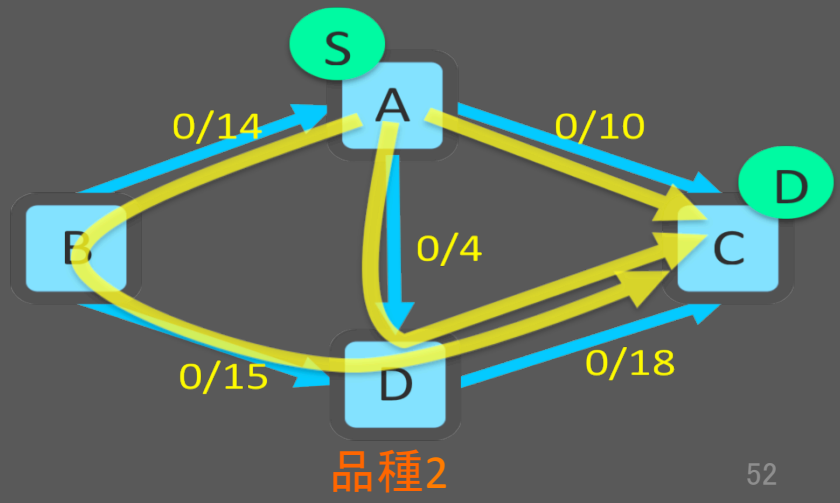
目的関数

Source群における出力フロー合計最大

制約条件（全ての品種の合計について）

1. エッジ上のフローはエッジの容量以下
2. 入出力フロー量の一致

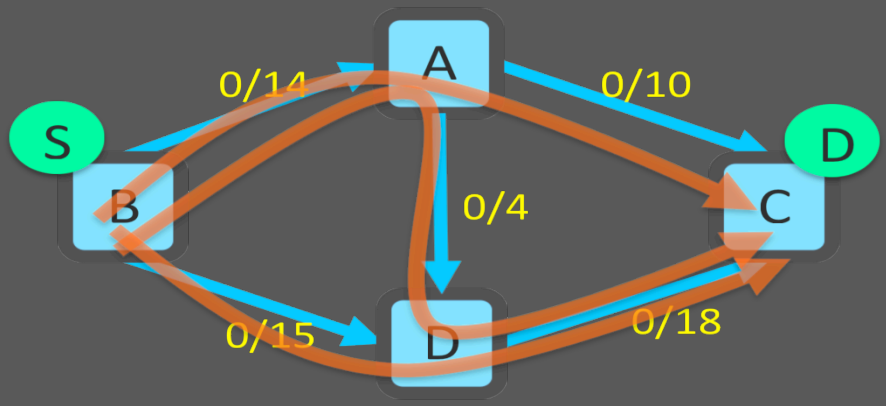




多品種フロー問題の定式化

目的関数(1品種)

$$\sum_{e \in \delta+(v)} x_a - \sum_{e \in \delta-(v)} x_a$$



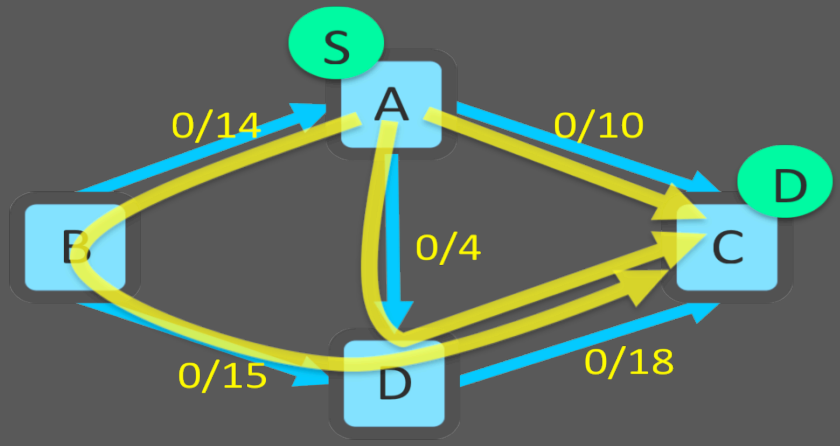
品種1

目的関数(k品種)

$$\sum_{i=1}^{k} \left(\sum_{e \in \delta+(v)} x_a - \sum_{e \in \delta-(v)} x_a \right)$$

