Depth from Focus
PA#1

Supplementary Materials

Supplementary Materials for PA#1

- Lens and Focusing
- Depth from Focus
- Graph cut optimization
- Learning based DFF method

Depth and Focus

Lens' aperture

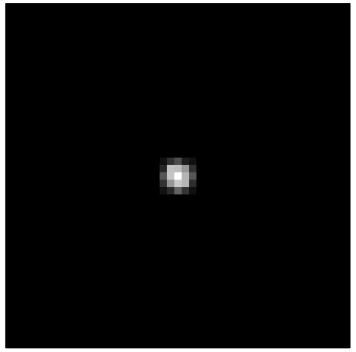
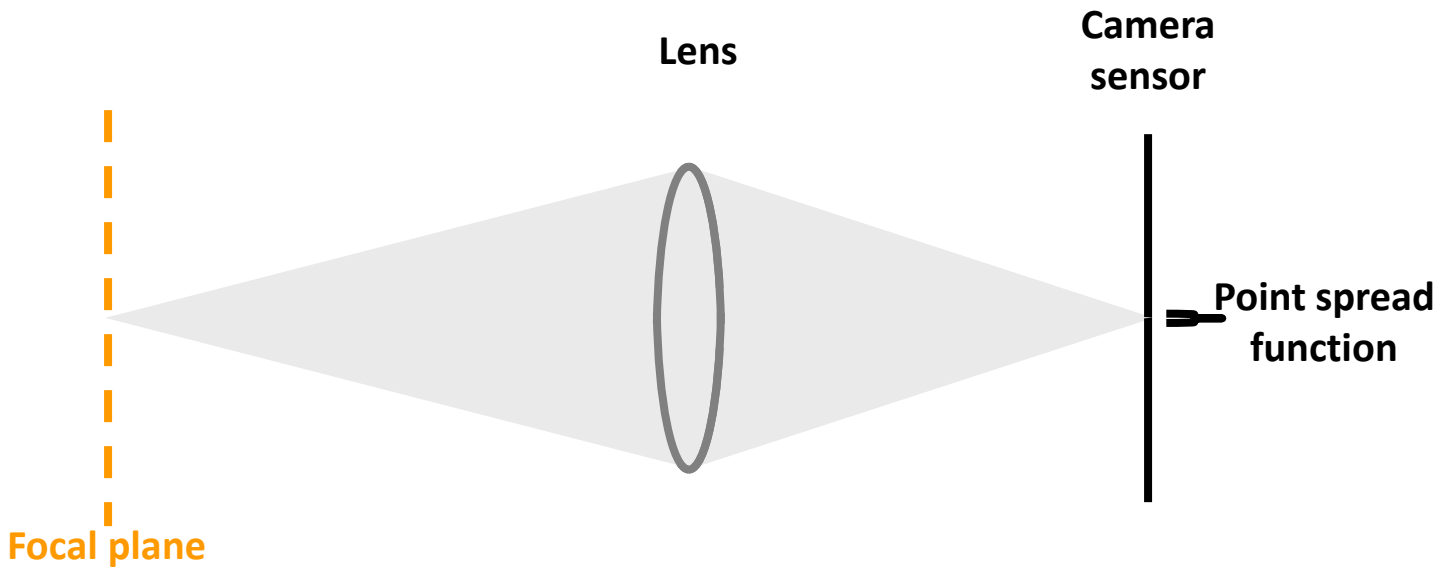


Image of a point light source



Slide by Levin

Depth and Focus

Lens' aperture

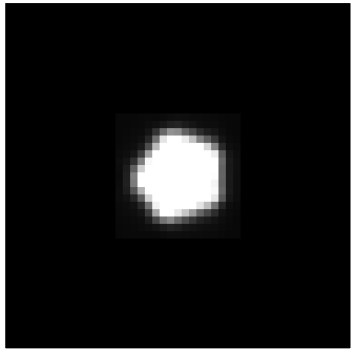
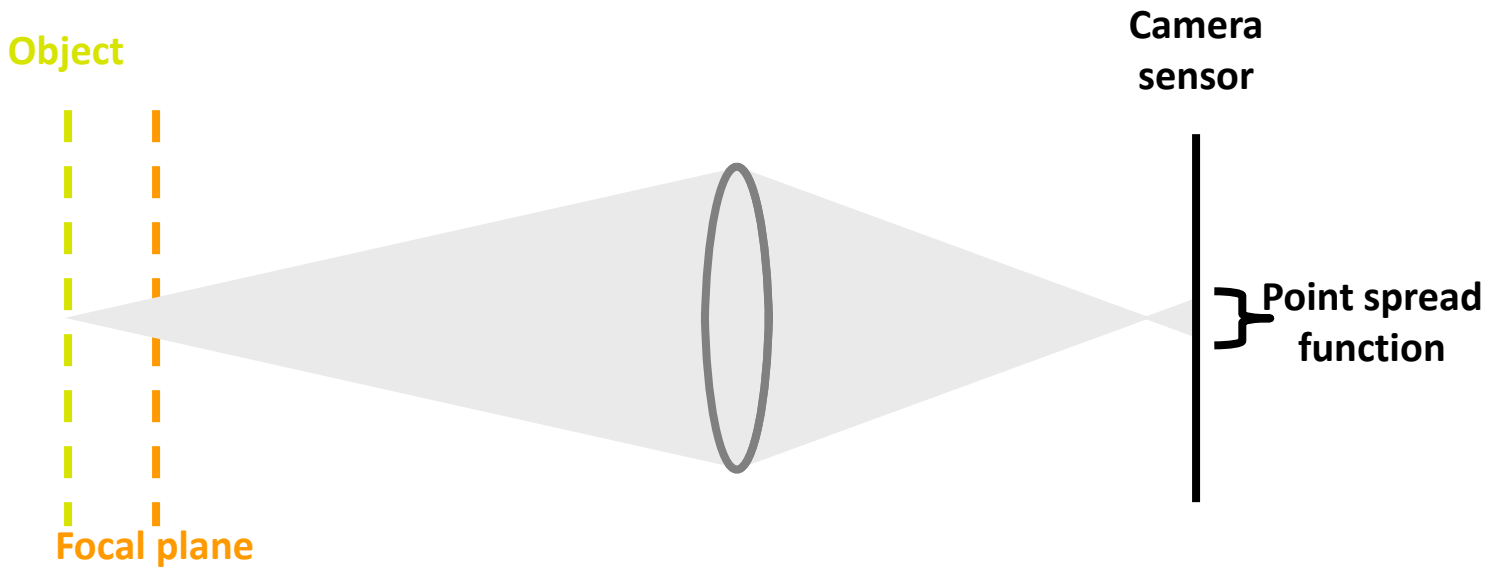
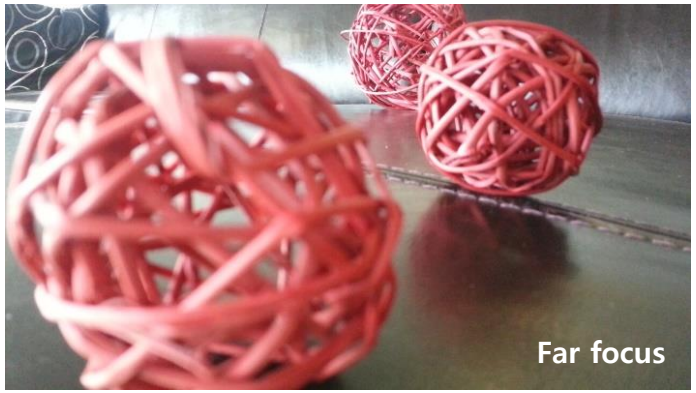


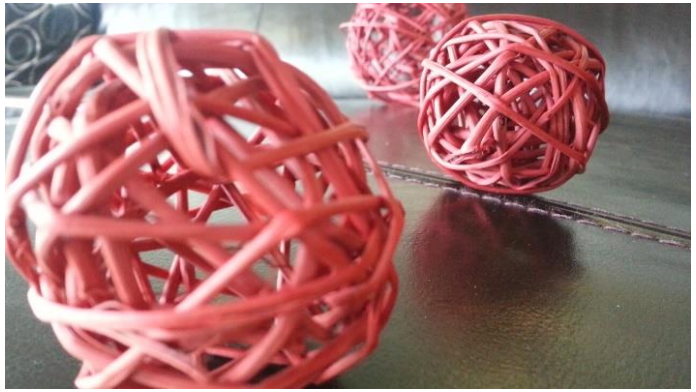
Image of a point light source

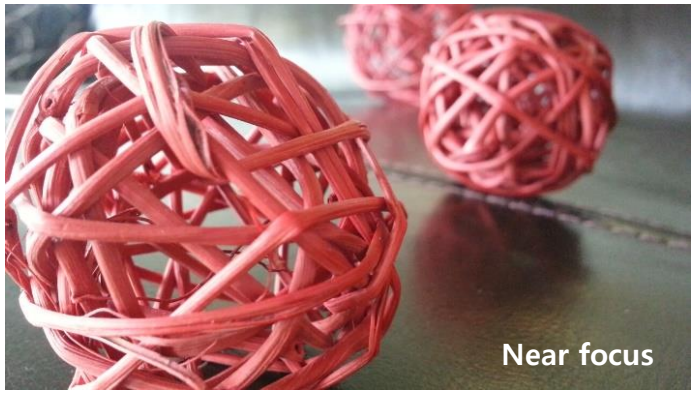


Slide by Levin

Depth and Focus







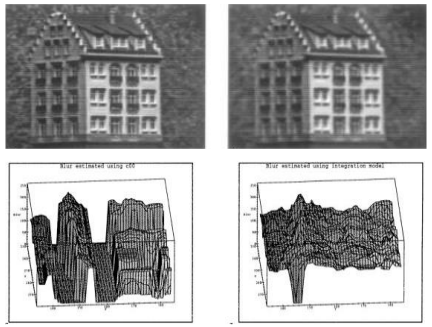
Focusing (Defocus)

- The amount of defocus is one of the priceless information that can be obtained from a single image.
- Estimated focus can be applied to many computer vision tasks.

Slides by Jinsun Park

Focusing: Applications

- Defocus information can be a cornerstone for various applications.



Depth from defocus [1]

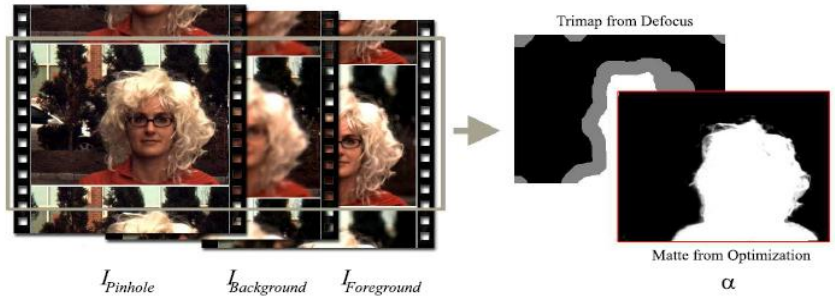
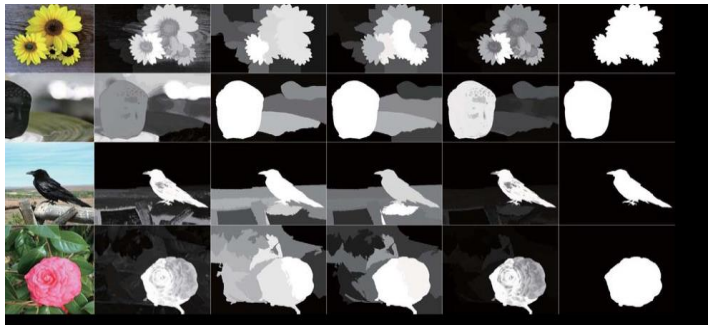


Image matting [2]



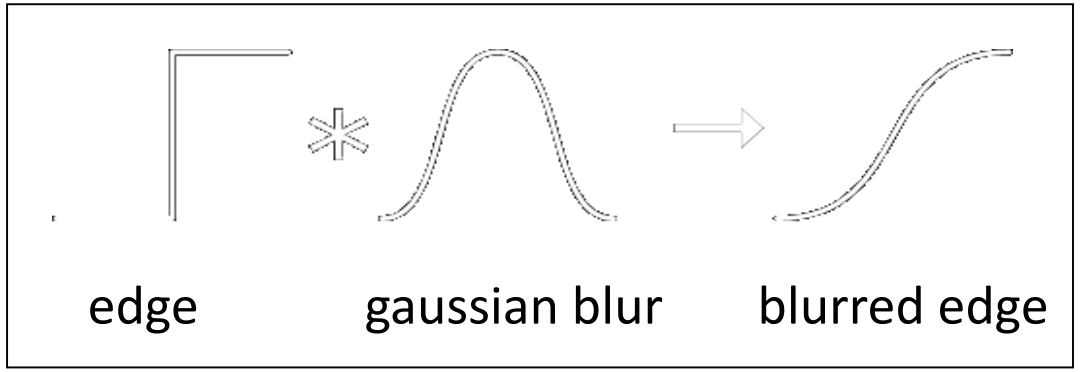
Salient region detection [3]

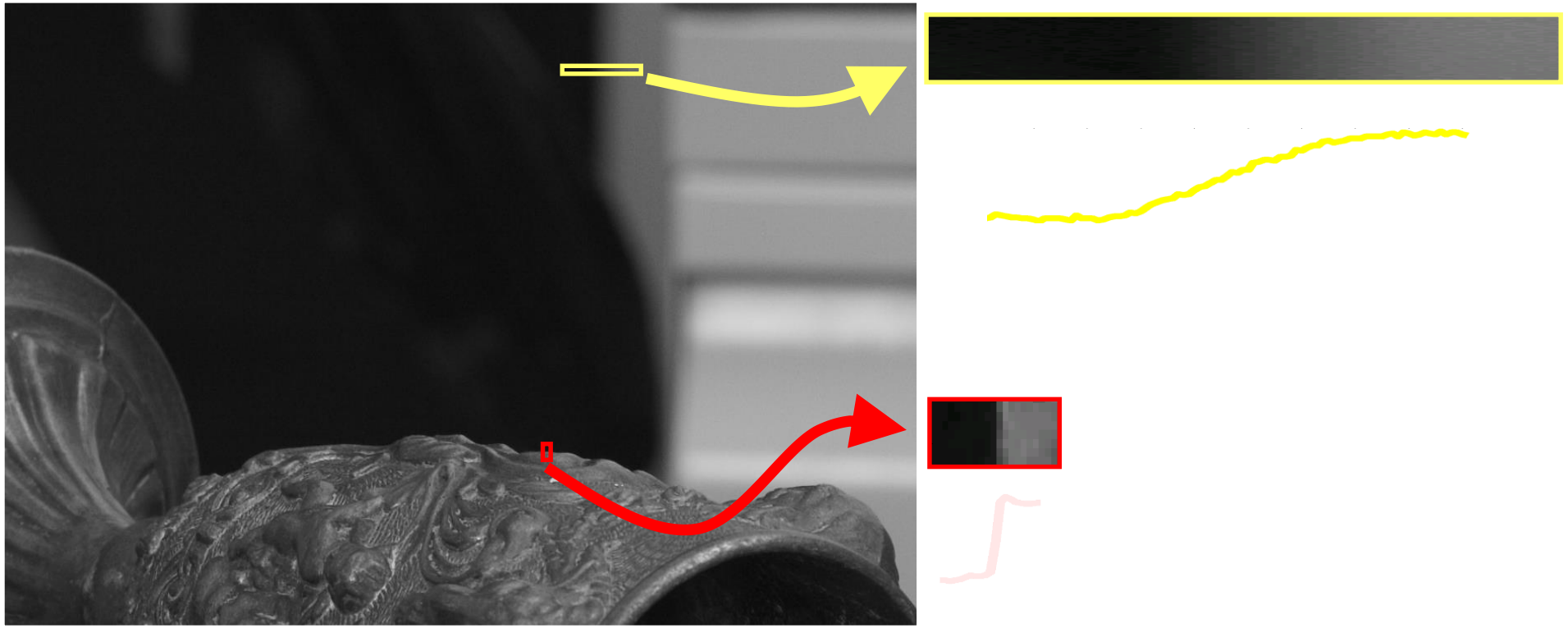
Digital refocusing [4]

[1] Ziou, Djemel, and Francois Deschênes. "Depth from defocus estimation in spatial domain." CVIU 2001.
[2] McGuire, Morgan, et al. "Defocus video matting." TOG 2005.
[3] Jiang, Peng, et al. "Salient region detection by ufo: Uniqueness, focusness and objectness." ICCV 2013.
[4] Cao, Yang, Shuai Fang, and Zengfu Wang. "Digital multi-focusing from a single photograph taken with an uncalibrated conventional camera." TIP 2013

Focus Estimation at Edges

- An edge
  - a step function in intensity
- The blur of this edge
  - a Gaussian blurring kernel

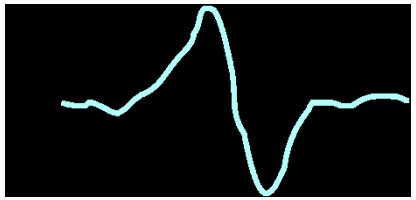




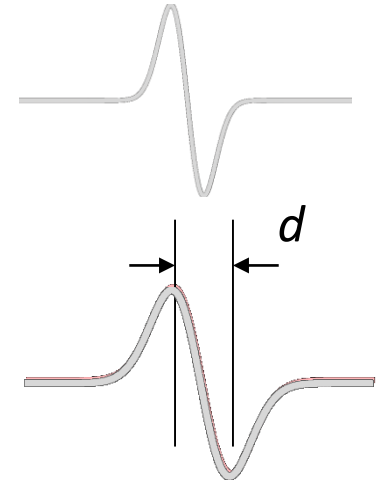
Focus Estimation at Edges

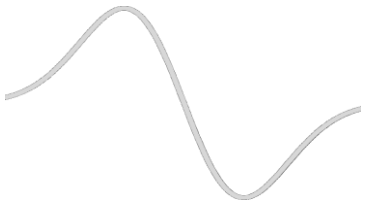
- Fit response models of various sizes

edge



2nd derivative





response model

less blurry

more blurry

Slide by S. Bae

Depth and Focus Measure

$$Laplacian = \frac{\partial^2 g(x,y)}{\partial^2 x} + \frac{\partial^2 g(x,y)}{\partial^2 y}$$

g is an image frame

(x, y) is a pixel index

$$Modified\ Laplacian(ML) = \left(\frac{\partial^2 g(x,y)}{\partial^2 x}\right)^2 + \left(\frac{\partial^2 g(x,y)}{\partial^2 y}\right)^2$$

$$Sum\ of\ ML = \sum_{p(x,y)=U(x_0,y_0)} \left(\frac{\partial^2 g(x,y)}{\partial^2 x}\right)^2 + \left(\frac{\partial^2 g(x,y)}{\partial^2 y}\right)^2$$

$p(x, y)$ is a pixel in the neighborhood $U(x_0, y_0)$ of pixel (x_0, y_0)

$$Tenenbaum\ Focus\ Measure = \sum_{p(x,y)=U(x_0,y_0)} \left(g_x(x,y)^2 + g_y(x,y)^2\right)^2$$

A novel algorithm for estimation of depth map using image focus for 3D shape recovery in the presence of noise, A. Malik and T.-S Choi, Pattern Recognition 2008

Depth and Focus Measure

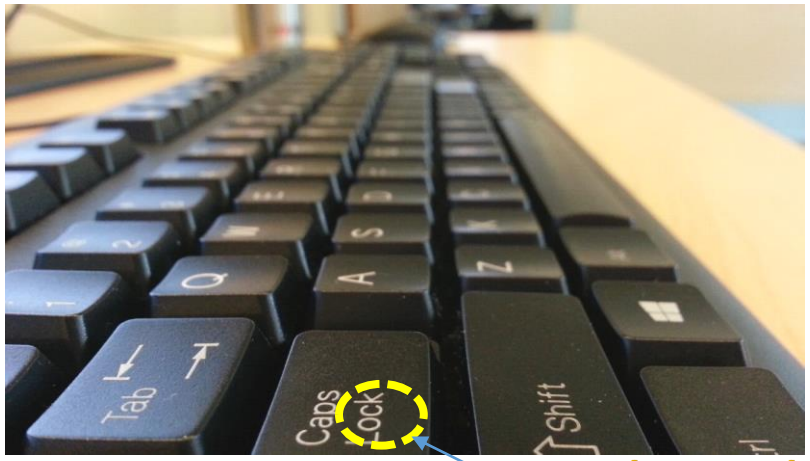
- Finding the maximum focus measure = Estimating the depth

Far

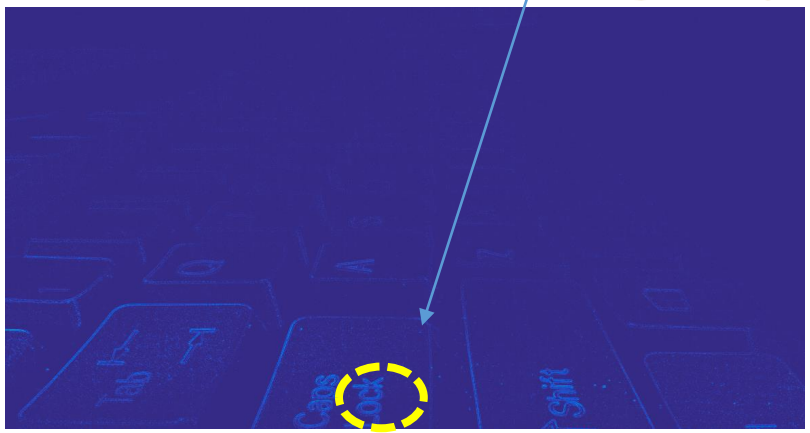
Near

***N** images*
in a focal stack

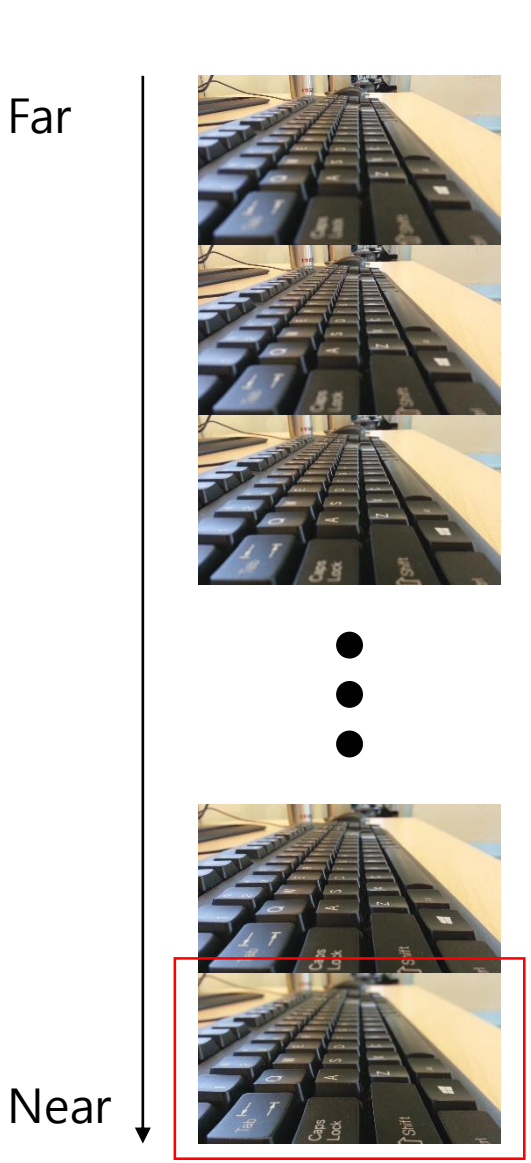
*Depth resolution : **N***

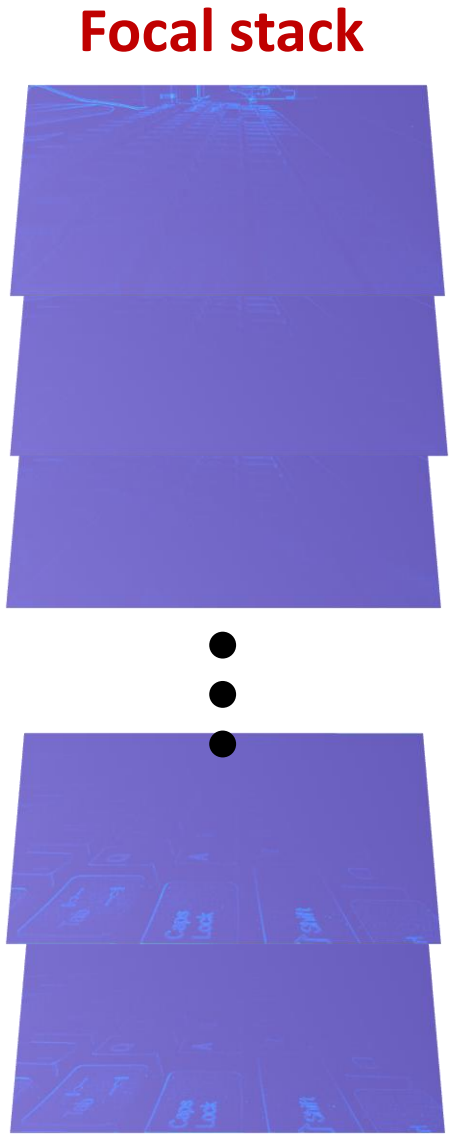


Sharp edges =
High response!!