1品種の時と同じ

K品種分の
合計を最大に

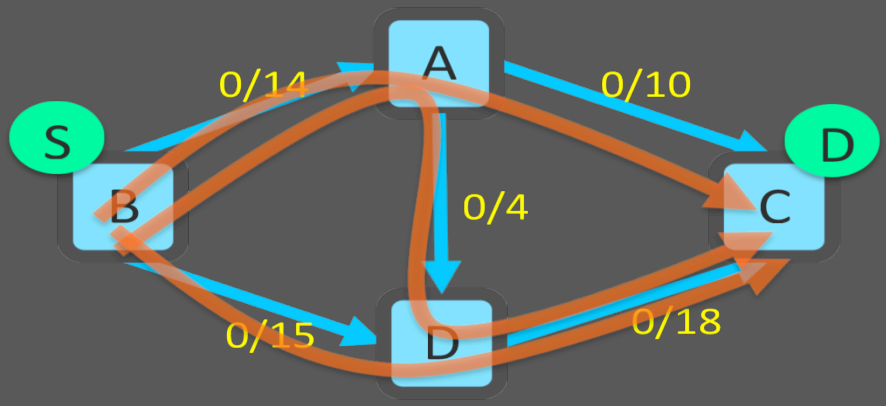


品種2

多品種フロー問題の定式化

容量制約(1品種)

$$0 \le x_e \le c_e$$



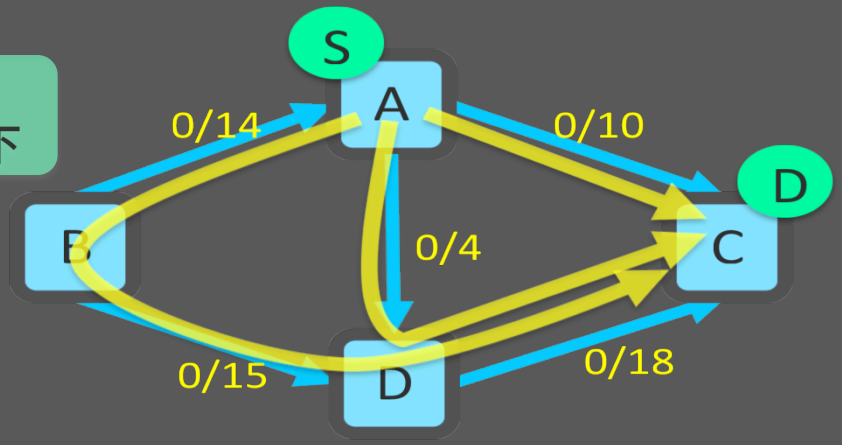
品種1

容量制約(k品種)

$$\sum_{i=1}^{k} x_e^i \le c_e$$

$$x_e^i \le c_e$$

K品種分の計が
エッジの容量以下

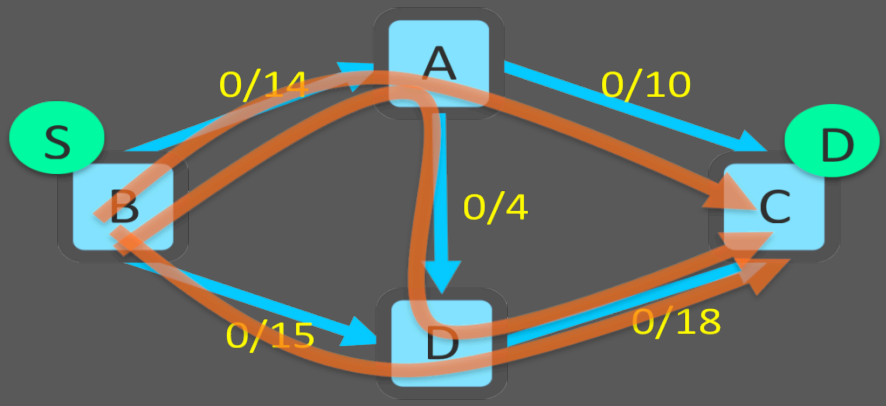


品種2

多品種フロー問題の定式化

フロー保存制約(1品種)

$$\sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0$$



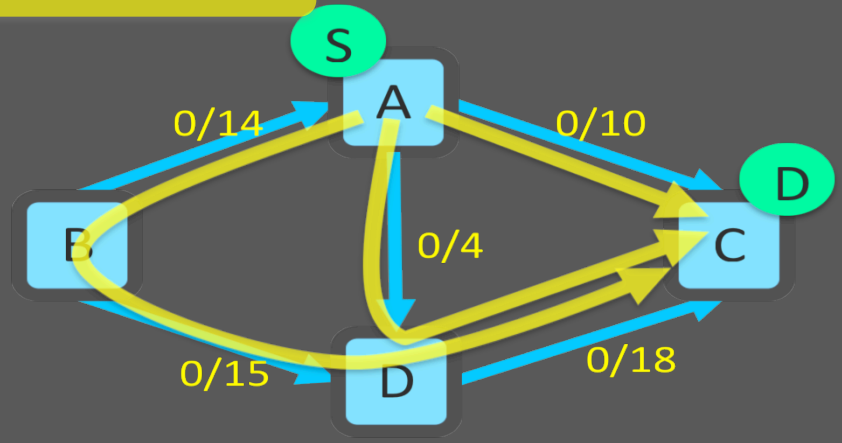
品種1

フロー保存制約(k品種)