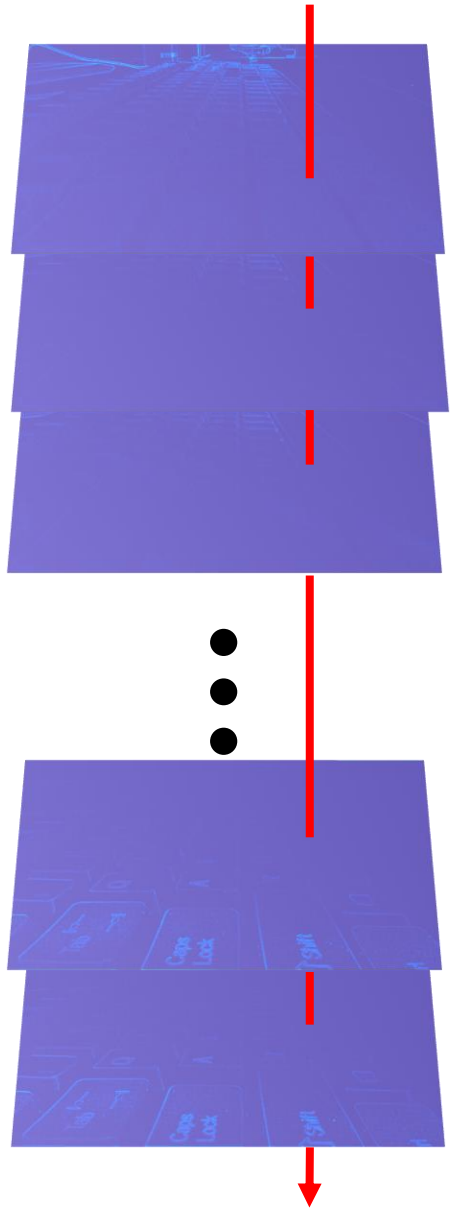


Focal Stack to Focus Measure Volume



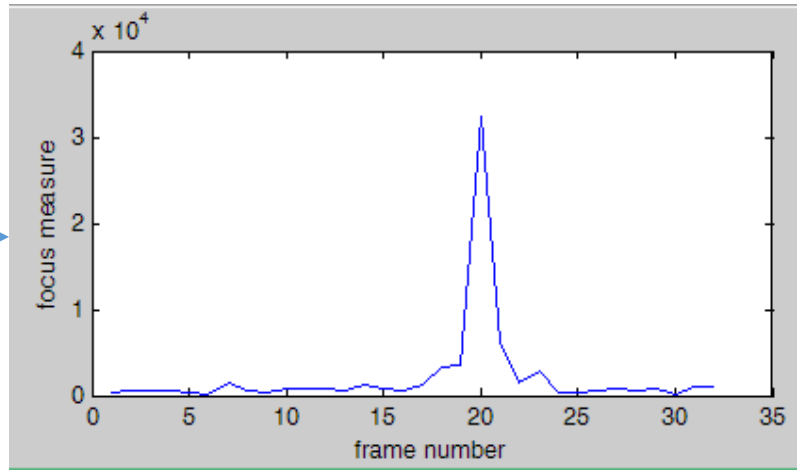


Focus Measure Volume to Index Map



***N**-th*

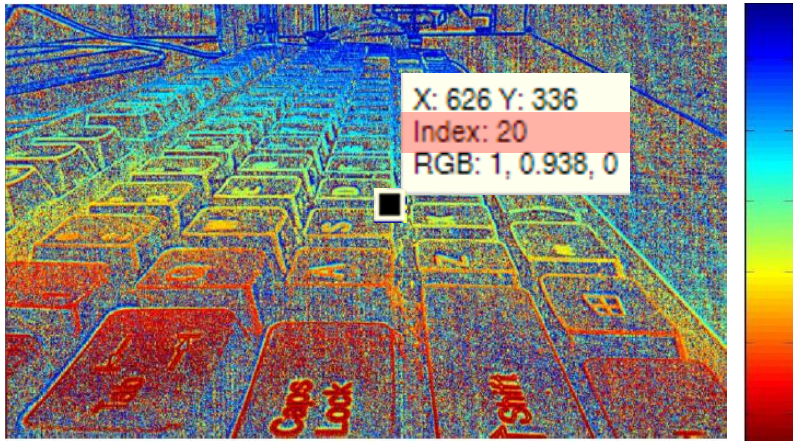
***20**-th*



→ Vote for the frame index with highest response.

***2**-nd*

***1**-st*



N

20

1

Initial Index Map

N
= # of images in a volume
= Depth resolution

Confidence of Focus Measure

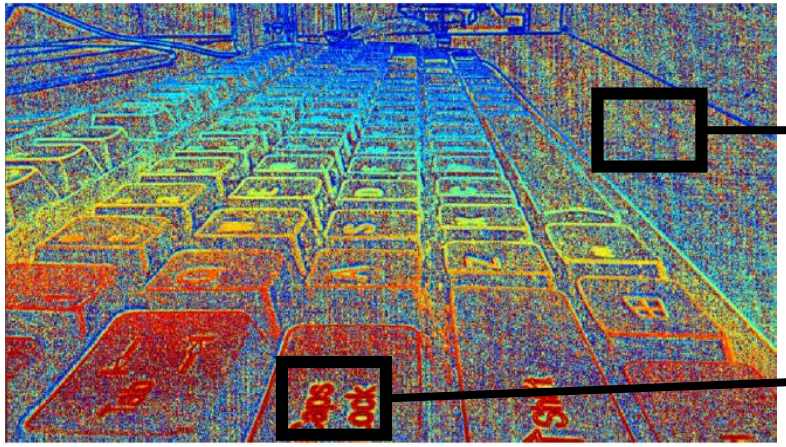
- However, focus estimation is a very challenging task.

→ *these pixels are **sparse.***

- There exist ambiguities in homogeneous regions.
- Homogeneous regions have almost no difference in appearance when they are blurred or not.

Confidence of Focus Measure

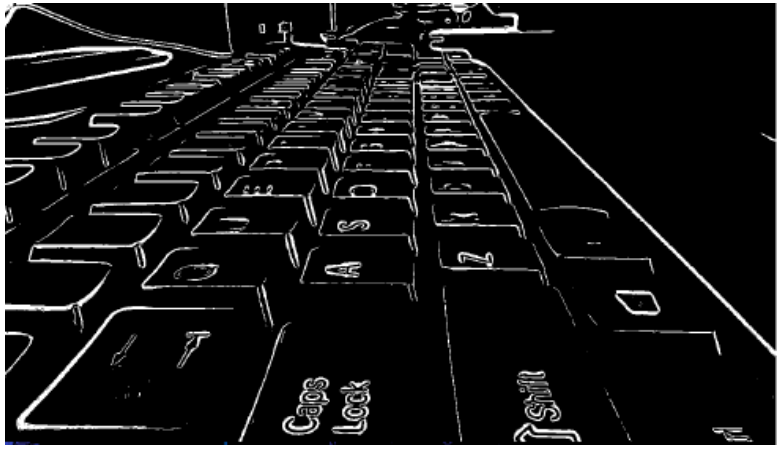
- **Initial index depth map**



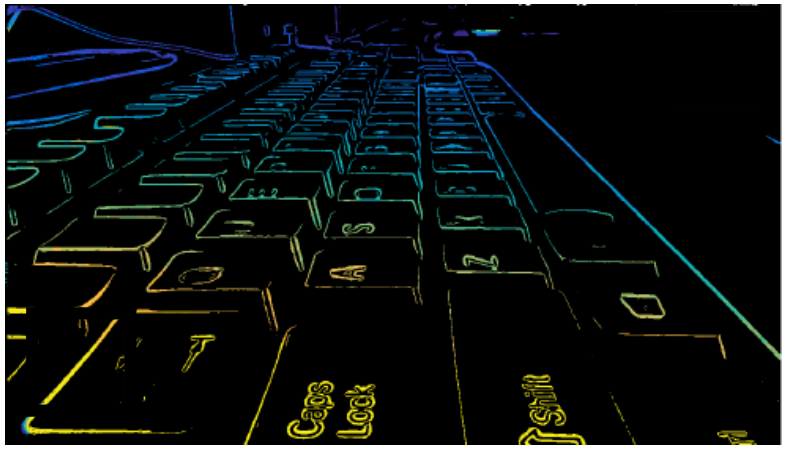
Ambiguous:
@ homogeneous regions,
Noisy, misclassified.

Confident:
@ edge regions.

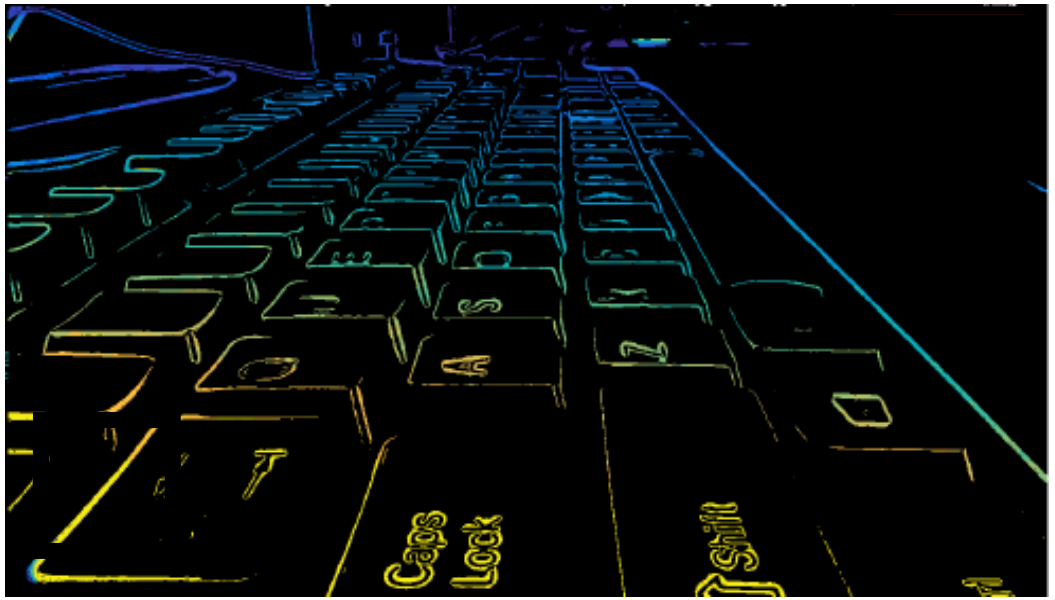
- **Collect edge detections from all frames**



- **Sparse depth map – confident (valid) pixels**

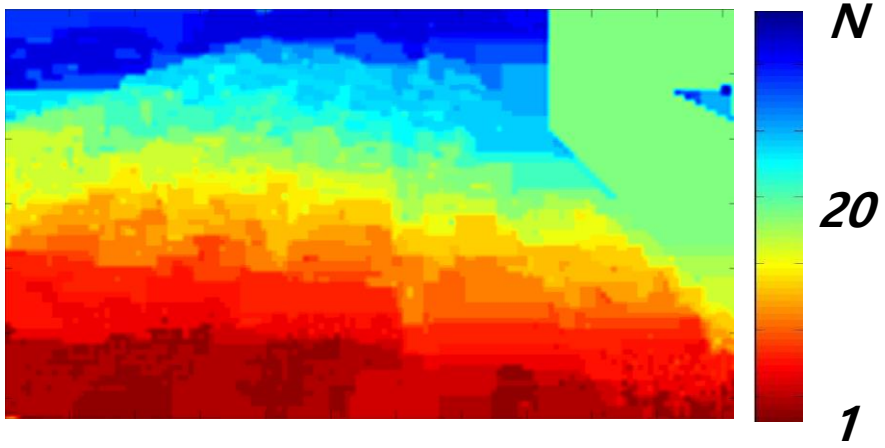


Sparse-to-Dense Depth Map



Sparse depth map – confident (valid) pixels

How to obtain dense depth map?



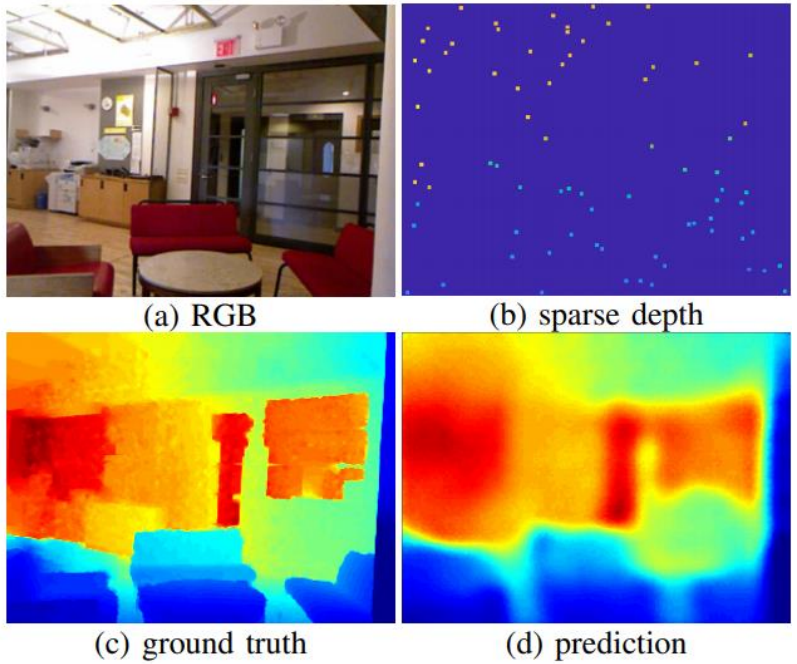
Sparse-to-Dense Propagation in Computer Vision

- Depth map propagation (from sparse sensor data points)

Sparse-to-Dense: Depth Prediction from Sparse Depth Samples and a Single Image

Fangchang Ma[1] and Sertac Karaman[1]

***Abstract*— We consider the problem of dense depth prediction from a sparse set of depth measurements and a single RGB image. Since depth estimation from monocular images alone is inherently ambiguous and unreliable, to attain a higher level of robustness and accuracy, we introduce additional sparse depth samples, which are either acquired with a low-resolution depth sensor or computed via visual Simultaneous Localization and Mapping (SLAM) algorithms. We propose the use of a single deep regression network to learn directly from the RGB-D raw data, and explore the impact of number of depth samples on prediction accuracy. Our experiments show that, compared to using only RGB images, the addition of 100 spatially random depth samples reduces the prediction root-mean-square error by 50% on the NYU-Depth-v2 indoor dataset. It also boosts the percentage of reliable prediction from 59% to 92% on the KITTI dataset. We demonstrate two applications of the proposed algorithm: a plug-in module in SLAM to convert sparse maps to dense maps, and super-resolution for LiDARs. Software[2] and video demonstration[3] are publicly available.**



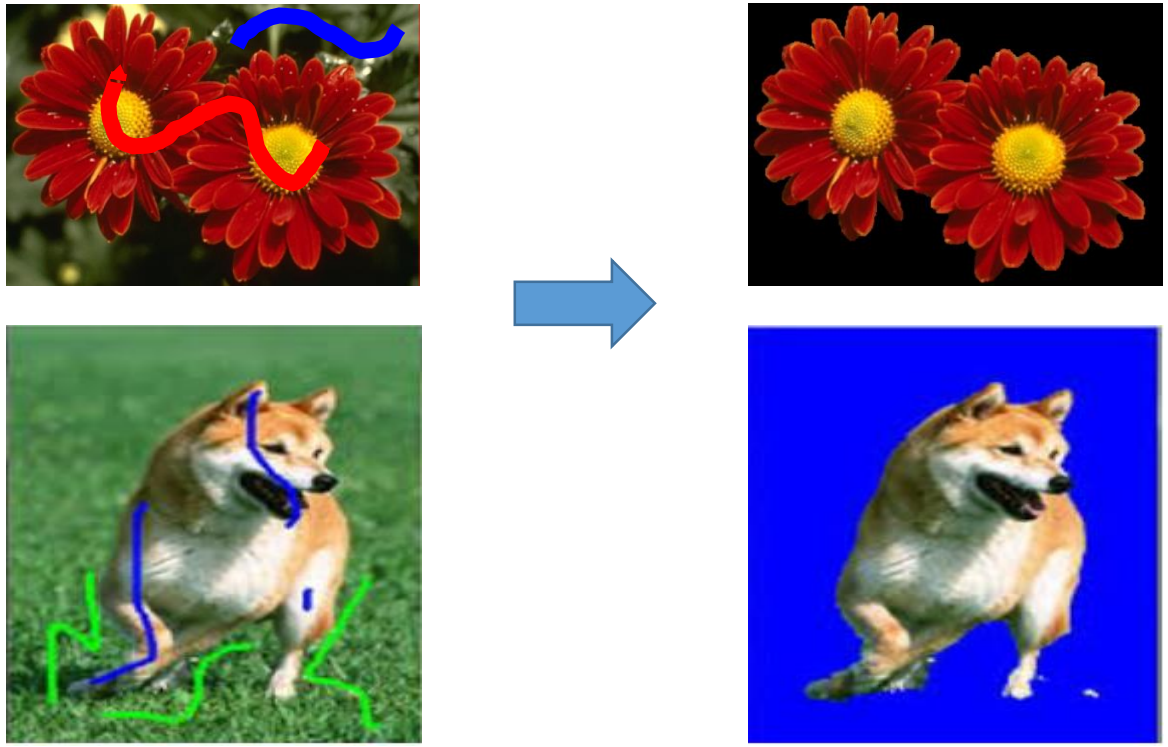
(a) RGB (b) sparse depth (c) ground truth (d) prediction

Sparse-to-Dense Propagation in Computer Vision

- Image labeling problems (segmentation)

Sparse confident pixels (user scribble)

Dense segmentation result



[Tutorial slides from Ľubor Ladický]

Example: Image Segmentation

How to formulate this?

Background : $x_i = 0$
Foreground : $x_i = 1$

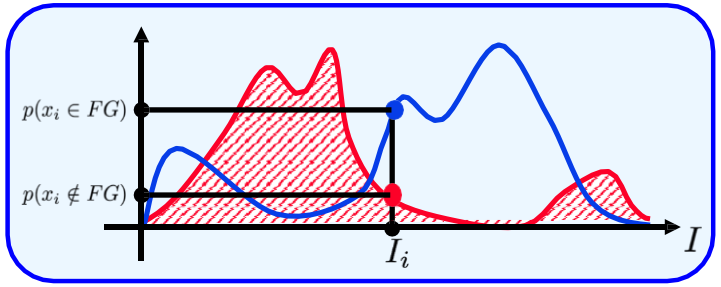
Many algorithms are based on Graph Cuts.