$$\sum_{e \in \delta^+(v)} x_e^i - \sum_{e \in \delta^-(v)} x_e^i = 0$$

x_e^i edge e上の品種kのフロー量

各品種ごと、各中継ノードごとにフロー保存則が成り立つ



品種2

多品種フロー，結果

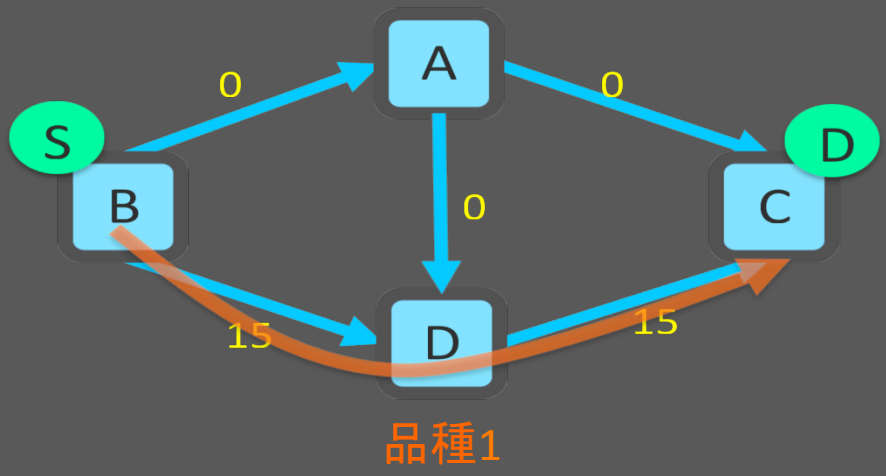
得られた解

目的関数：28

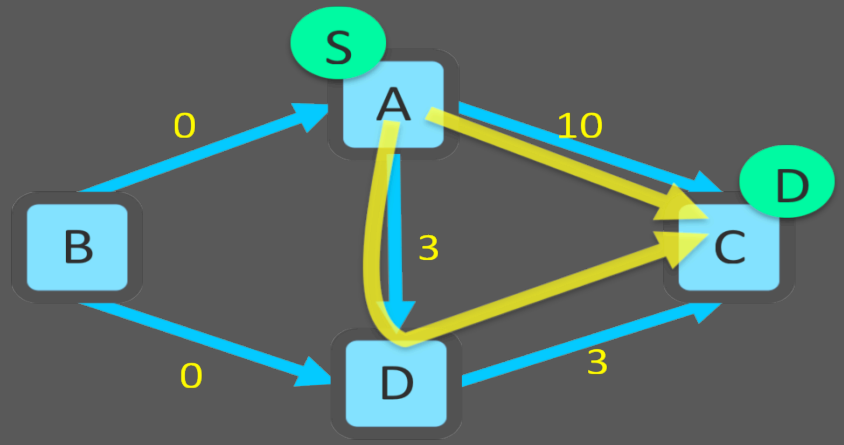
各品種の流量

品種1：15

品種2：13



品種1

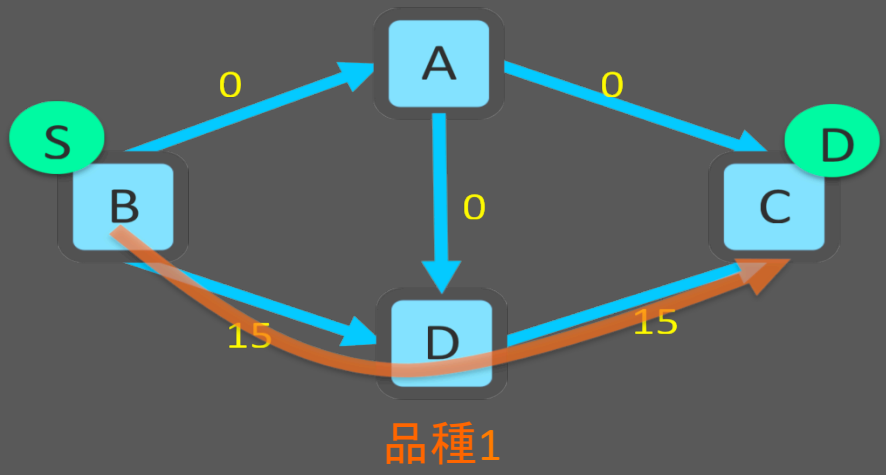


品種2

多品種フロー，結果

得られた解

目的関数：28

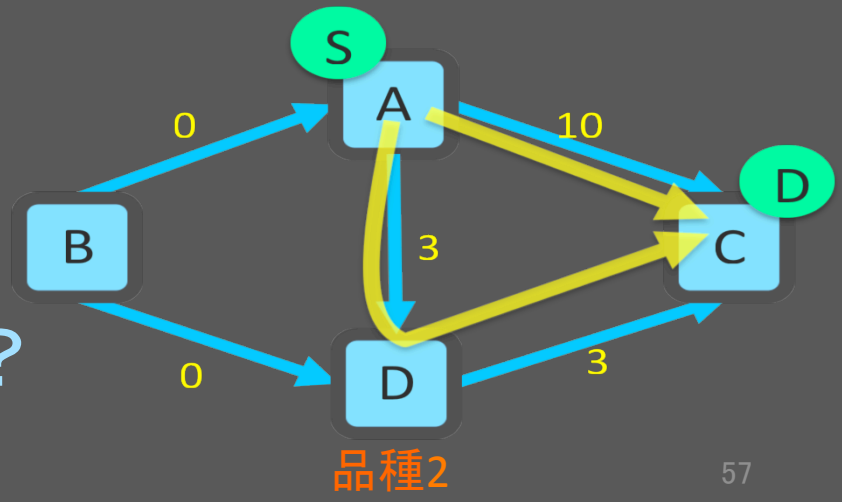


各品種の流量

品種1：15

品種2：13

これは”最適”な流し方？



最適な”割り当て”の問題

最適にネットワークを使いきれる条件

　１品種の場合　品種1:28

　２品種の場合　品種1:15, 品種2:13

要求

　Aさん「品種１のパスに２０流したい」

　Bさん「品種２のパスに１３流したい」

どうする？？

最適な”割り当て”の問題

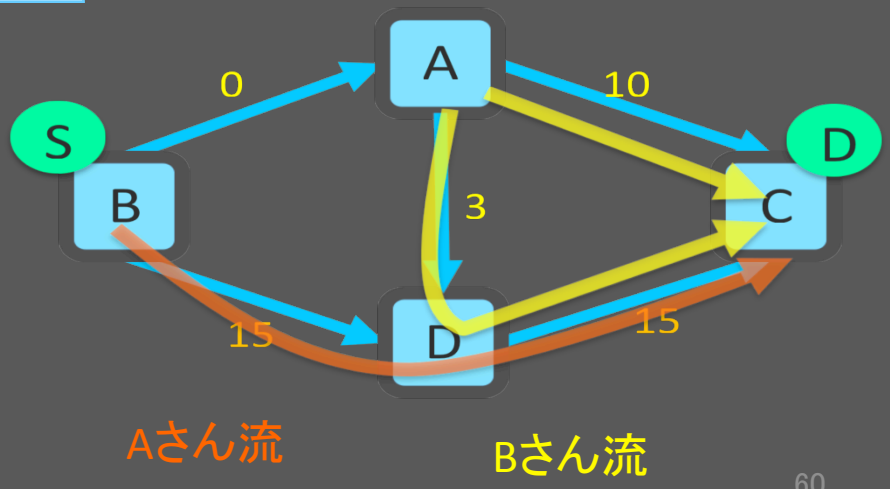
回答1

「Aさん品種1そんなに流せないので15で」

ネットワーク全体の利用率を再優先した考え方

	要求	割り当て	割り当て/要求
Aさん	20	15	75%
Bさん	13	13	100%

	最大流	割り当て	利用率
グラフ全体	28	28	100%
品種1	15	15	100%
品種2	13	13	100%
品種1単独	28	13	65%



最適な”割り当て”の問題

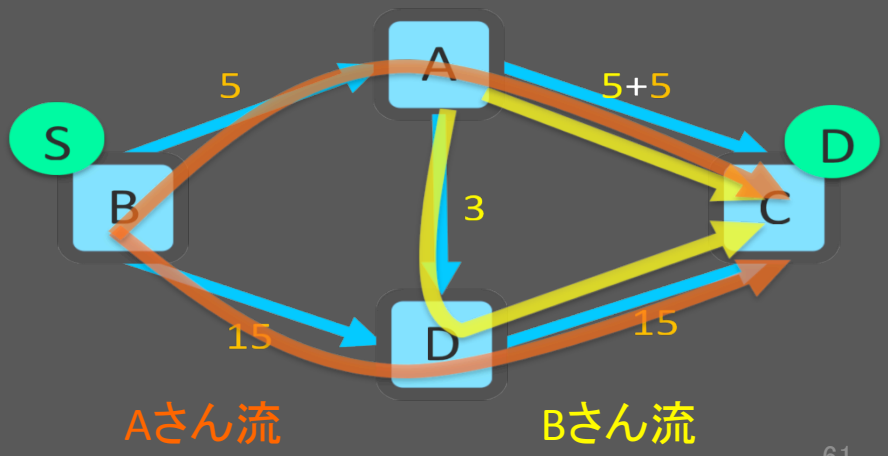
回答2

「Aさんの方が偉いからAさん優先にしよう」

要求に完全に優先度を付け順に割当てる作戦

	要求	割り当て	割り当て/要求
Aさん	20	20	100%
Bさん	13	8	61%

	最大流	割り当て	利用率
グラフ全体	28	28	100%
品種1	15	20	133%
品種2	13	8	61%
品種1単独	20	20	100%



Fairness

フェアな割り当てとはなにか？

1. 最低でもXの帯域を全員が確保できる
2. 均一な割り当て/要求レートを目指す
3. ネットワークの利用率をとにかく上げる

公平性の一つの考え方として
Max-min Fairnessを紹介

Fairness Policy

A-B-Cと直列に繋がれたトポロジ

e1は2 ， e2は4の容量を持つ

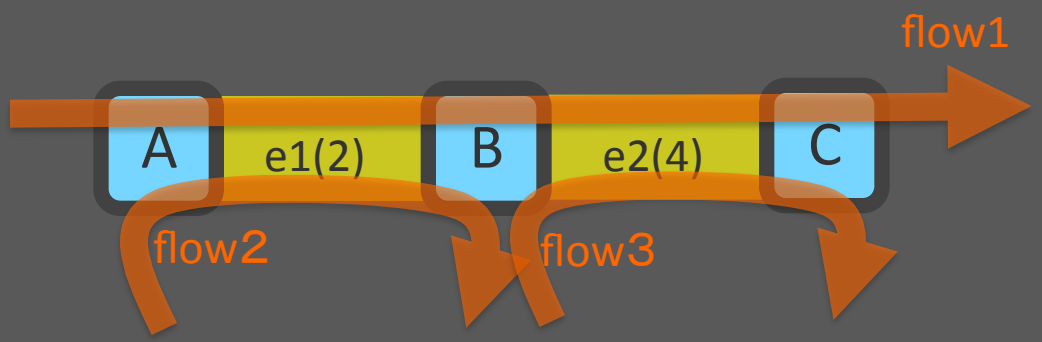