Image Segmentation by Graph Cuts

Should be estimated automatically
or semi-automatically

F/B based on data and color model

e.g. if color≈red or color≈yellow
p is foreground
else
p is background



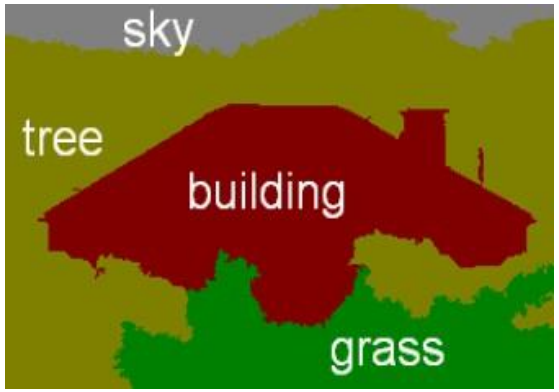
Estimated using FG / BG color models

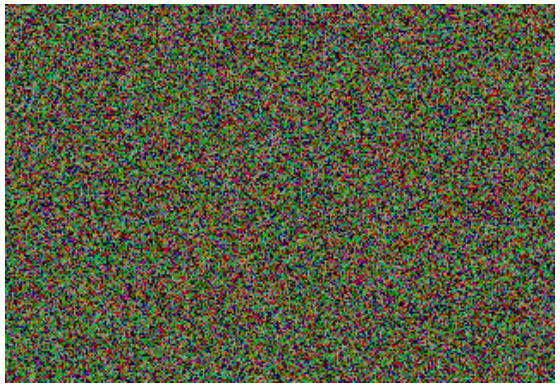
Data term: based on the data (e.g. color) and model (e.g. color model)

Usually not perfect (noise, etc), so need **smoothness**

[tutorial slides from Ľubor Ladický]

Image Segmentation by Graph Cuts





F/B based on neighbor labels

e.g. if neighbor of p is FG
p is FG
else
p is BG

Should get the same label

Smoothness term: based on the neighbor labels
i.e. "neighbouring pixels tend to take the same label"

Also high-level smoothness term or constraints

[tutorial slides from Ľubor Ladický]

Image Segmentation by Graph Cuts

$$\mathbf{x}^* = \arg\min_{x \in \mathbf{L}} E(\mathbf{x}) \qquad E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \psi_i(x_i) + \sum_{i \in \mathcal{V}, j \in \mathcal{N}_i} \psi_{ij}(x_i, x_j)$$

Unknowns

Data term **Smoothness term**

$x_i = 0 \Longrightarrow i \in Background$

$x_i = 1 \Longrightarrow i \in Foreground$

Other popular notation:

$l_i = 0 \Longrightarrow i \in Background$

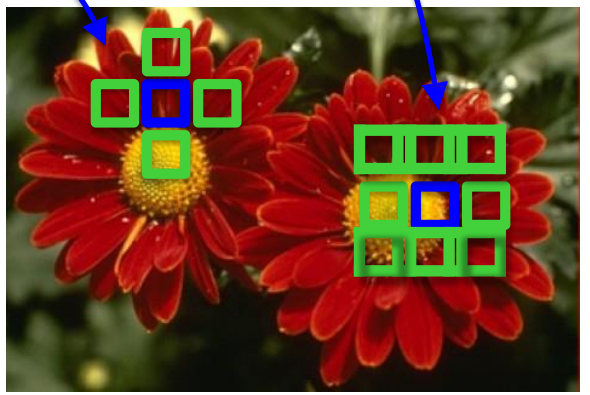
$l_i = 1 \Longrightarrow i \in Foreground$

$\mathcal{V}$: the set of all the pixels

$\mathcal{N}_i$: the set of all the pixels in the neighborhood of pixel *i*
i.e. "the neighbors of *i*"

8-neighbor connectivity

4-neighbor connectivity



[tutorial slides from Ľubor Ladický]

Image Segmentation by Graph Cuts

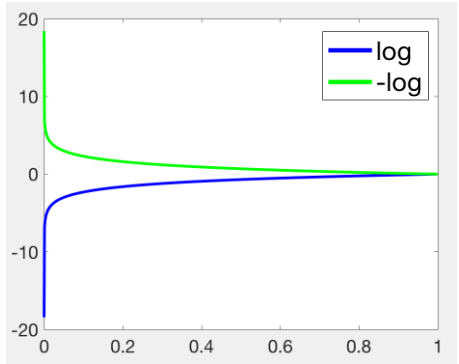
$$\mathbf{x}^* = \arg\min_{x \in \mathbf{L}} E(\mathbf{x})$$

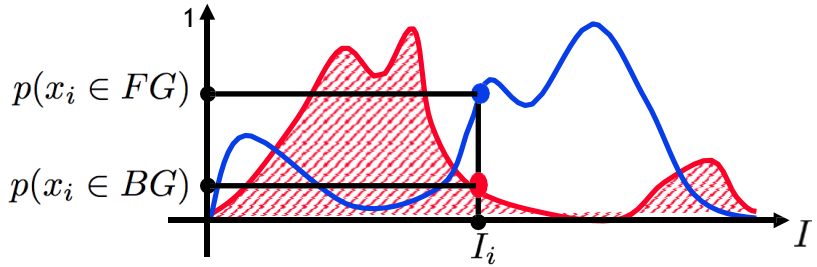
$$E(\mathbf{x}) = \underbrace{\sum_{i \in \mathcal{V}} \psi_i(x_i)}_{\text{Data term}} + \underbrace{\sum_{i \in \mathcal{V}, j \in \mathcal{N}_i} \psi_{ij}(x_i, x_j)}_{\text{Smoothness term}}$$

Unknowns

$x_i = 0 \Longrightarrow i \in Background$

$x_i = 1 \Longrightarrow i \in Foreground$





Estimated using FG / BG color models

$$\psi_i(0) = \psi_i(BG) = -\log(p(x_i \in BG))$$
$$\psi_i(1) = \psi_i(FG) = -\log(p(x_i \in FG))$$

Why -log?

- we are adding weights. We multiply probabilities, so add logs
- maximize probabilities, so minimize -log

[tutorial slides from Ľubor Ladický]

Image Segmentation by Graph Cuts

$$\mathbf{x}^* = \arg\min_{x \in \mathbf{L}} E(\mathbf{x})$$

$$E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \psi_i(x_i) + \sum_{i \in \mathcal{V}, j \in \mathcal{N}_i} \psi_{ij}(x_i, x_j)$$

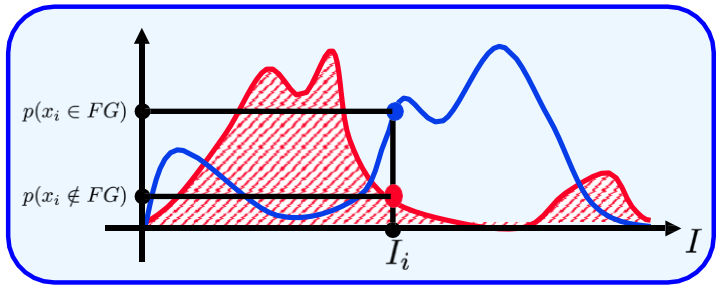
Unknowns

$x_i = 0 \Longrightarrow i \in Background$

$x_i = 1 \Longrightarrow i \in Foreground$

Data term

$$\psi_i(0) = -\log(p(x_i \notin FG))$$
$$\psi_i(1) = -\log(p(x_i \in FG))$$



Estimated using FG / BG color models

Smoothness term

"Potts model"

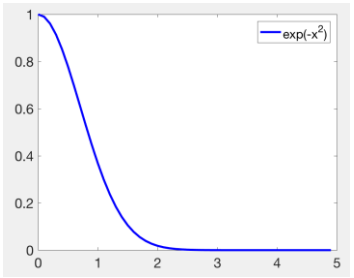
$$\psi_{ij}(x_i, x_j) = \delta(x_i \neq x_j)$$

if $x_i = x_j \Rightarrow 0$
else 1

$$\psi_{ij}(x_i, x_j) = K_{ij}\delta(x_i \neq x_j)$$

where $K_{ij} = \lambda_1 + \lambda_2 \exp(-\beta(I_i - I_j)^2)$

Intensity dependent smoothness



[tutorial slides from Ľubor Ladický]

Image Segmentation by Graph Cuts

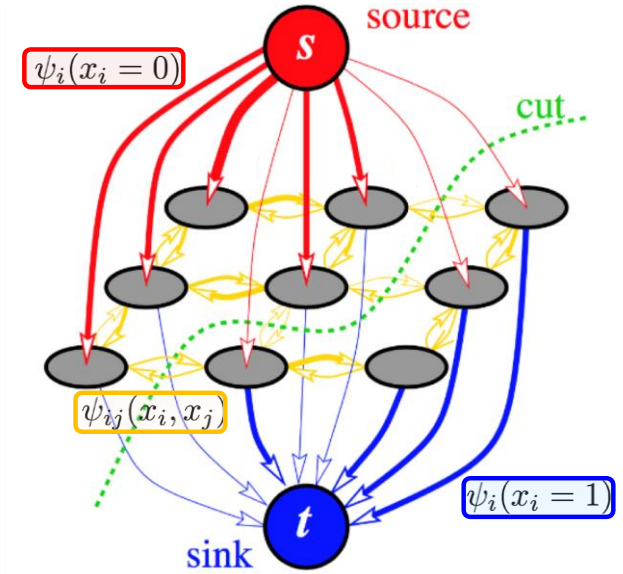
$$E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \psi_i(x_i) + \sum_{i \in \mathcal{V}, j \in \mathcal{N}_i} \psi_{ij}(x_i, x_j)$$

Data term **Smoothness term**

$$\mathbf{x}^* = \arg\min_{x \in \mathbf{L}} E(\mathbf{x})$$

How to solve this optimization problem?

- Transform into max-flow / min-cut problem
- Solve it using max-flow / min-cut algorithm



[tutorial slides from Ľubor Ladický]

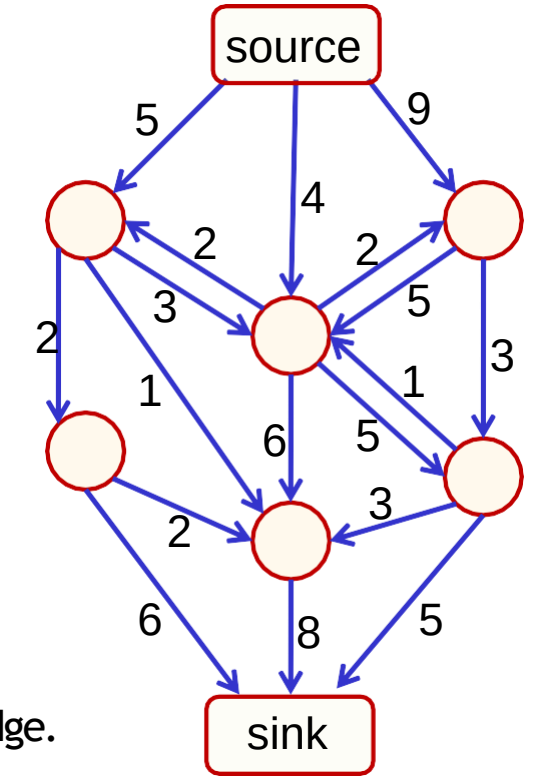
Max Flow Problem

Task :

Maximize the flow from the source to the sink such that

1) The flow is conserved for each node (in=out)

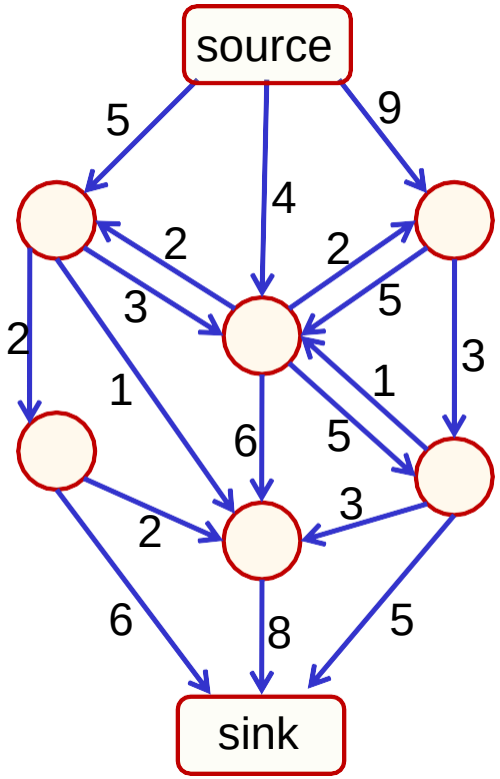
2) The flow for each pipe does not exceed the capacity



"capacity" of an edge: the maximum amount of flow that can pass through an edge.

[tutorial slides from Ľubor Ladický]

Max Flow Problem



flow from the source

flow from node *i* to node *j*

capacity from node *i* to node *j*

$$\max \sum_{i \in V} f_{si}$$

set of edges

$$s.t. \quad 0 \leq f_{ij} \leq c_{ij}, \qquad \forall (i,j) \in E$$

$$\sum_{j \in N(i)} f_{ji} - f_{ij} = 0, \qquad \forall i \in V \setminus \{s,t\}$$

conservation of flow

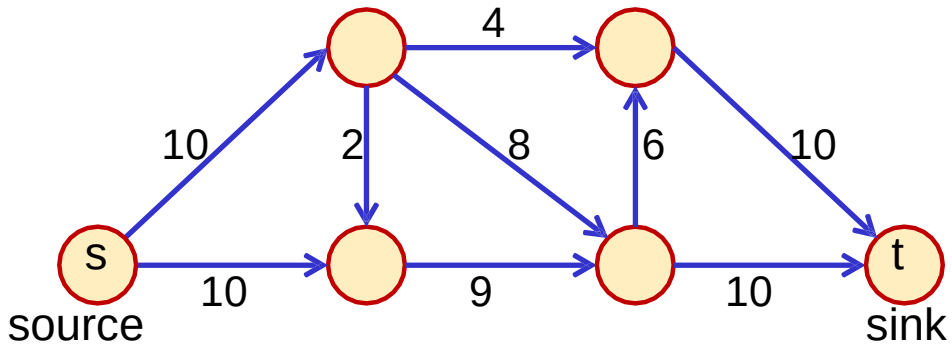
i.e. flow in=flow out

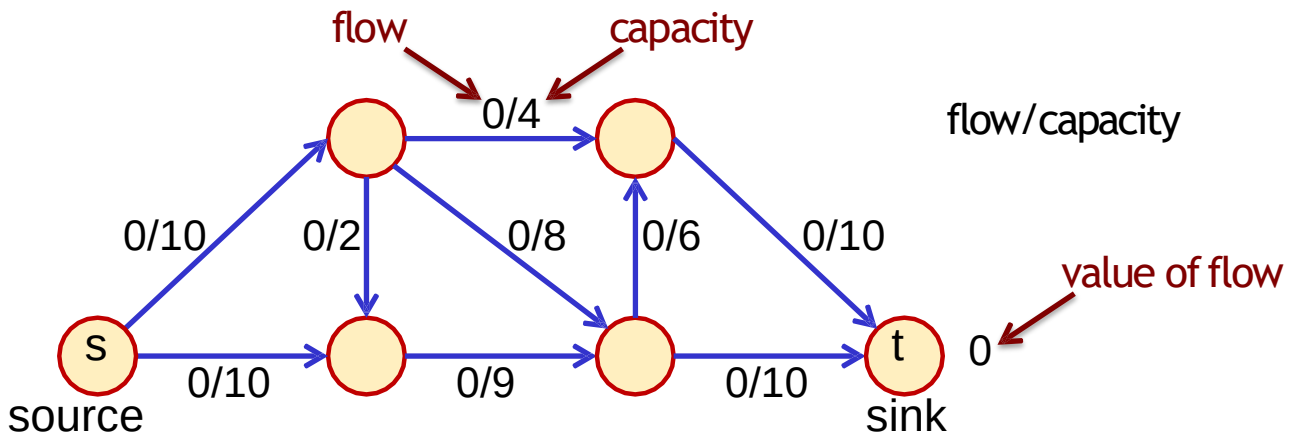
set of nodes

[tutorial slides from Ľubor Ladický]

Max Flow Problem

Graph with capacity

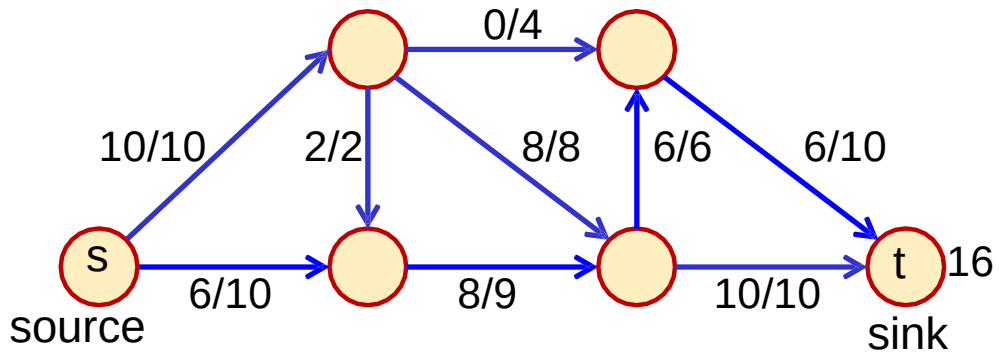




[tutorial slides from Ľubor Ladický]

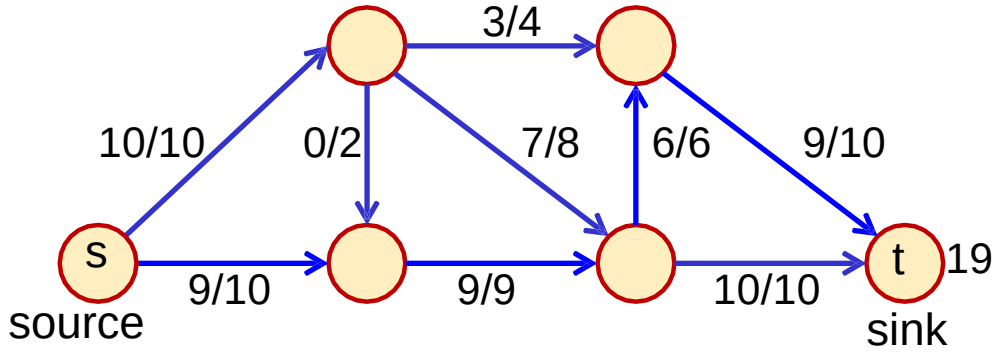
Max Flow Problem

flow/capacity



flow in=flow out
flow conversation
flow<=capacity

Total flow=16



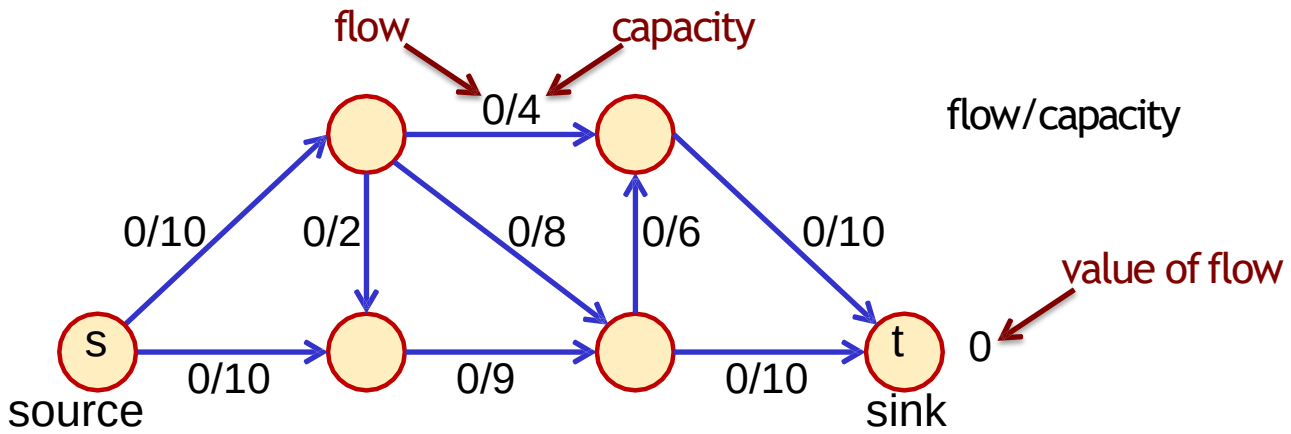
Total flow=19

[tutorial slides from Ľubor Ladický]

Max Flow Problem - Greedy algorithm

Greedy algorithm:

flow=0
Find a “valid” path from source to sink
While (path exists)
    “Augment” flow
    Find a “valid” path from source to sink
End



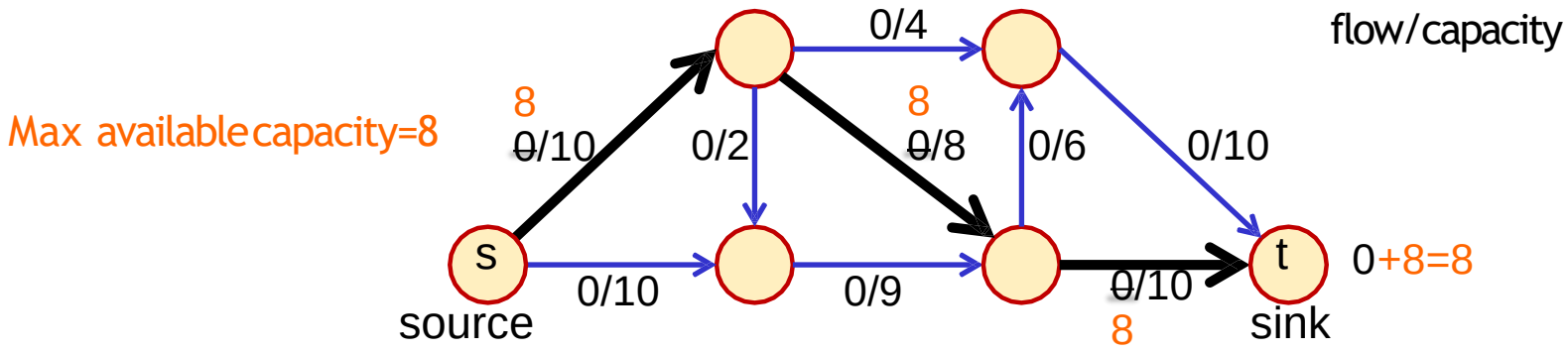
http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/07NetworkFlowI-2x2.pdf

[tutorial slides from Ľubor Ladický]

Max Flow Problem - Greedy algorithm

Greedy algorithm:

flow=0
Find a “valid” path from source to sink
While (path exists)
“Augment” flow
Find a “valid” path from source to sink
End



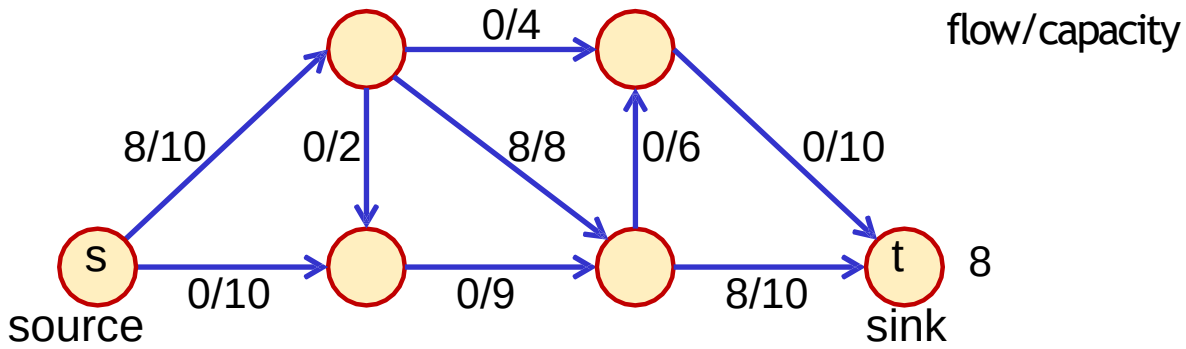
http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/07NetworkFlowI-2x2.pdf

[tutorial slides from Ľubor Ladický]

Max Flow Problem - Greedy algorithm

Greedy algorithm:

flow=0
Find a “valid” path from source to sink
While (path exists)
“Augment” flow
Find a “valid” path from source to sink
End



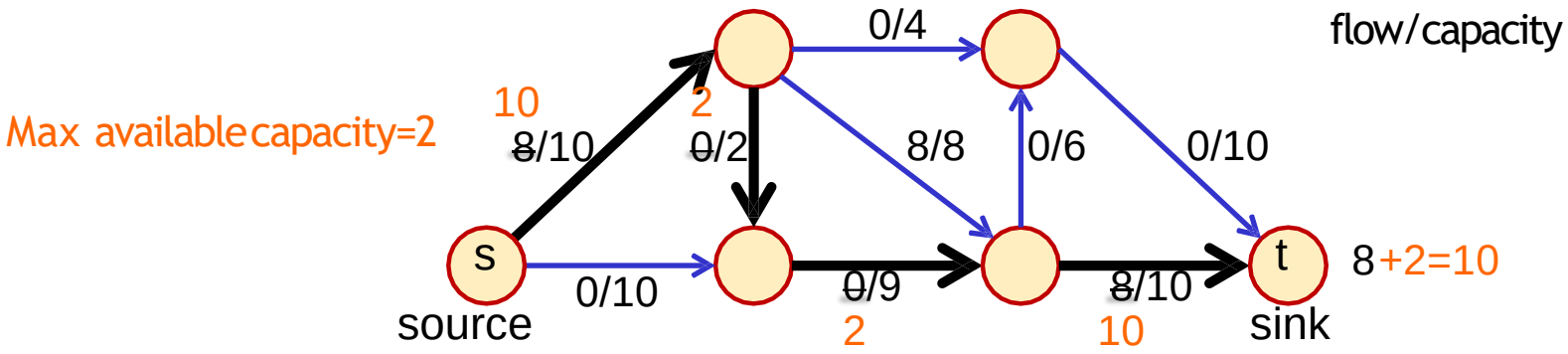
http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/07NetworkFlowI-2x2.pdf