Flow群:A->B,B->C,A->C

flow	割り当て
1	0
2	2
3	4
Sum	6

最大フローを目的とする場合の解

flow1:0,flow2:2,flow3:4 の割り当てが最適

ネットワーク全体で6の流量を確保->最大フロー



Fairness Policy

A-B-Cと直列に繋がれたトポロジ

e1は2 ， e2は4の容量を持つ

Flow群:A->B,B->C,A->C

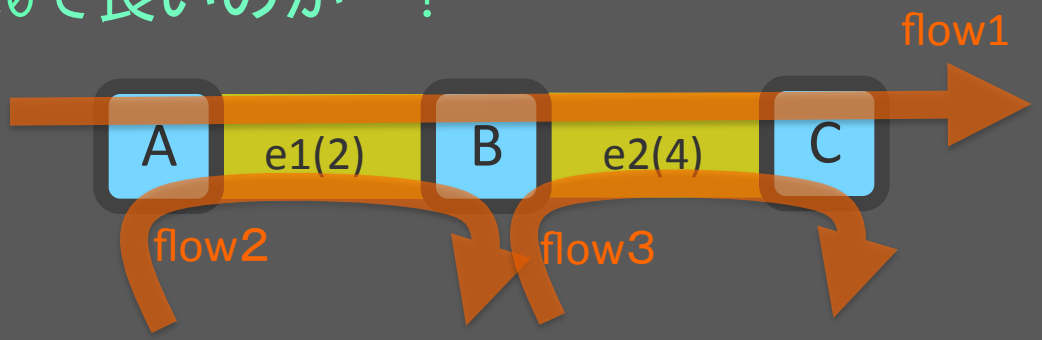
flow	割り当て
1	0
2	2
3	4
Sum	6

最大フローを目的とする場合の解

flow1:0,flow2:2,flow3:4 の割り当てが最適

ネットワーク全体で6の流量を確保->最大フロー

flow1は0で良いのか…？



Max-min Fairness Policy

Maximize the minimum (to allocate)

割り当て可能な帯域が最も小さいものを優先する

アルゴリズム

1. 全てのflowを同一ペースで増加させた場合に、最初に飽和するリンク(ボトルネックリンクと呼ぶ)を特定する
2. 全てのflowに対して、ボトルネックリンクが発生する流量分割り当てを行い、NWからボトルネックリンクを削除
3. 増加することのできるflowが存在する場合は1に戻る