[tutorial slides from Ľubor Ladický]

Max Flow Problem - Greedy algorithm

Greedy algorithm:

flow=0
Find a “valid” path from source to sink
While (path exists)
“Augment” flow
Find a “valid” path from source to sink
End

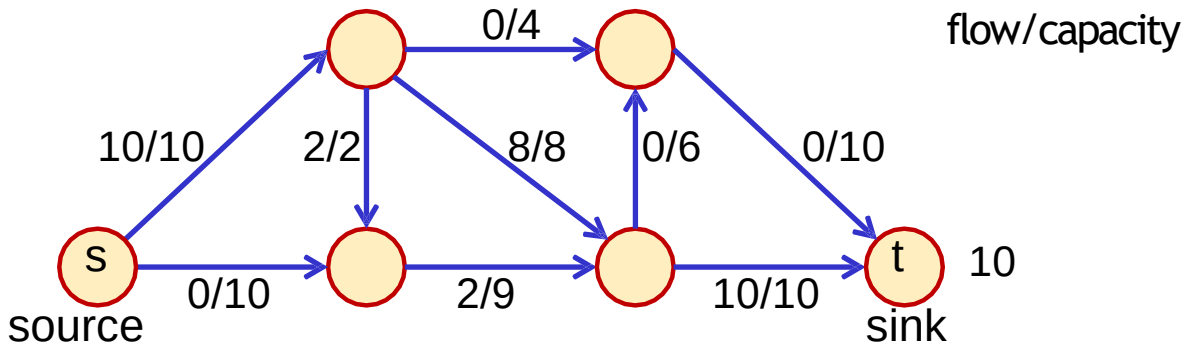


[tutorial slides from Ľubor Ladický]

Max Flow Problem - Greedy algorithm

Greedy algorithm:

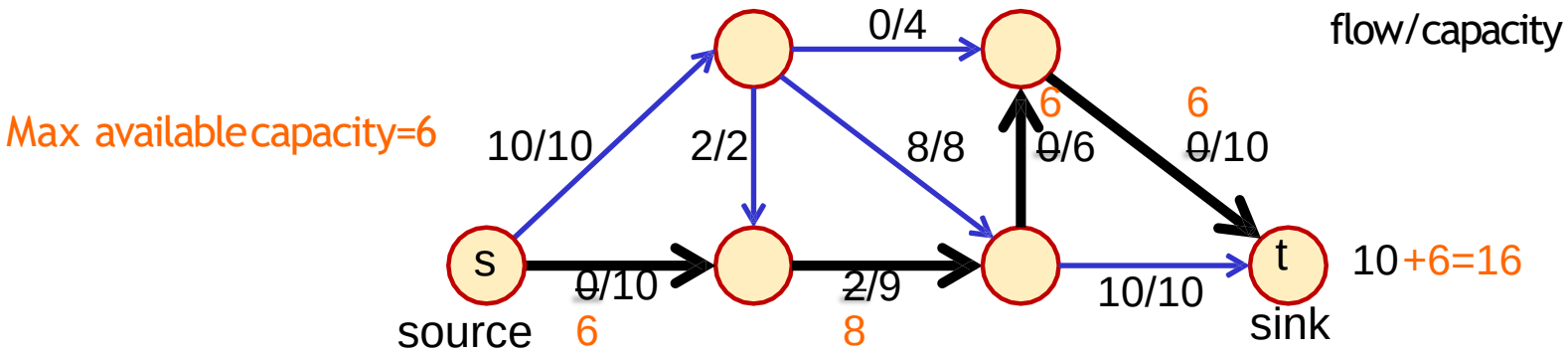
flow=0
Find a “valid” path from source to sink
While (path exists)
    “Augment” flow
    Find a “valid” path from source to sink
End



Max Flow Problem - Greedy algorithm

Greedy algorithm:

flow=0
Find a “valid” path from source to sink
While (path exists)
“Augment” flow
Find a “valid” path from source to sink
End

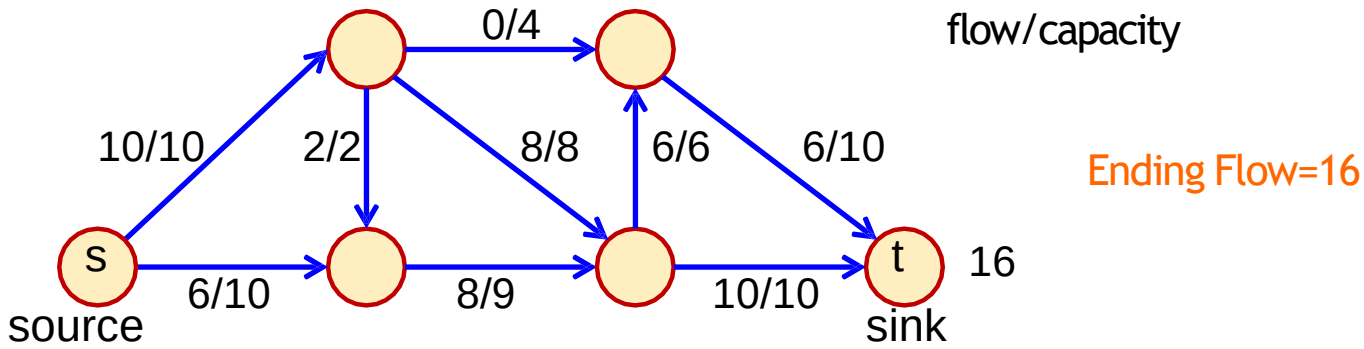


[tutorial slides from Ľubor Ladický]

Max Flow Problem - Greedy algorithm

Greedy algorithm:

flow=0
Find a “valid” path from source to sink
While (path exists)
“Augment” flow
Find a “valid” path from source to sink
End

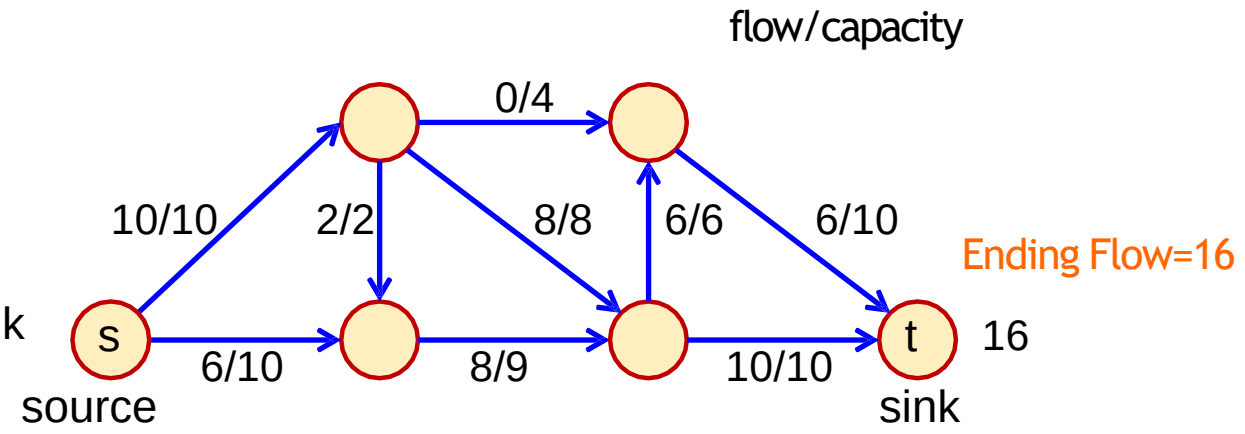


[tutorial slides from Ľubor Ladický]

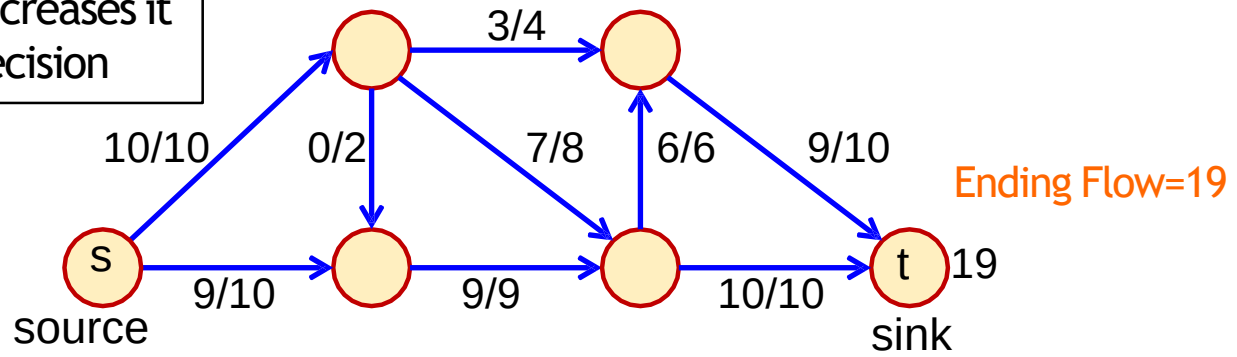
Max Flow Problem - Greedy algorithm

Greedy algorithm:

flow=0
Find a “valid” path from source to sink
While (path exists)
“Augment” flow
Find a “valid” path from source to sink
End



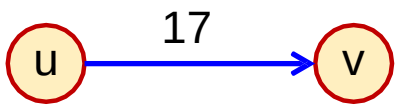
Why does it fail?
➔ Once it increases flow on an edge, it never decreases it
➔ We need some mechanisms to “undo” bad decision

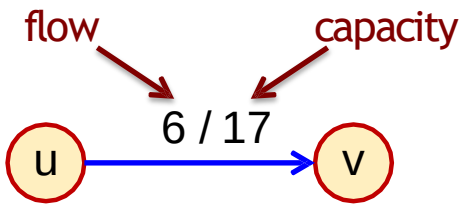


[tutorial slides from Ľubor Ladický]

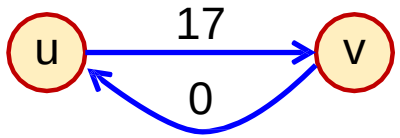
Max Flow Problem – Residual Network

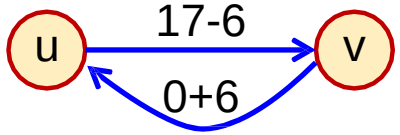
Original network *G*:

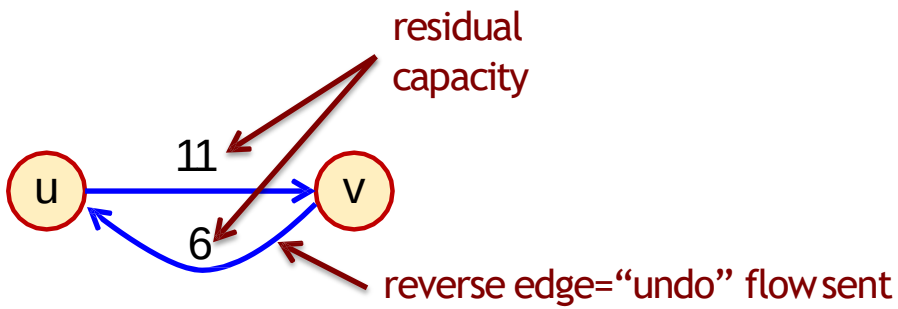




Residual network G_f:





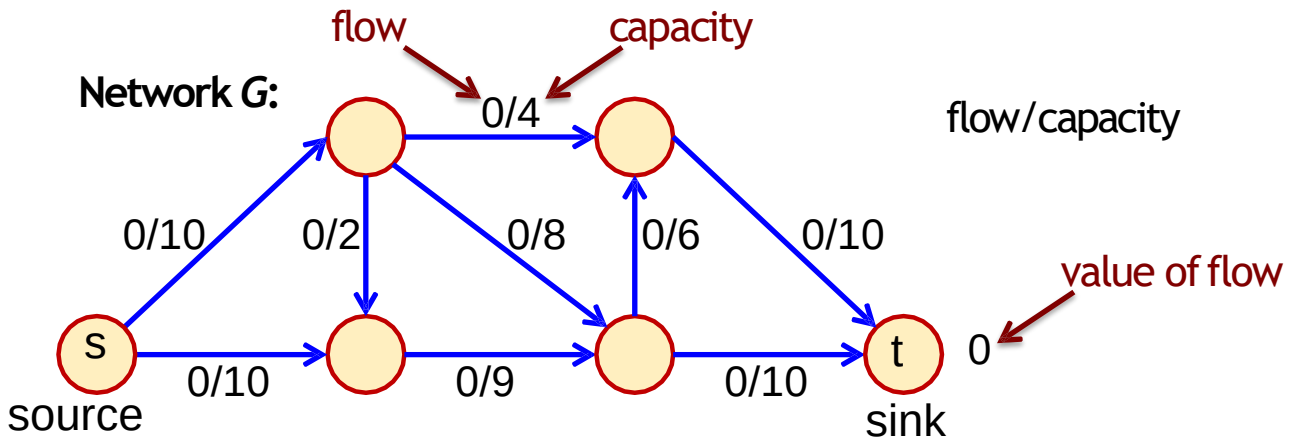


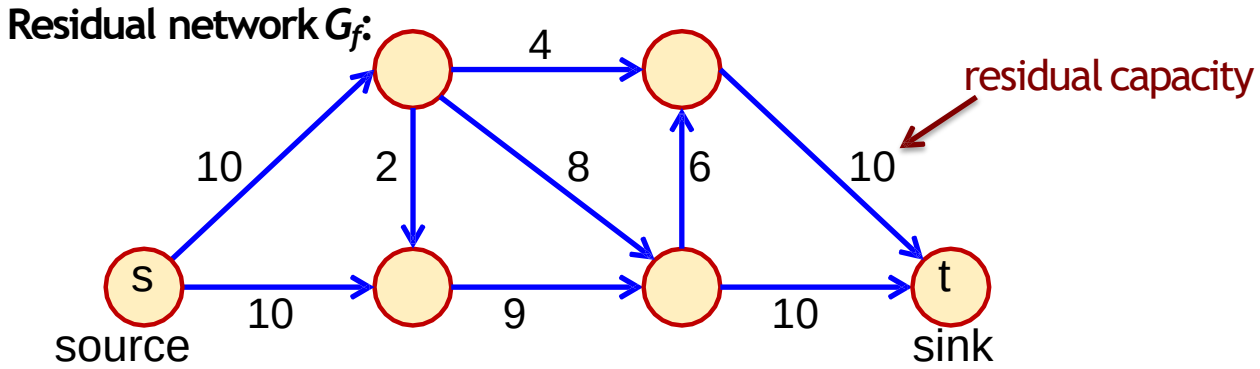
[tutorial slides from Ľubor Ladický]

Max Flow Problem – Ford & Fulkerson

Ford & Fulkerson algorithm (1956)

- flow=0
- Find a path from source to sink
- While (path exists)
  - flow += maximum capacity in the path
  - Build the residual graph
  - Find a path in the residual graph
- End



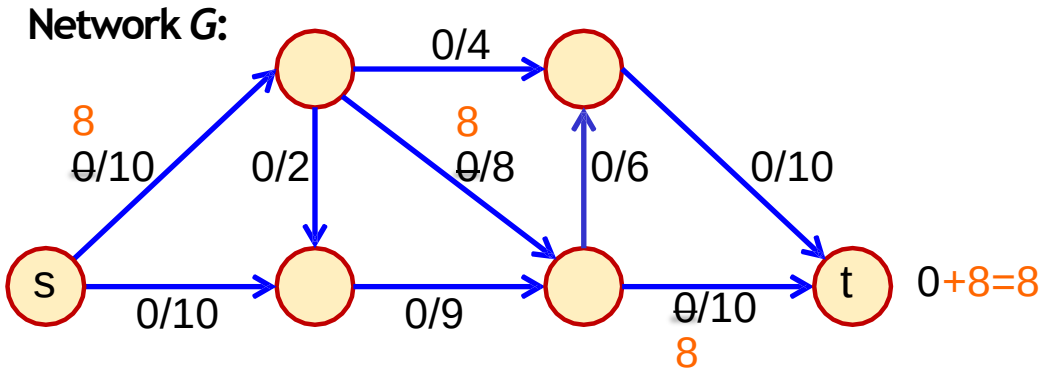


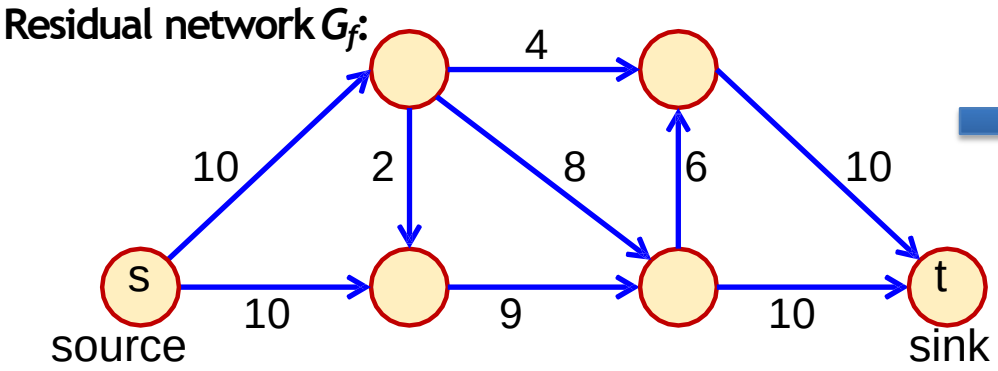
[tutorial slides from Ľubor Ladický]

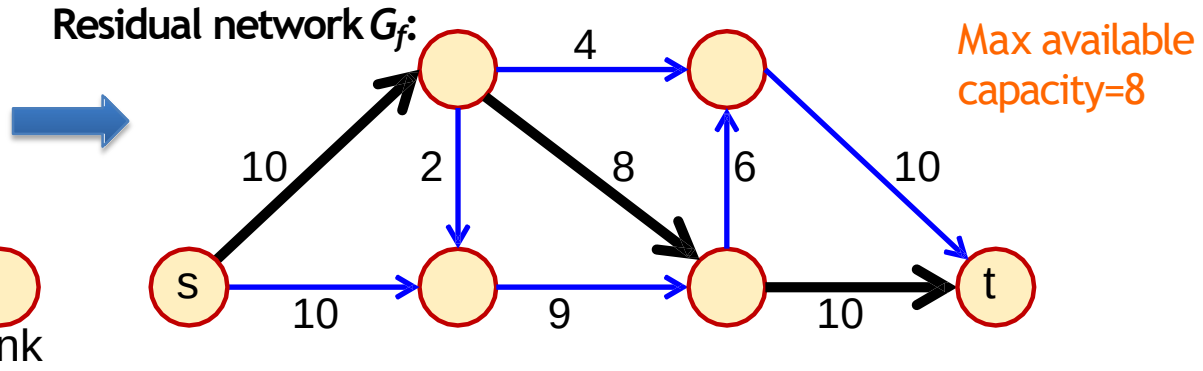
Max Flow Problem – Ford & Fulkerson

Ford & Fulkerson algorithm (1956)

- flow=0
- Find a path from source to sink
- While (path exists)
  - flow += maximum capacity in the path
  - Build the residual graph
  - Find a path in the residual graph
- End





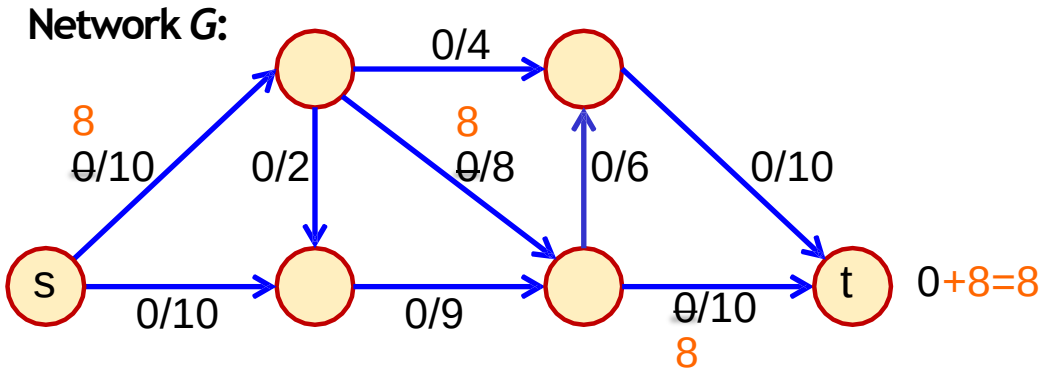


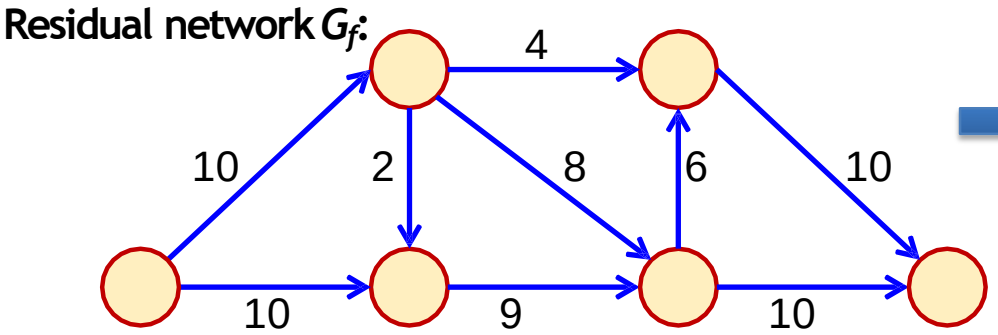
[tutorial slides from Ľubor Ladický]

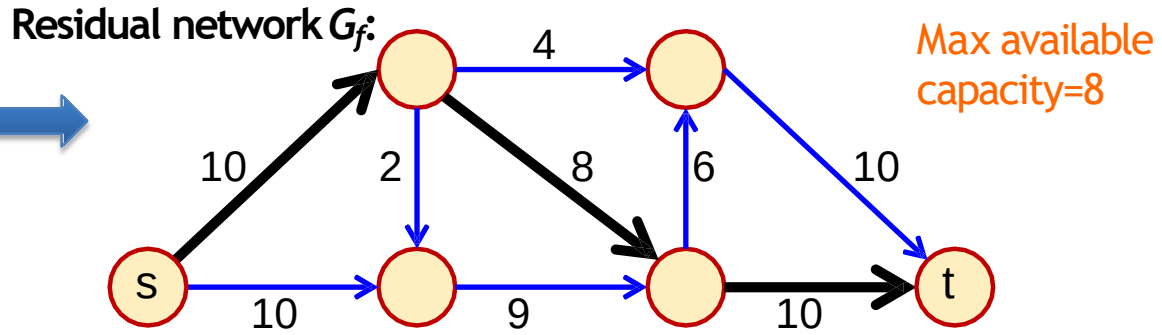
Max Flow Problem – Ford & Fulkerson

Ford & Fulkerson algorithm (1956)

- flow=0
- Find a path from source to sink
- While (path exists)
  - flow += maximum capacity in the path
  - Build the residual graph
  - Find a path in the residual graph
- End



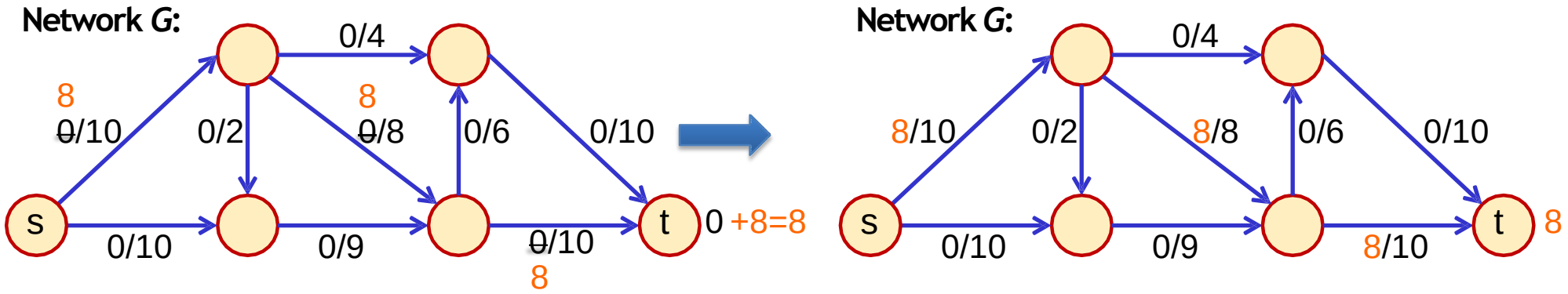




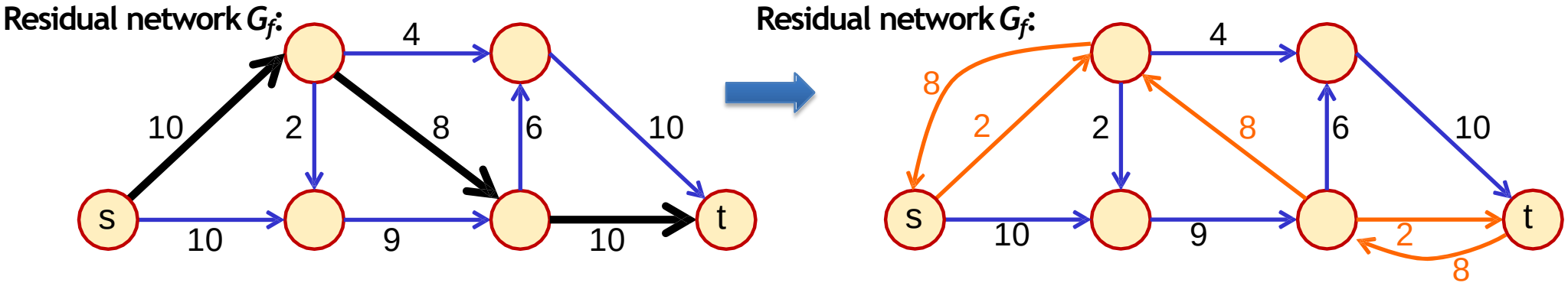
[tutorial slides from Ľubor Ladický]

Max Flow Problem – Ford & Fulkerson

flow/capacity



Max available capacity=8

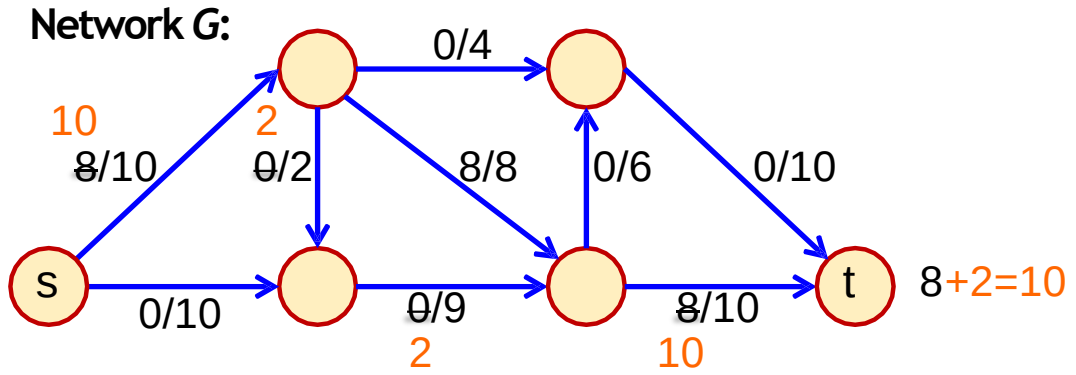


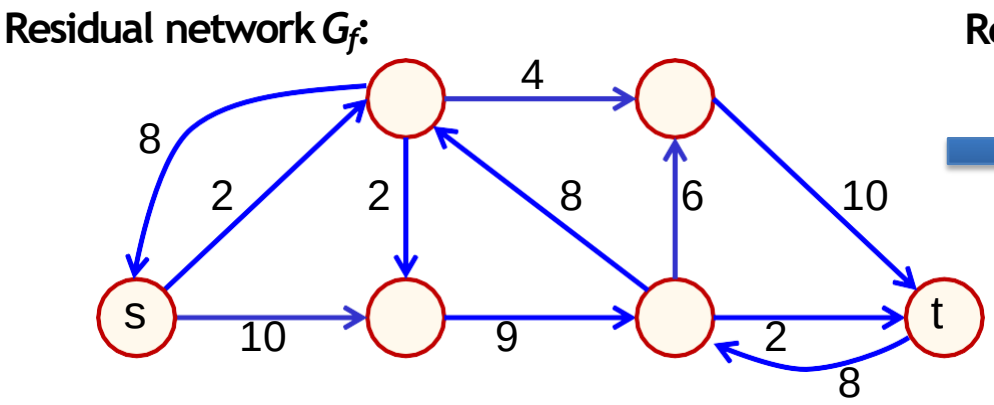
[tutorial slides from Ľubor Ladický]

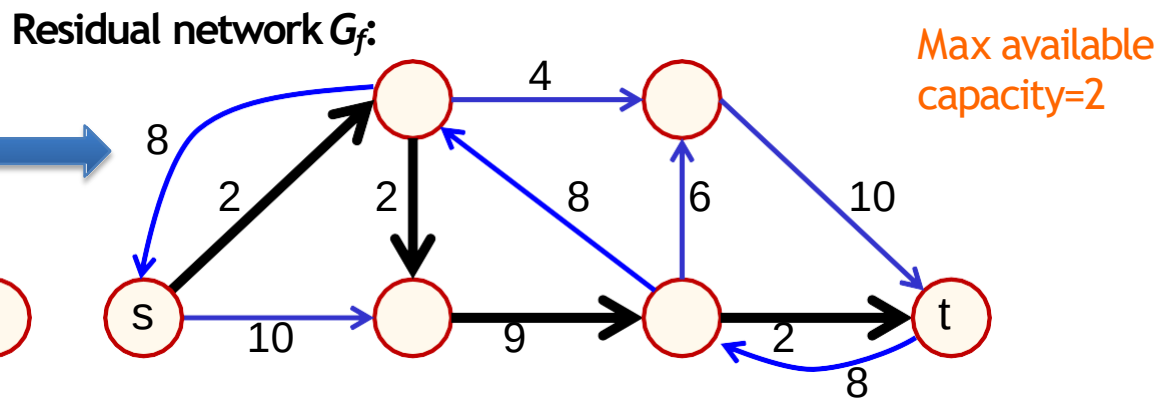
Max Flow Problem – Ford & Fulkerson

Ford & Fulkerson algorithm (1956)

- flow=0
- Find a path from source to sink
- While (path exists)
  - flow += maximum capacity in the path
  - Build the residual graph
  - Find a path in the residual graph
- End



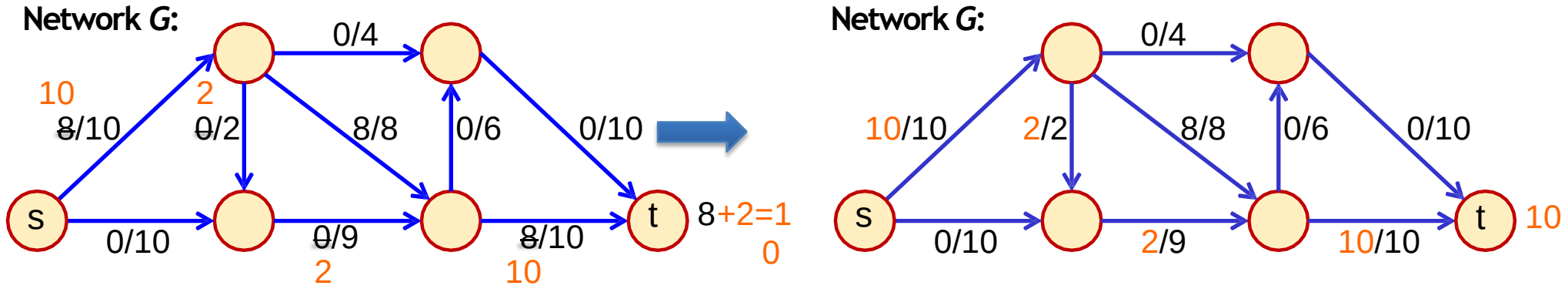




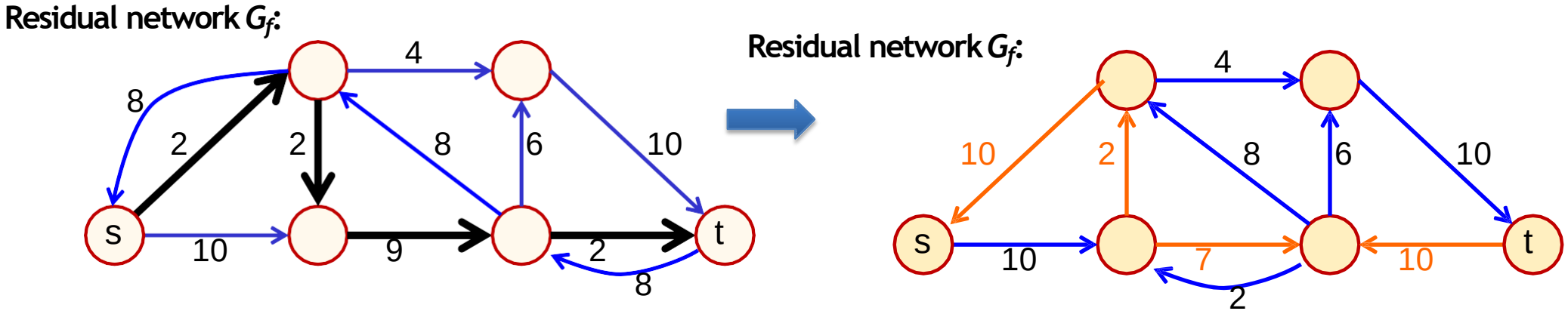
[tutorial slides from Ľubor Ladický]

Max Flow Problem – Ford & Fulkerson

flow/capacity



Max available capacity=2

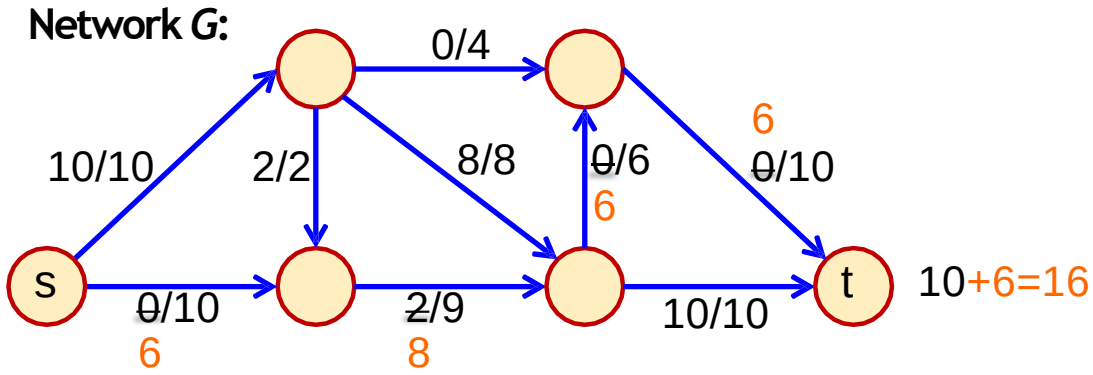


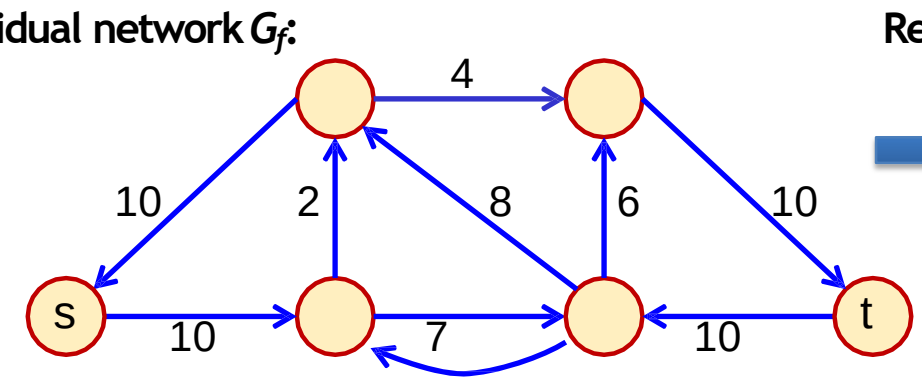
[tutorial slides from Ľubor Ladický]

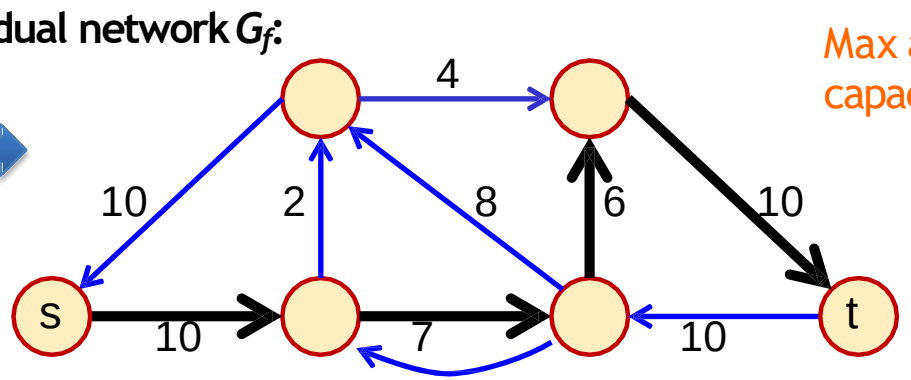
Max Flow Problem – Ford & Fulkerson

Ford & Fulkerson algorithm (1956)

- flow=0
- Find a path from source to sink
- While (path exists)
  - flow += maximum capacity in the path
  - Build the residual graph
  - Find a path in the residual graph
- End







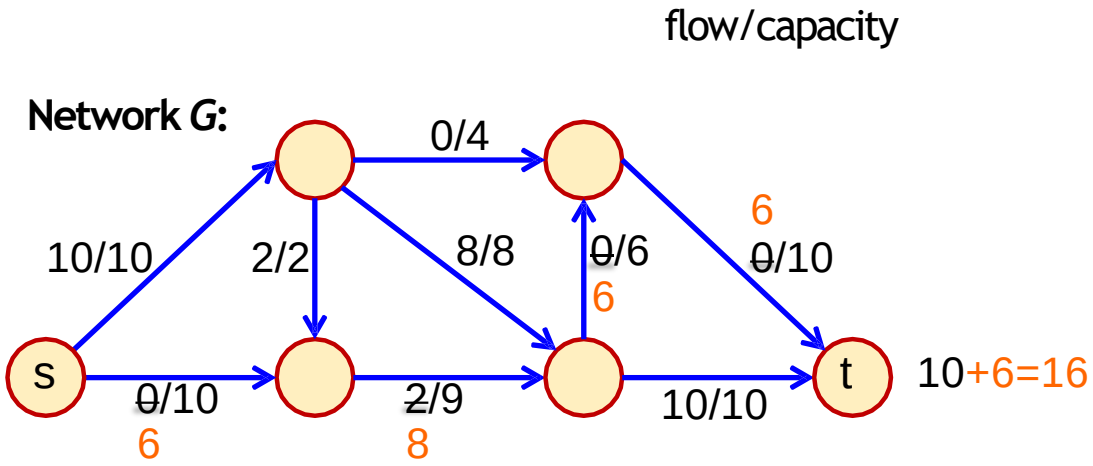
Max available capacity=6

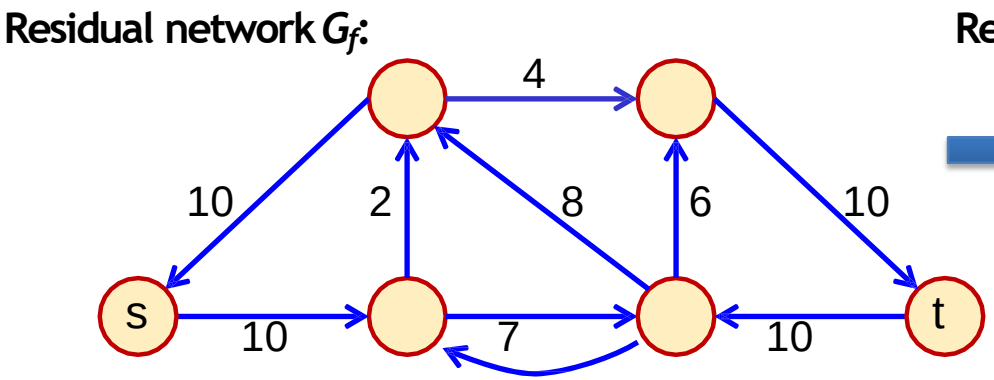
[tutorial slides from Ľubor Ladický]

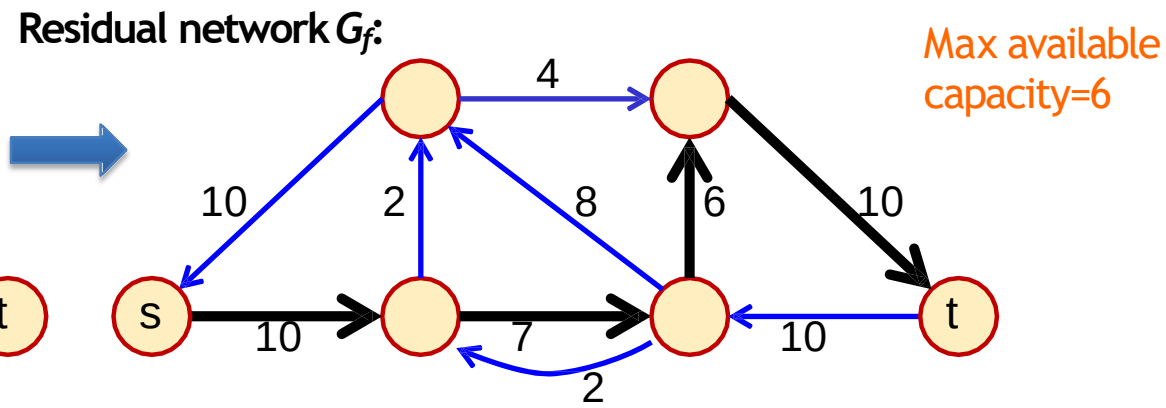
Max Flow Problem – Ford & Fulkerson

Ford & Fulkerson algorithm (1956)

- flow=0
- Find a path from source to sink
- While (path exists)
  - flow += maximum capacity in the path
  - Build the residual graph
  - Find a path in the residual graph
- End



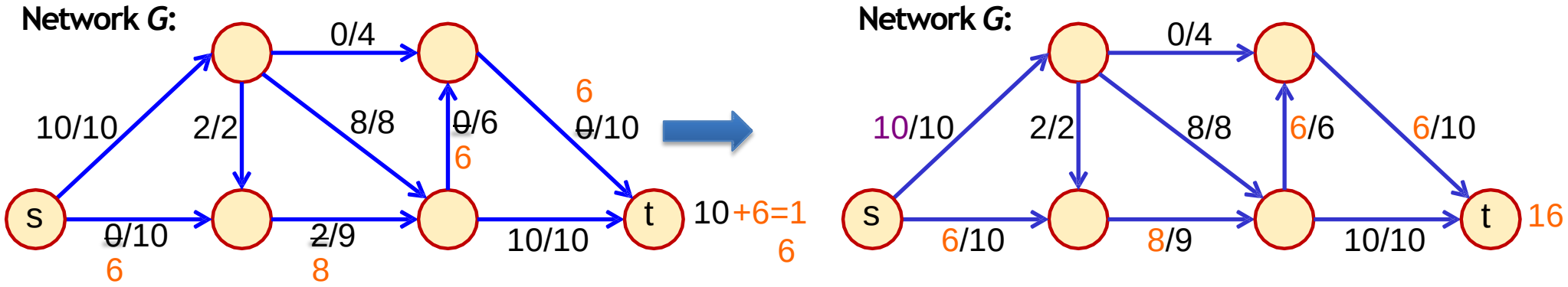




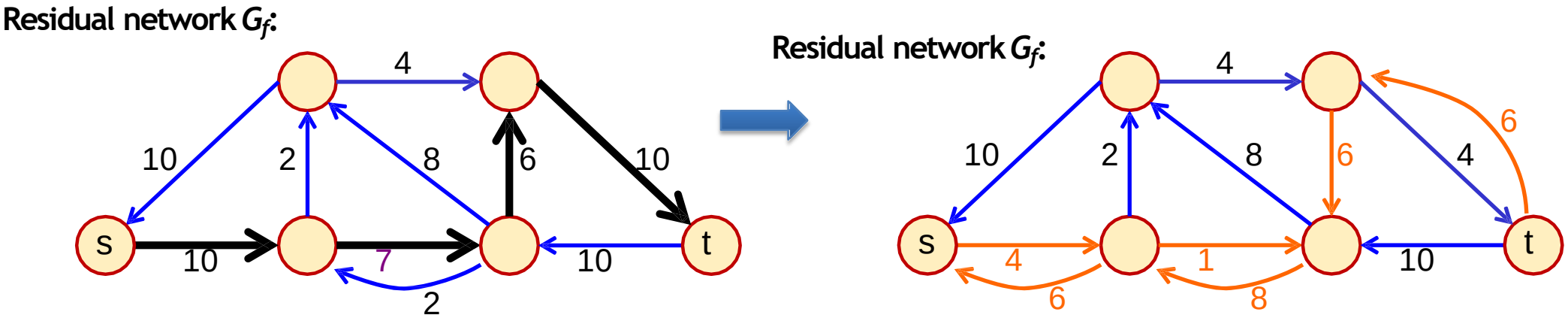
[tutorial slides from Ľubor Ladický]