Max-min Fairness Policy

1.全てのflowを同一ペースで増加させた場合に、最初に飽和するリンク(ボトルネックリンクと呼ぶ)を特定する

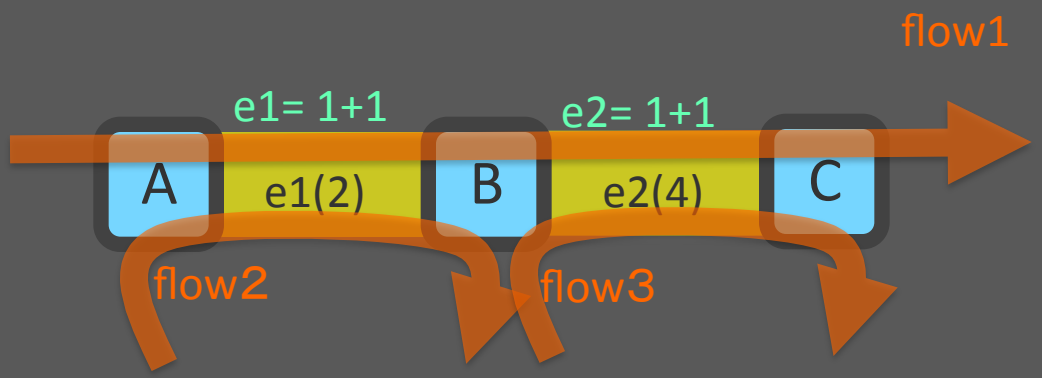
flow	割り当て
1	0
2	0
3	0
Sum	0

下のトポロジの場合

flow1=flow2=flow3= 1 の時にe1が飽和

e1がボトルネックリンク

ボトルネックリンクが最初に発生する流量は1



Max-min Fairness Policy

2.全てのflowに対して、ボトルネックリンクが発生する流量分割り当てを行い、NWからボトルネックリンクを削除

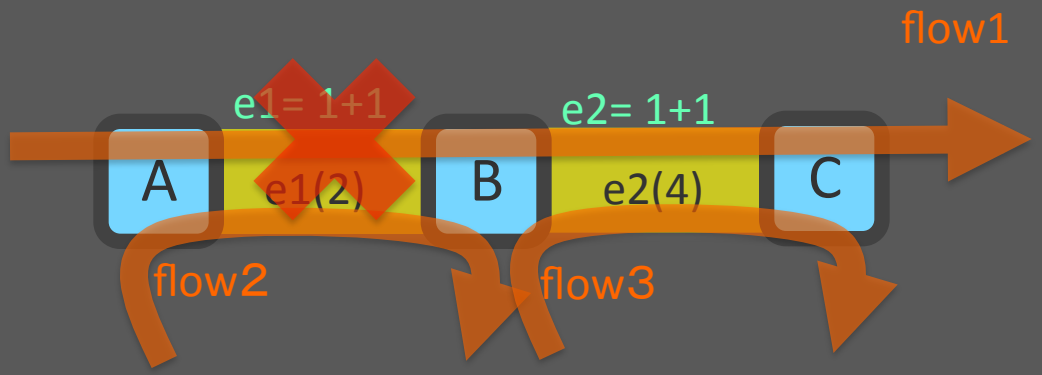
flow	割り当て
1	1
2	1
3	1
Sum	3

下のトポロジの場合

flow1=flow2=flow3=に1を割り当て

ボトルネックリンクe1を削除

まだflow3は増やせるので1に戻る



Max-min Fairness Policy

1.全てのflowを同一ペースで増加させた場合に、最初に飽和するリンク(ボトルネックリンクと呼ぶ)を特定する

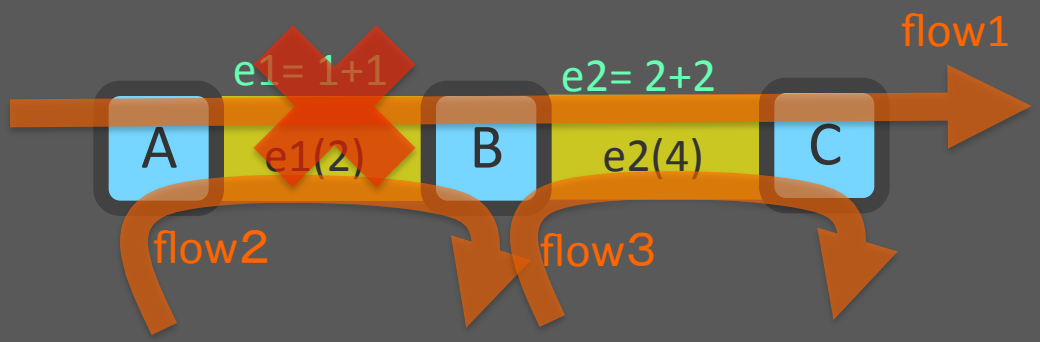
flow	割り当て
1	1
2	1
3	1
Sum	3

下のトポロジの場合

flow3=2 の時にe2が飽和

e2がボトルネックリンク

ボトルネックリンクが最初に発生する流量は2



Max-min Fairness Policy

2.全てのflowに対して、ボトルネックリンクが発生する流量分割り当てを行い、NWからボトルネックリンクを削除

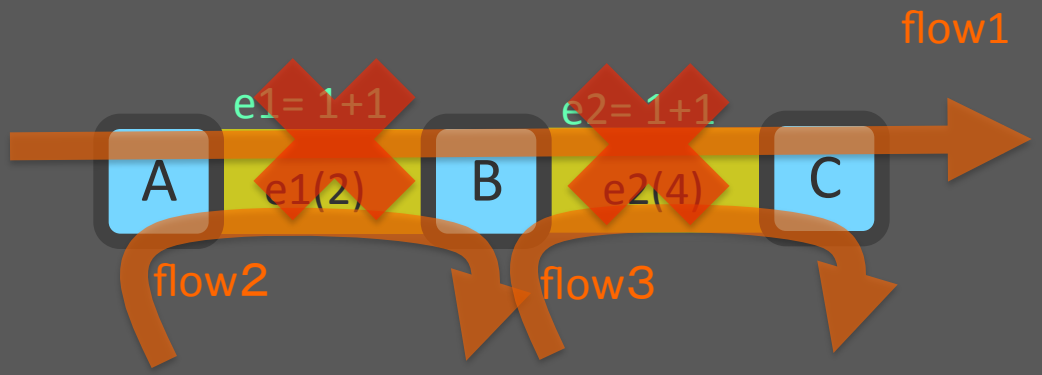
flow	割り当て
1	1
2	1
3	3
Sum	5

下のトポロジの場合

flow3=に2を割り当て

ボトルネックリンクe2を削除

増加可能なflowが存在しないので、終了



Fairness Policy

Max-min fairness

– 合計帯域は減少したが割り当ては公平に

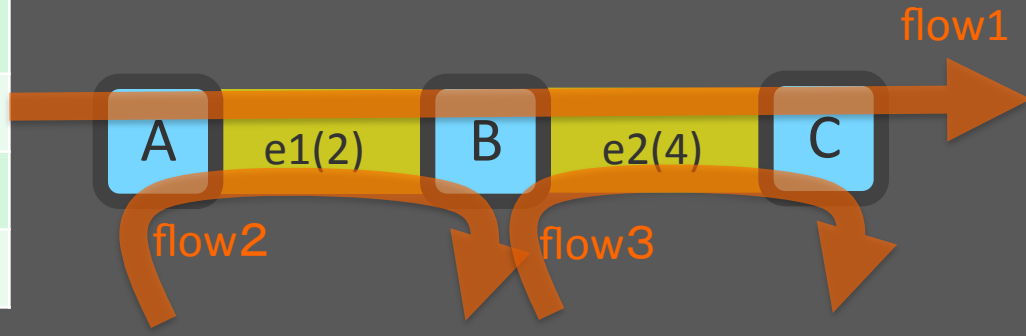
Fairness Policyはユースケース次第

最大流

flow	割り当て
1	0
2	2
3	4
Sum	6

Max-min

flow	割り当て
1	1
2	1
3	3
Sum	5



まとめ

- ネットワークフロー問題
  - 容量付きエッジ，流れを扱う組み合わせ最適化
  - 線形計画法に落としこむことができる
- 最大フロー問題
  - Ford-Fulkerson法、線形計画法
- Fairness
  - 複数のフロー間でどう容量を分け合うか？
  - Max-min fairness

SDN時代の今こそ”動かせる”！