Max Flow Problem – Ford & Fulkerson

flow/capacity



Max available capacity=6

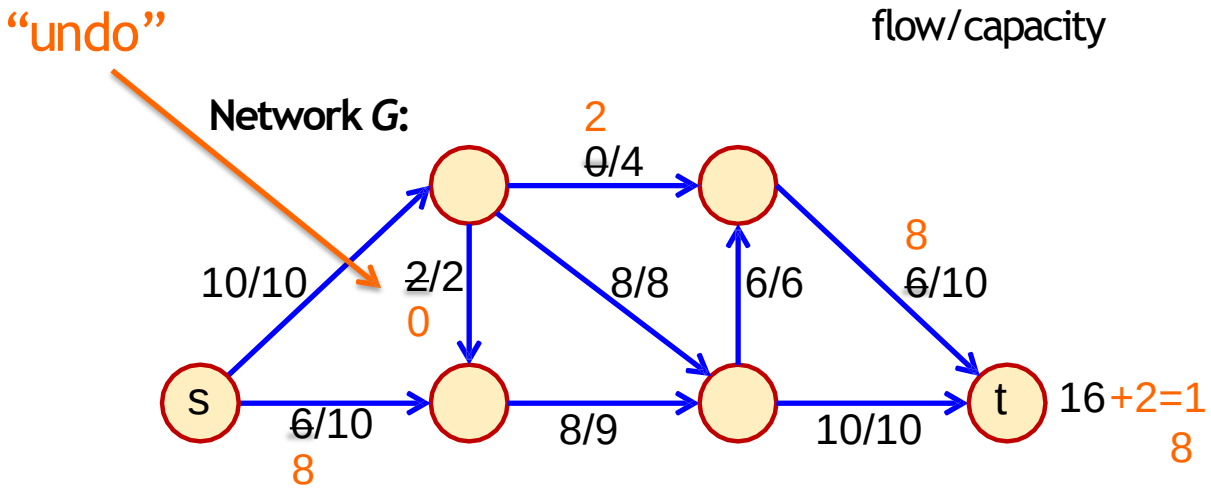


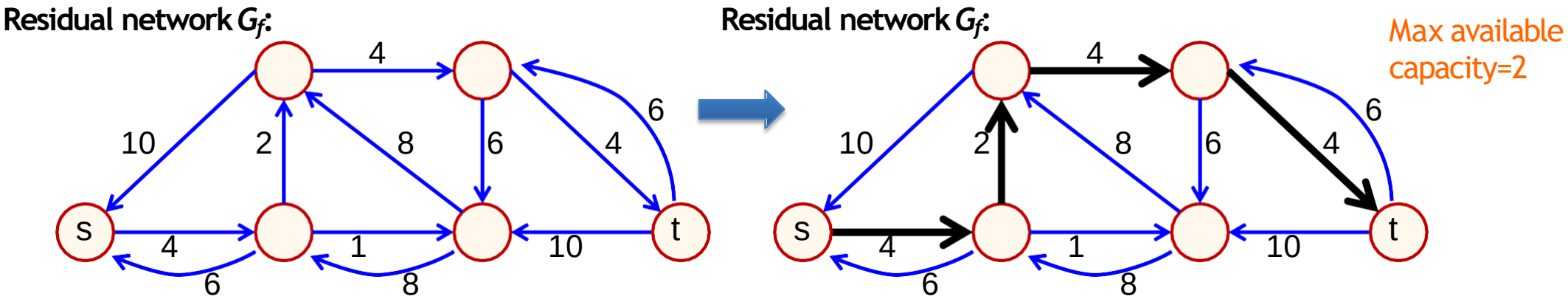
[tutorial slides from Ľubor Ladický]

Max Flow Problem – Ford & Fulkerson

Ford & Fulkerson algorithm (1956)

- flow=0
- Find a path from source to sink
- While (path exists)
  - flow += maximum capacity in the path
  - Build the residual graph
  - Find a path in the residual graph
- End

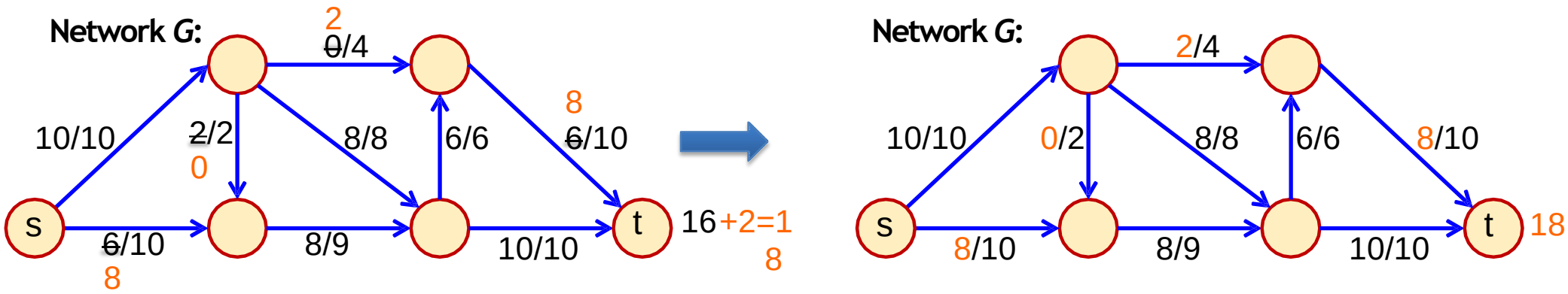




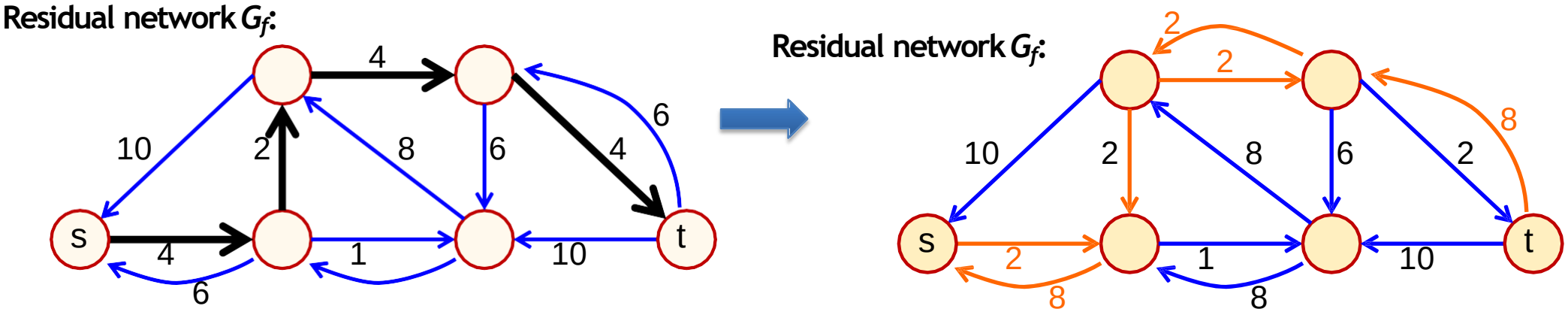
[tutorial slides from Ľubor Ladický]

Max Flow Problem – Ford & Fulkerson

flow/capacity



Max available capacity=2

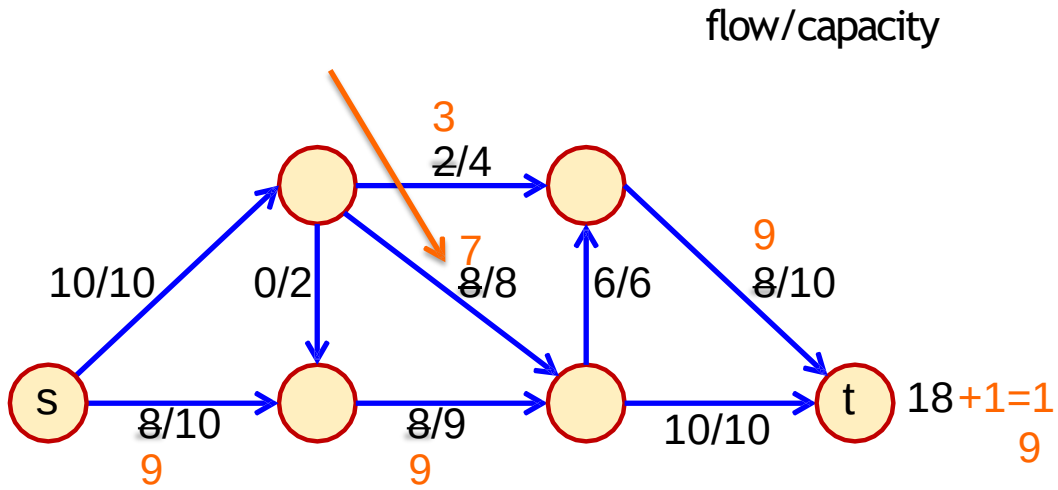


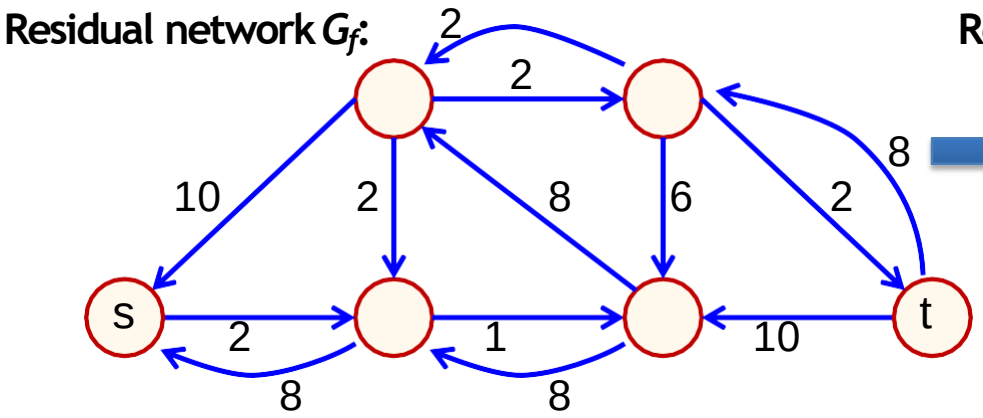
[tutorial slides from Ľubor Ladický]

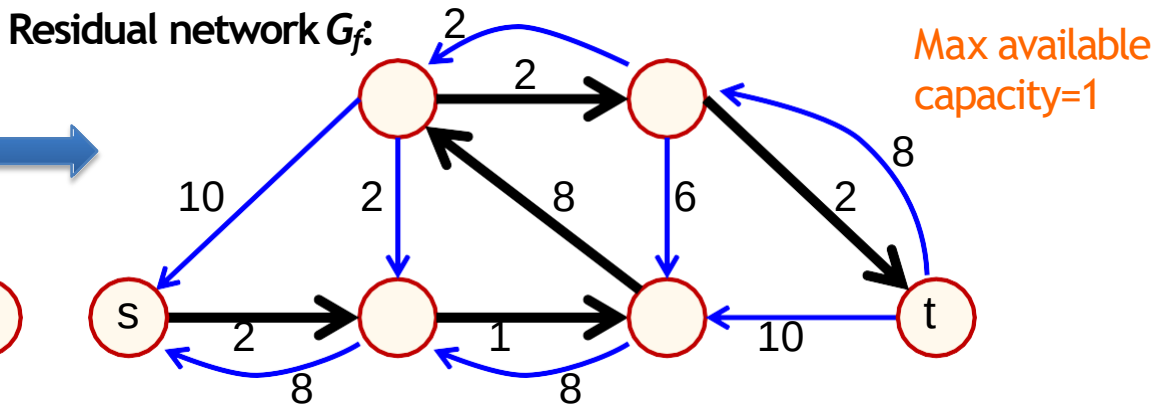
Max Flow Problem – Ford & Fulkerson

Ford & Fulkerson algorithm (1956)

- flow=0
- Find a path from source to sink
- While (path exists)
  - flow += maximum capacity in the path
  - Build the residual graph
  - Find a path in the residual graph
- End

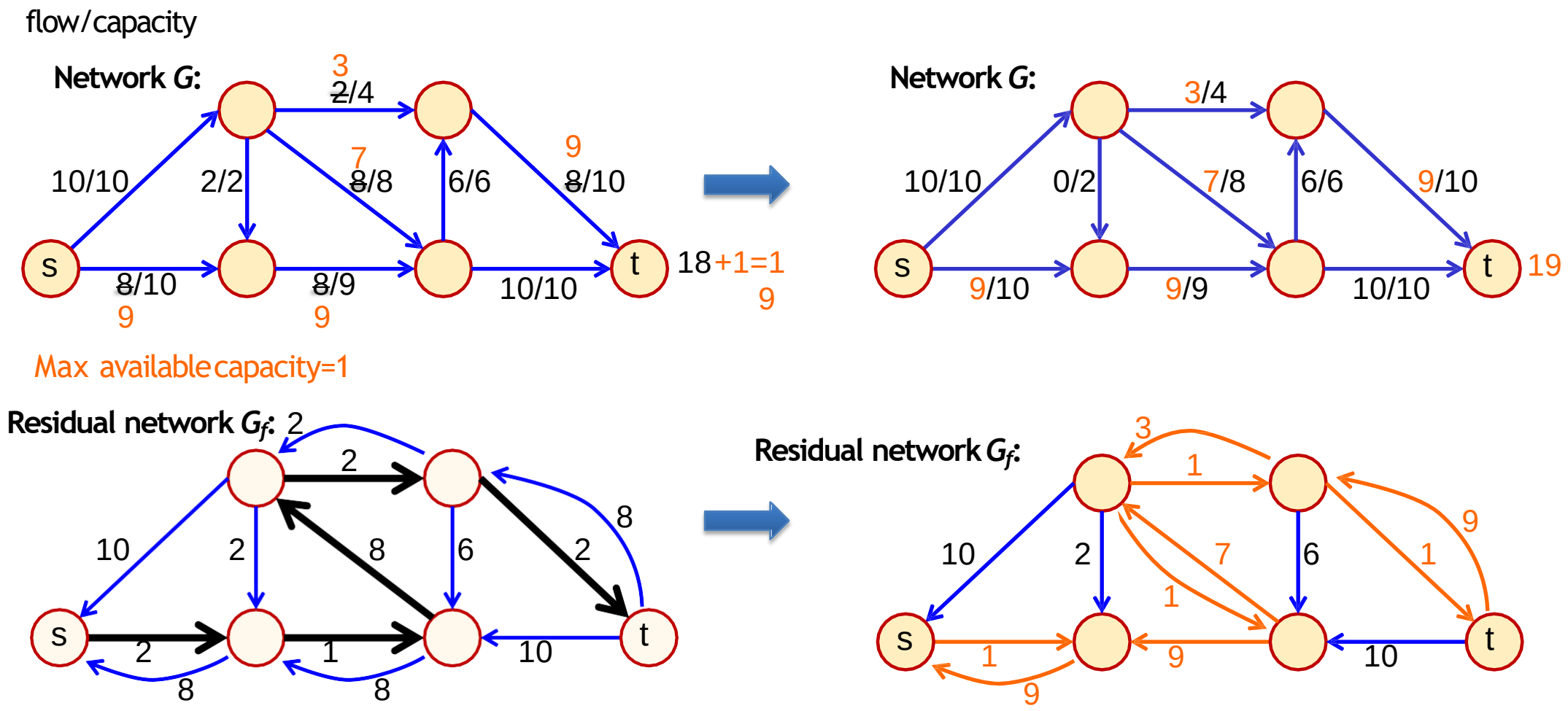






[tutorial slides from Ľubor Ladický]

Max Flow Problem – Ford & Fulkerson

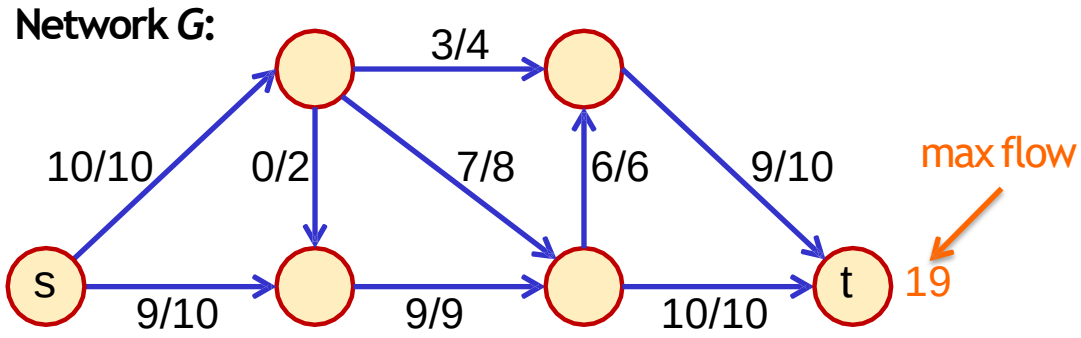


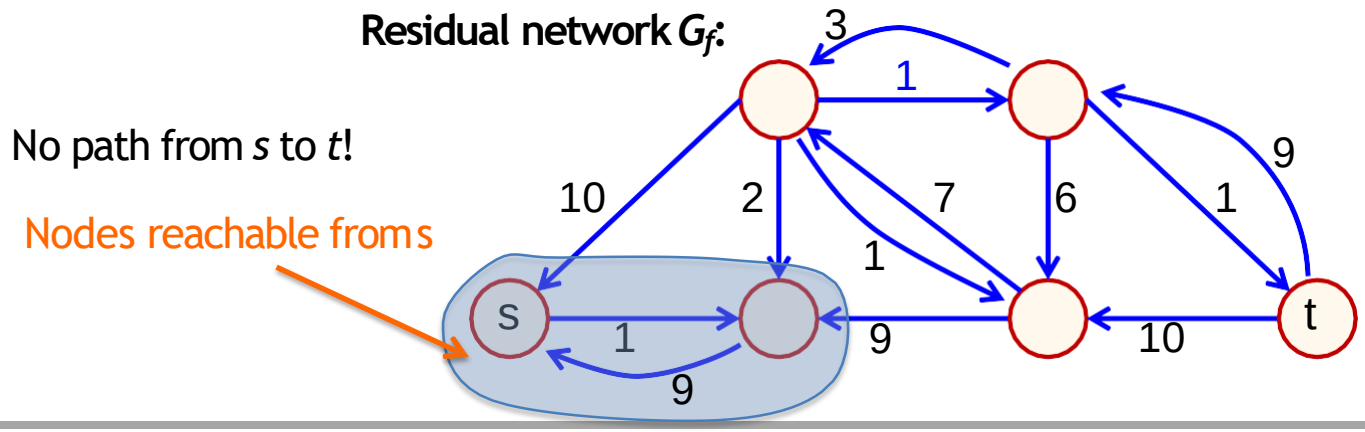
[tutorial slides from Ľubor Ladický]

Max Flow Problem – Ford & Fulkerson

Ford & Fulkerson algorithm (1956)

- flow=0
- Find a path from source to sink
- While (path exists)
  - flow += maximum capacity in the path
  - Build the residual graph
  - Find a path in the residual graph
- End





[tutorial slides from Ľubor Ladický]

Min Cut Problem – Image Segmentation

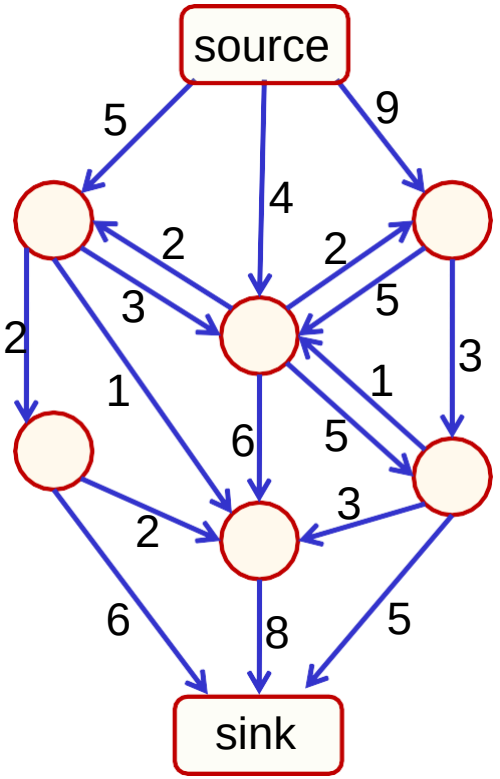
*** Min-Cut**

Task :

Minimize the cost of the cut

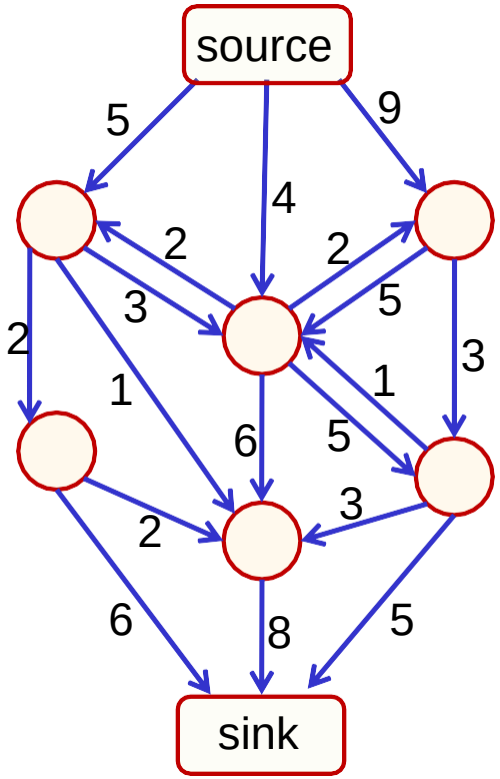
1) Each node is either assigned to the source S or sink T

2) The cost of the edge (i, j) is taken if (i ∈S) and (j ∈T)



[tutorial slides from Ľubor Ladický]

Min Cut Problem – Image Segmentation



edge costs

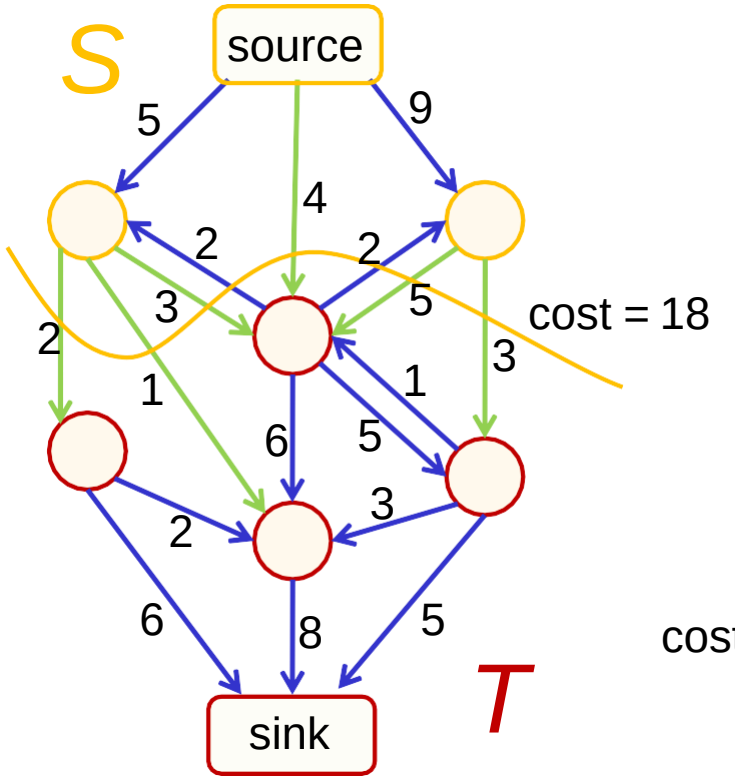
$$\min_{S,T} \sum_{i \in S, j \in T} c_{ij} \qquad s.t. \quad s \in S, \quad t \in T$$

source set

sink set

[tutorial slides from Ľubor Ladický]

Min Cut Problem – Image Segmentation



edge costs

$$\min_{S,T} \sum_{i \in S, j \in T} c_{ij} \qquad s.t. \quad s \in S, \quad t \in T$$

source set sink set

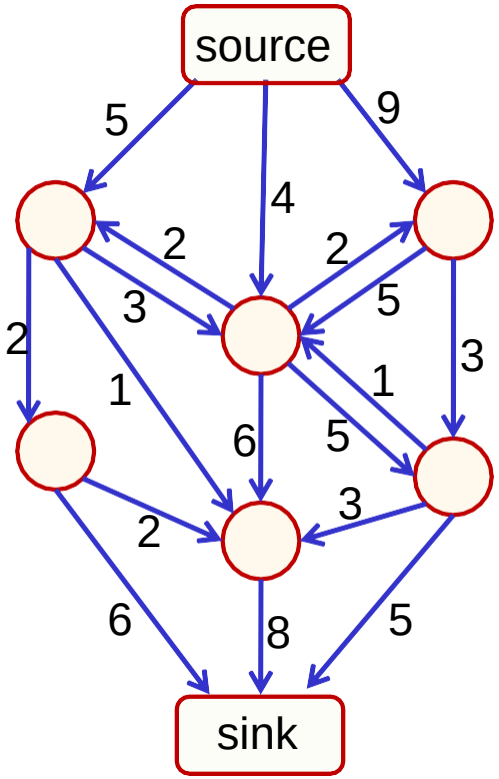
cost = sum of edge costs from source set S to sink test T

= 2+1+3+4+5+3

= 18

[tutorial slides from Ľubor Ladický]

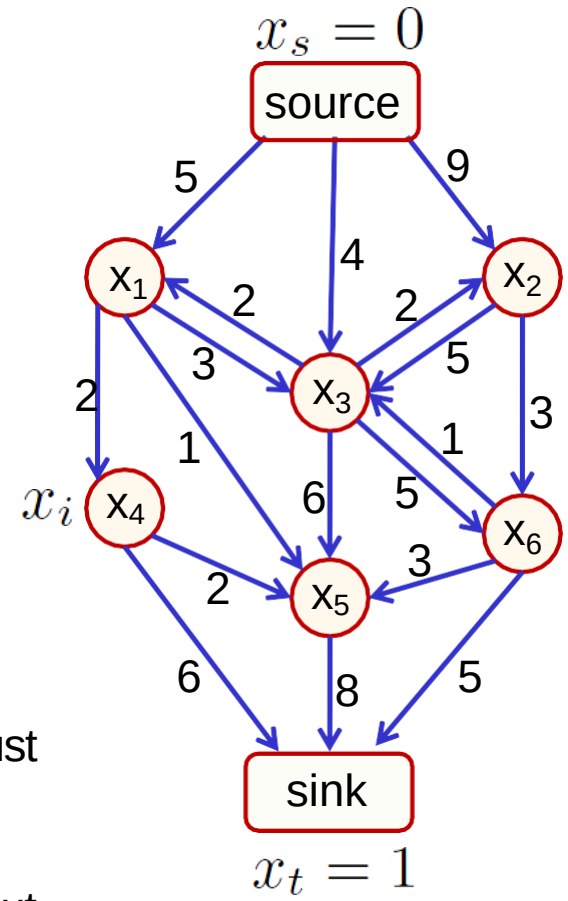
Min Cut Problem – Image Segmentation



Find the min-cut
i.e. find the partition that provides the lowest cut cost

$$x_i = 0 \implies x_i \in S$$
$$x_i = 1 \implies x_i \in T$$

Find the binary "partition variables"



***Min-cut** solution is equivalent to **Max-Flow** solution.*

→ A cut is a bottleneck; any flow through the network must all path through the bottleneck at once.

→ The total flow is limited by the narrow part, minimum cut.

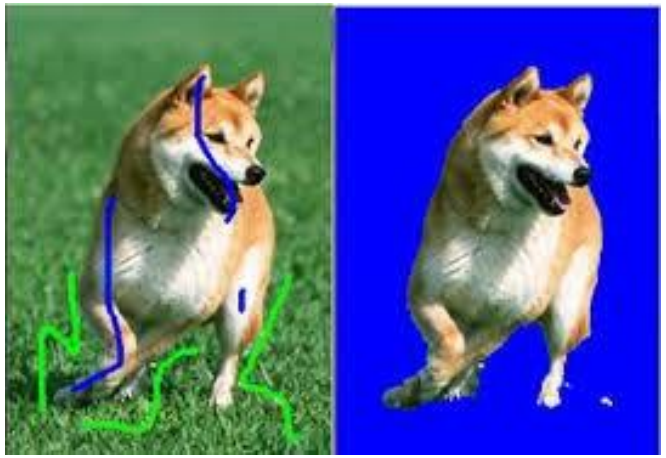
[tutorial slides from Ľubor Ladický]

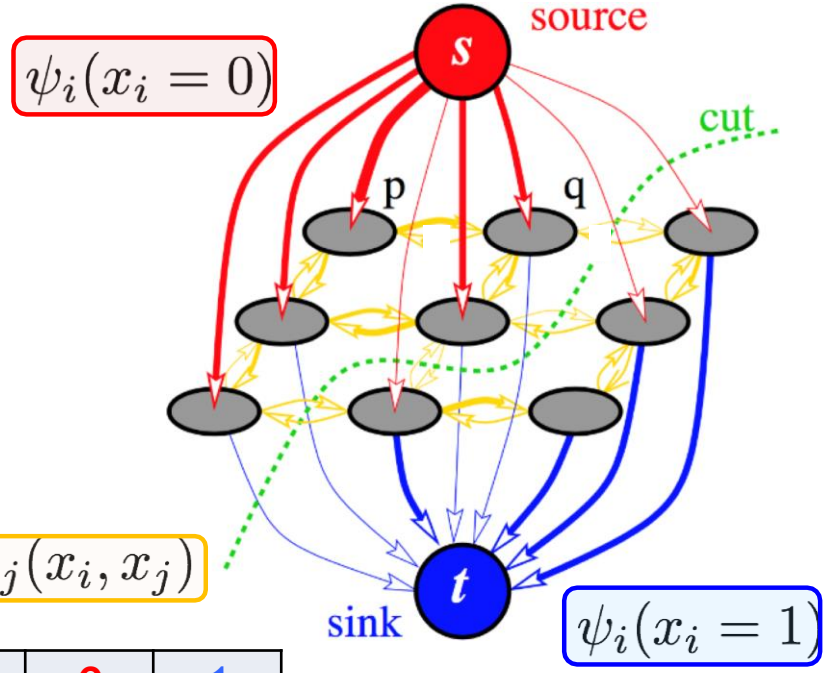
Min Cut Problem – Image Segmentation

$$E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \psi_i(x_i) + \sum_{i \in \mathcal{V}, j \in \mathcal{N}_i} \psi_{ij}(x_i, x_j)$$

Data term **Smoothness term**

$$\mathbf{x}^* = \arg\min_{x \in \mathbf{L}} E(\mathbf{x})$$





0	0	1
0	0	1
0	1	1

[tutorial slides from Ľubor Ladický]

Min Cut Problem – Image Segmentation

- Labeling problem : label pixels as foreground or background

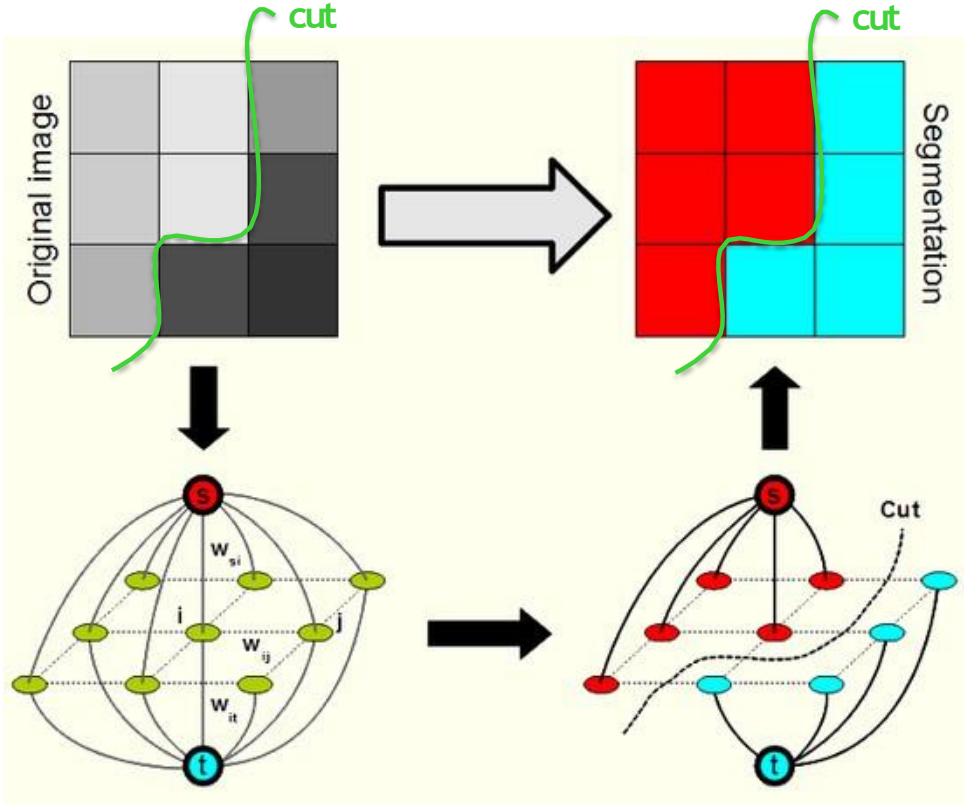


Image Source : http://www.ondrej-danek.net/en/research

[tutorial slides from Ľubor Ladický]

For More Dedicated Study on Graph Cuts…

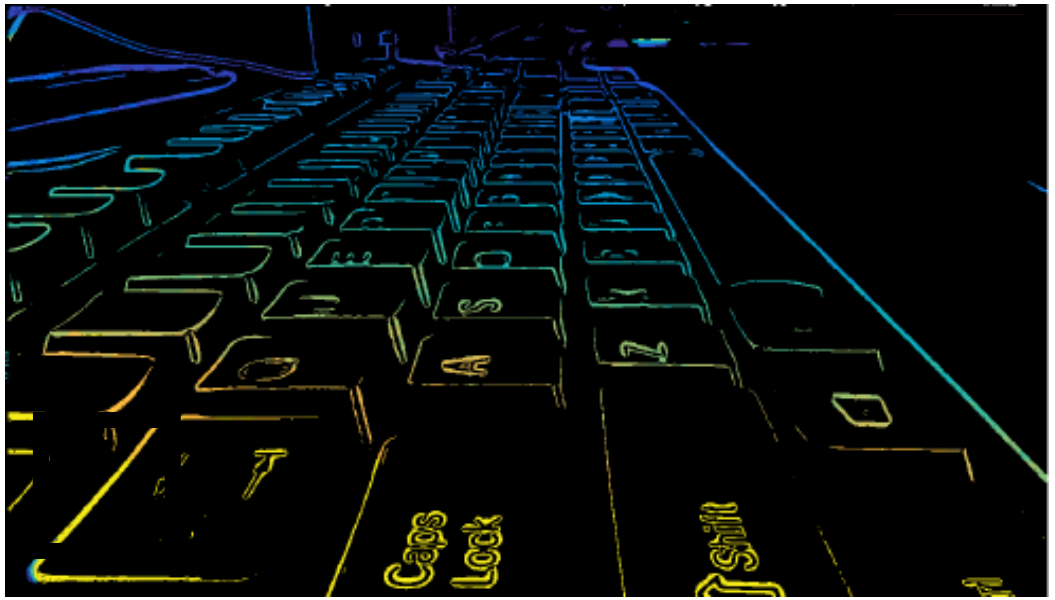
CVPR 2015 Tutorial on Energy Minimization and Discrete Optimization

Min Cut / Max Flow Algorithms, Alpha Expansion (multi label), etc.

https://inf.ethz.ch/personal/ladickyl/CVPR_Tutorial2015.htm
https://inf.ethz.ch/personal/ladickyl/gcut_cvpr15_part1.pdf
https://my.eng.utah.edu/~cs6320/cv_files/GraphCuts.pdf

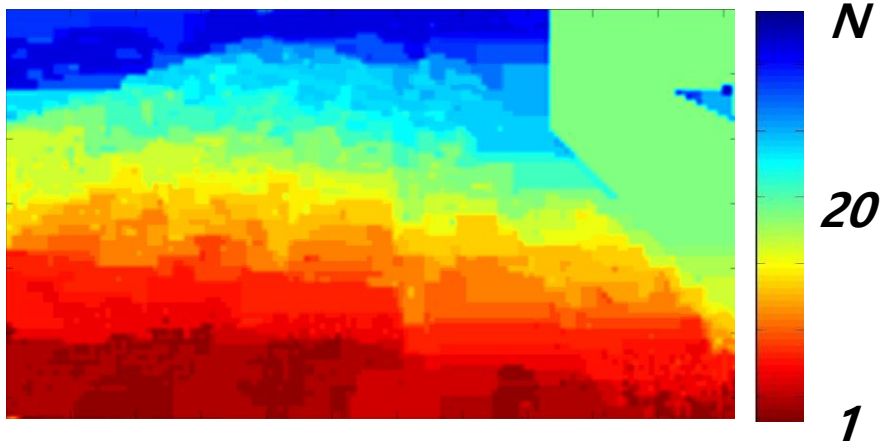
Ľubor Ladický

Depth(Index) Map Propagation Problem



Sparse depth map – confident (valid) pixels

[Multi-label Segmentation Problem]



Graph-Cut for Depth(Index) Map Propagation

$$E(\mathbf{x}) = \sum_{i \in S} \psi_i(x_i) + \sum_{i \in \mathcal{V}, j \in \mathcal{N}_i} \psi_{ij}(x_i, x_j)$$

Data term **Smoothness term**

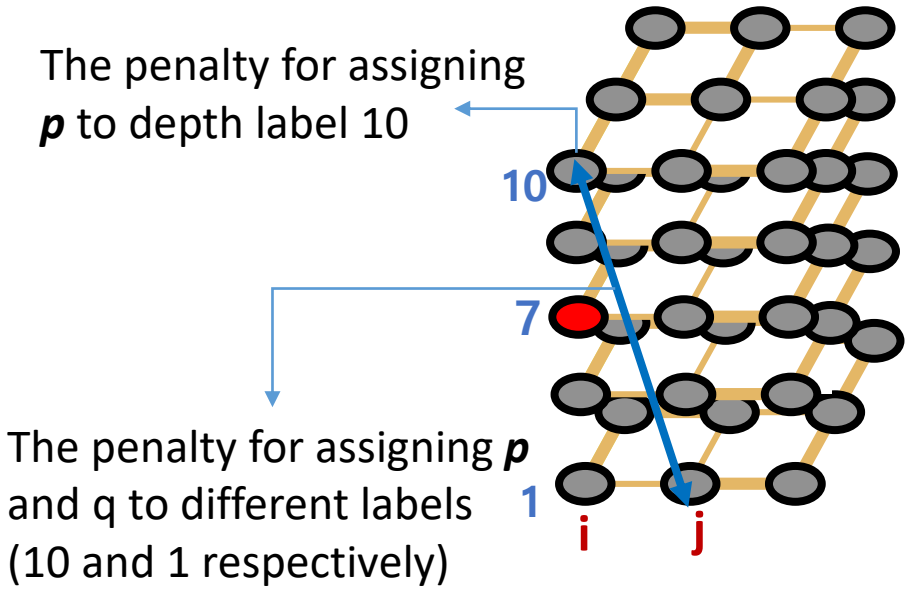
Multi-label segmentation

- **Data term** : $\psi_i(x_i)$

Key Idea. Assign close-by-value depth
for known data g_i (from the initial depth)

- **Smoothness term** : $\psi_{ij}(x_i, x_j) = |x_i - x_j|$
  10 1

Key Idea. Assign similar depth for neighbor pixels

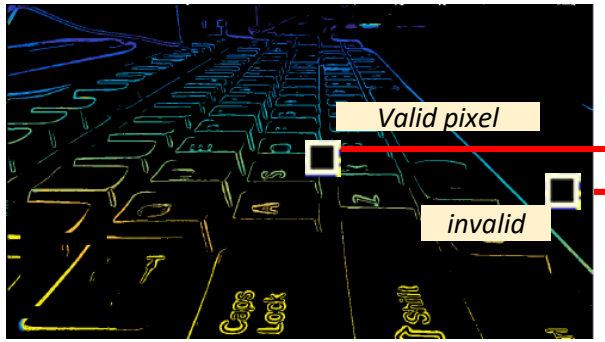


Another Energy Term Design

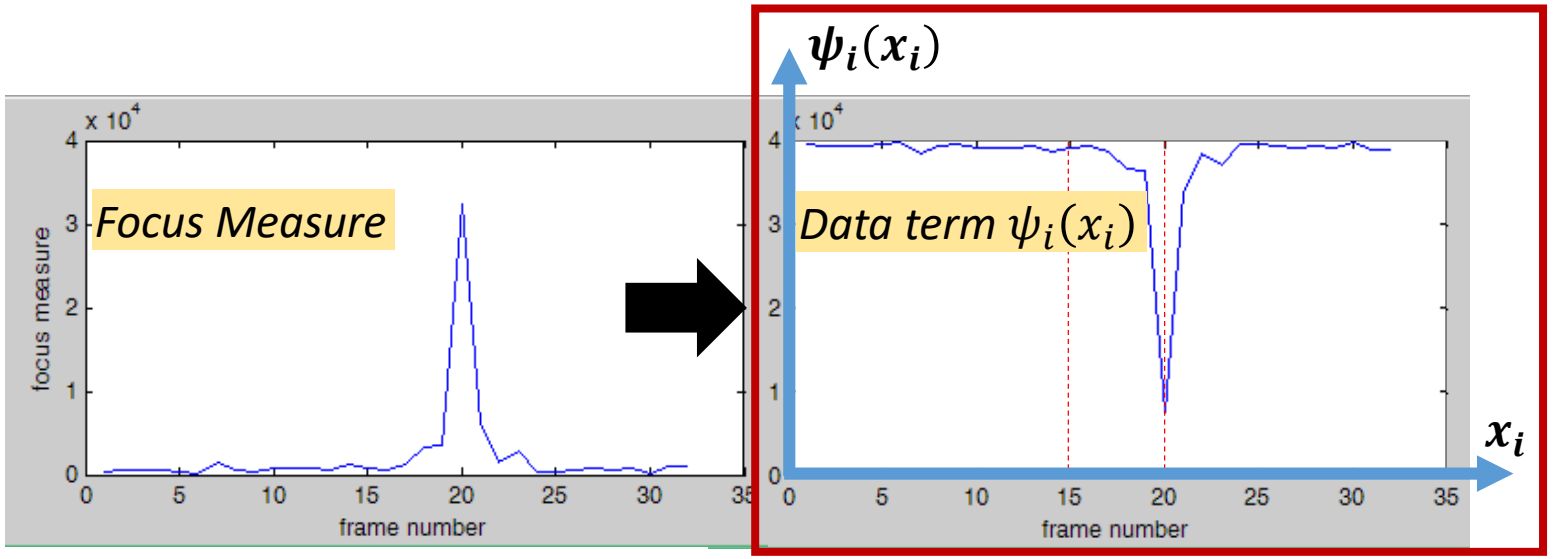
- **Data term** : $\psi_i(x_i)$

Key Idea. Assign close-by-value depth for known pixels

`for each pixel:`



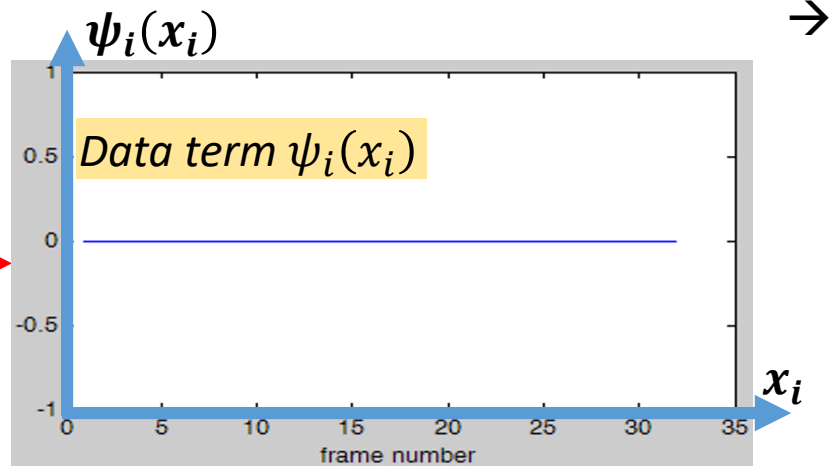
Sparse Depth(Index) Map



→ *s.t. Pixel i prefers label **20***

- Data term matrix **`mat_D`**

`mat_D` : `N x H x W`
`N` : `# frames`
`H,W` : `image height, width`

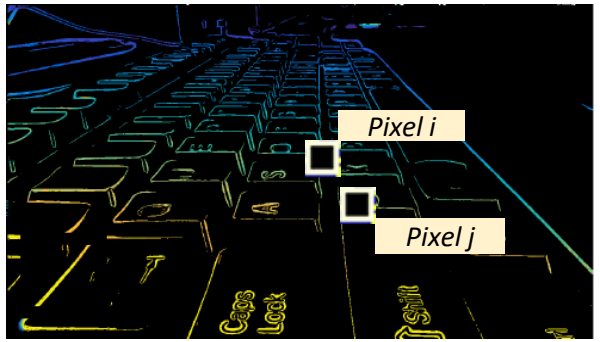


→ *s.t. Pixel i shows **flat** preferences*

Another Energy Term Design

- **Smoothness term** : $\psi_{ij}(x_i, x_j)$ Key Idea. Assign similar depth for neighbor pixels

for each pixel pair (i,j):



Sparse Depth(Index) Map

$$\psi_{ij}(x_i, x_j) = \begin{cases} |x_i - x_j| & \text{if } j \text{ is neighbor of } i \\ 0 & \text{Otherwise, ignore} \end{cases}$$

- Smoothness term matrix **mat_S**

mat_S : N x N
N : # frames

mat_S

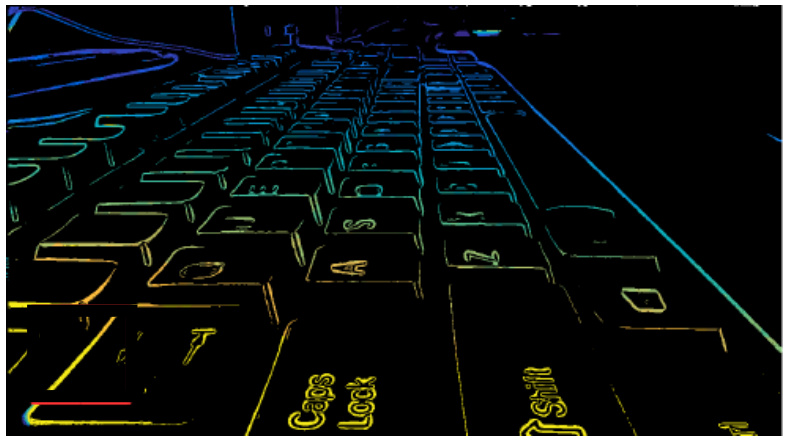
← N →

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12
	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11
	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10
↑	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9
	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8
N	7	6	5	4	3	2	1	0	1	2	3	4	5	6	7
	8	7	6	5	4	3	2	1	0	1	2	3	4	5	6
	9	8	7	6	5	4	3	2	1	0	1	2	3	4	5
↓	10	9	8	7	6	5	4	3	2	1	0	1	2	3	4
	11	10	9	8	7	6	5	4	3	2	1	0	1	2	3
	12	11	10	9	8	7	6	5	4	3	2	1	0	1	2
	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1
	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Graph-Cut for Depth(Index) Map Propagation

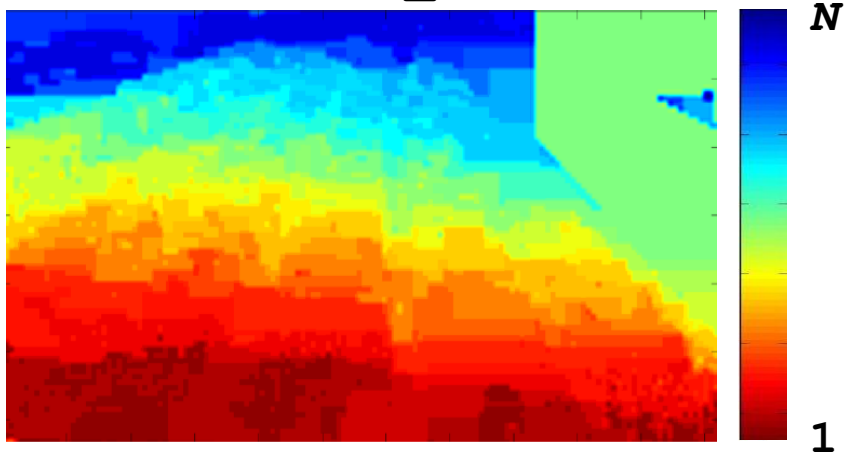
- **Optimization**
  Using GCO-v3 MATLAB library

```
h = GCO_Create(H*W,N);
GCO_SetDataCost(h,int32(mat_D));
GCO_SetSmoothCost(h,int32(mat_S));
                ⋮
Labeled_data = GCO_GetLabeling(h); % (H*W)
```



Sparse Depth (Index) map

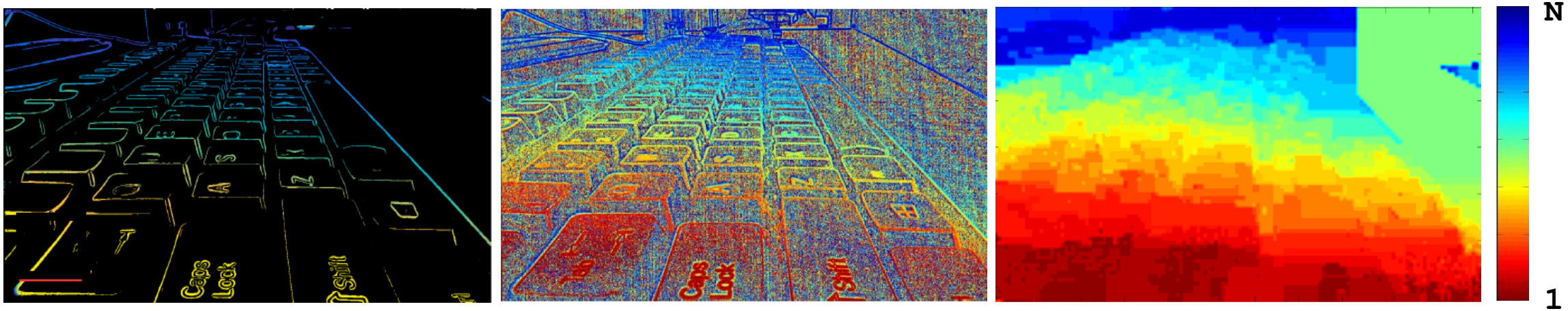
Labeled_data



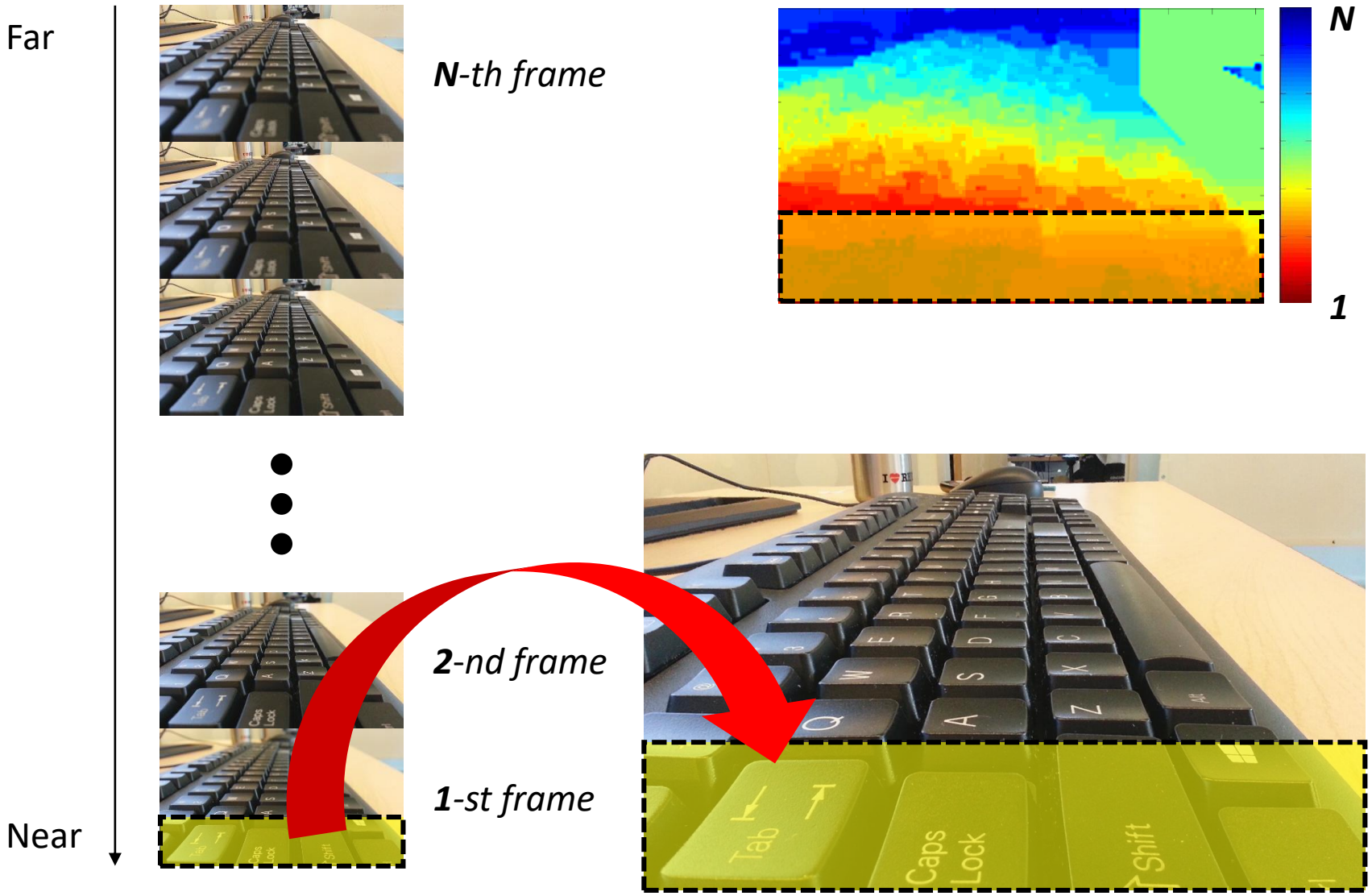
Graph-Cut Results

Library tutorial : https://github.com/nsubtil/gco-v3.0/tree/master/matlab

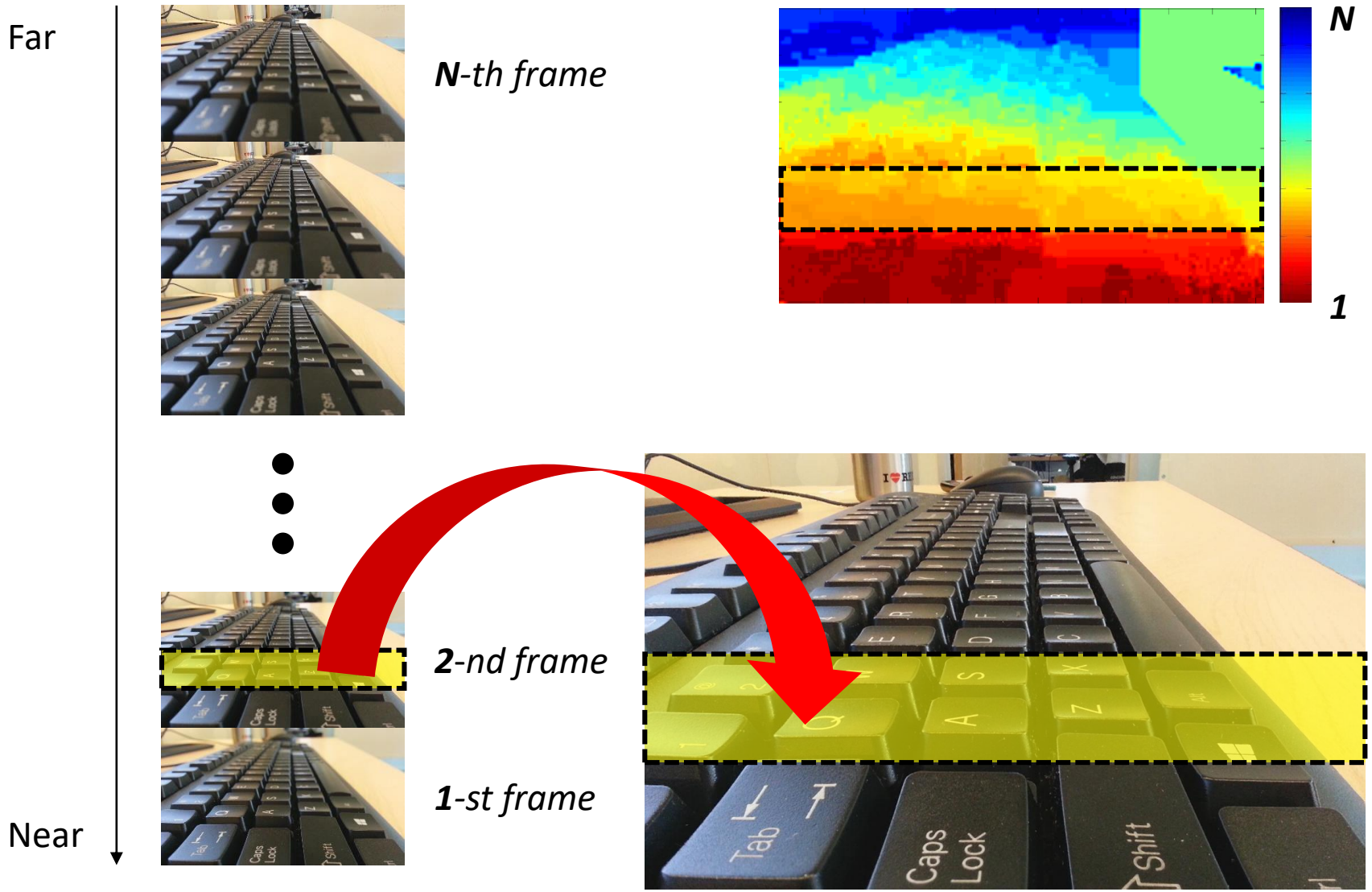
Sparse-to-Dense Depth Map Propagation



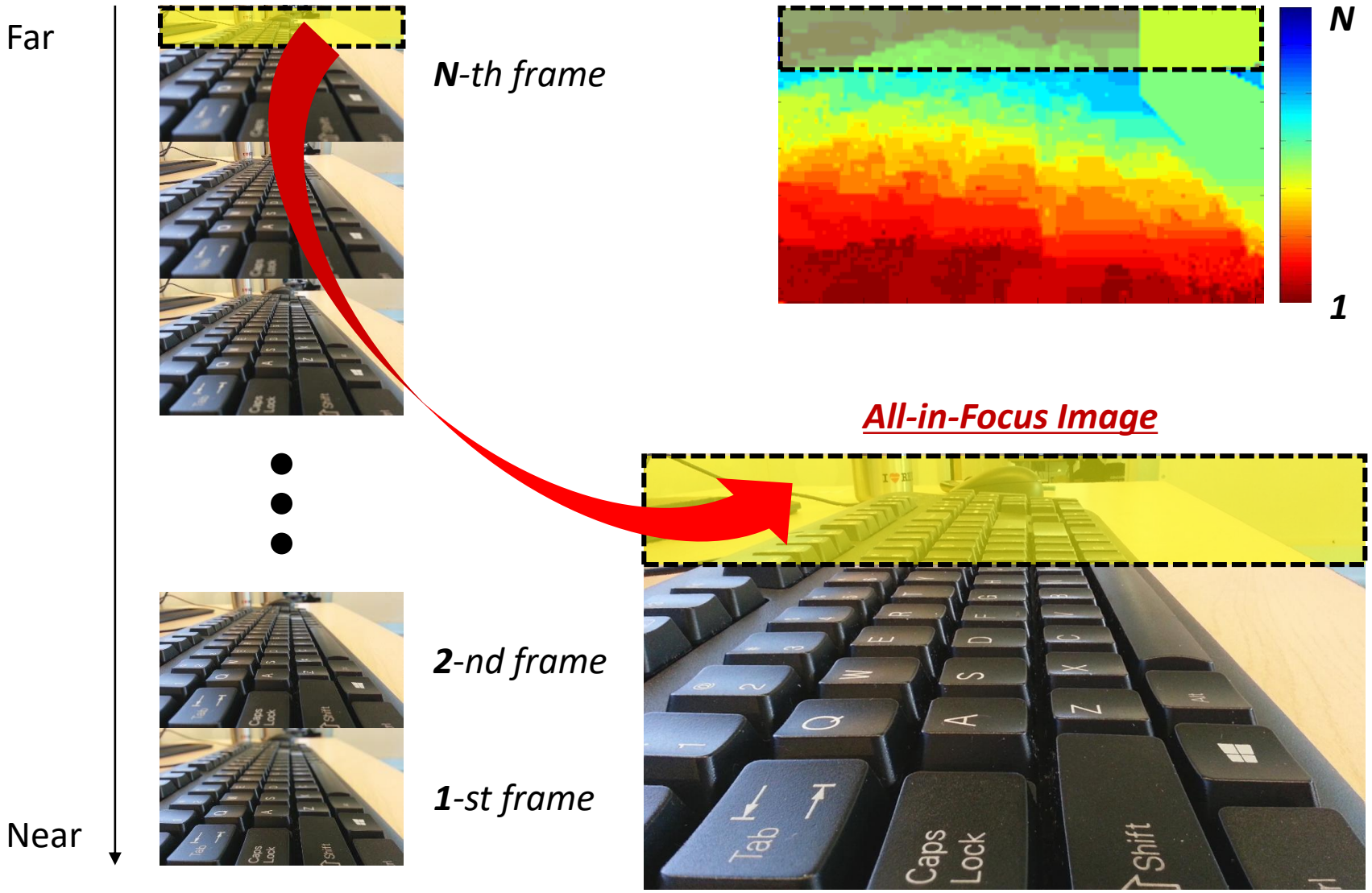
All-in-Focus Image Stitching



All-in-Focus Image Stitching

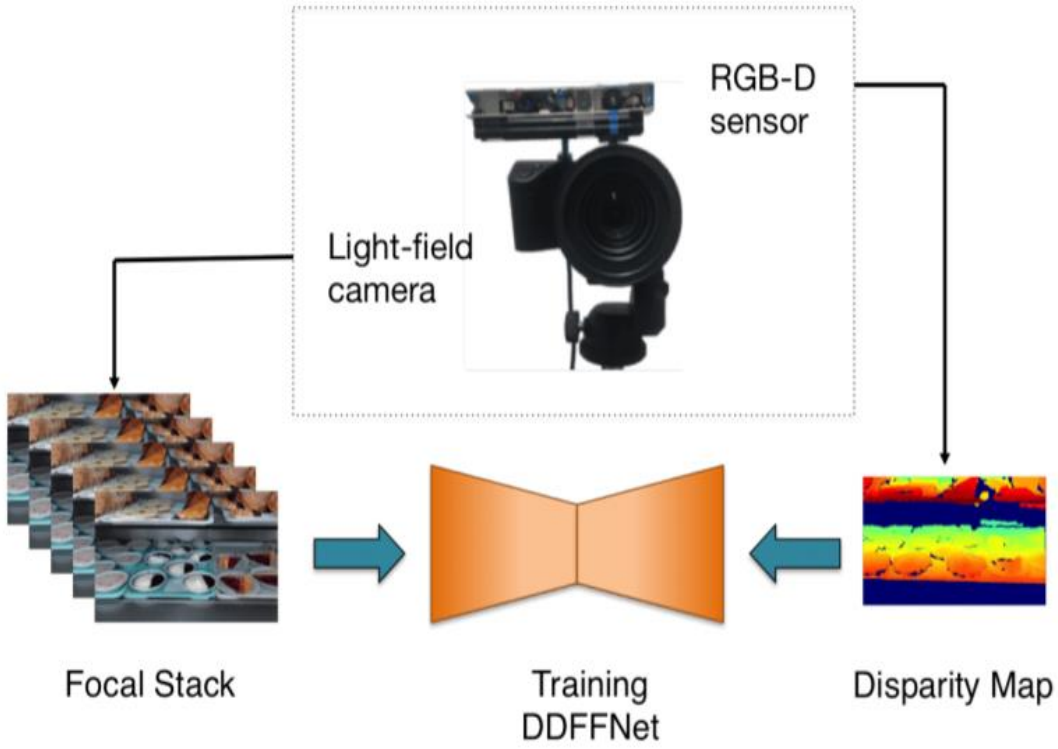


All-in-Focus Image Stitching



Network Architecture

- Use deep learning to reconstruct a pixel-accurate depth map from a given a stack of i mages with gradually changing optical focus



- Training DNNs require a significant amount of data.
- Manually obtaining a stack of refocused images is time consuming
- Light field imaging
  - ✓ One image per scene needs to be taken, meaning all images will have the same photographic exposure and the capturing process will be efficient.
  - ✓ Refocusing can be performed digitally, which allows to easily generate stacks with different focal steps
- RGB-D sensor for ground truth depth maps
- DDFF 12-Scene benchmark
  - ✓ Dataset in 12 different indoor environments

Deep Depth From Focus, *Caner Hazirbas, Sebastian Georg Soyer, Maximilian Christian Staab, Laura Leal-Taix´e, and Daniel Cremers, ACCV 2018*

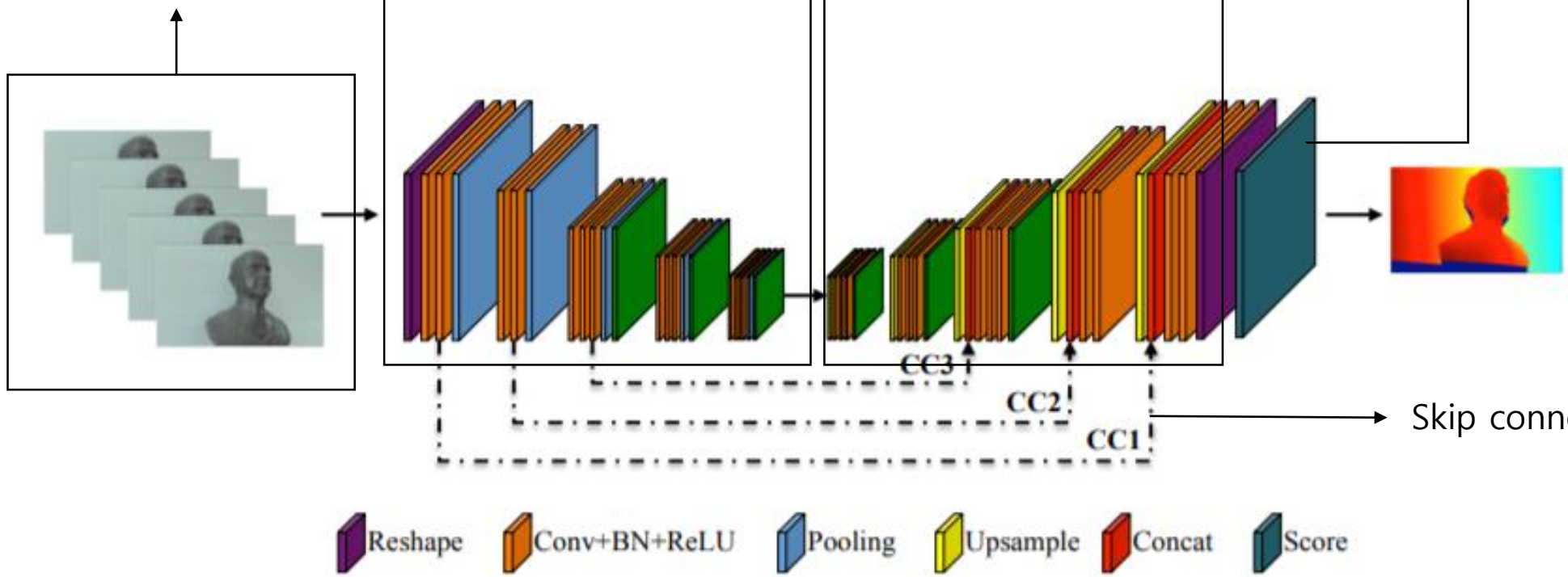
Network Architecture

A stack of images with gradually changing optical focus
Input size: B * S x H X W x 3

Encoder
VGG-16 network with the fc layers removed

Decoder
Mirrored encoder with 2 x 2 layers replaced with deconvolution

Regression layer
1 x 1 convolution to regress depth from out put feature maps



Deep Depth From Focus, *Caner Hazirbas, Sebastian Georg Soyer, Maximilian Christian Staab, Laura Leal-Taix´e, and Daniel Cremers, ACCV 2018*

Quantitative Results

	Method	MSE ↓	RMS ↓	log RMS ↓	Abs. rel. ↓	Sqr. rel.↓	Accuracy ($\delta = 1.25$) δ ↑	δ^2 ↑	δ^3 ↑	Bump. ↓
DDFFNet	Unpool	$2.9\,e^{-3}$	0.050	0.50	0.64	0.05	39.95	63.46	78.26	0.62
	BL	$2.1\,e^{-3}$	0.041	0.43	0.46	0.03	51.29	74.81	85.28	0.54
	UpConv	$1.4\,e^{-3}$	0.034	0.33	0.30	0.02	52.41	83.09	93.78	0.54
	CC1	$1.4\,e^{-3}$	0.033	0.33	0.37	0.02	60.38	82.11	90.63	0.75
	CC2	$1.8\,e^{-3}$	0.039	0.39	0.39	0.02	44.80	76.27	89.15	0.75
	CC3	$9.7\,e^{-4}$	0.029	0.32	0.29	0.01	61.95	85.14	92.99	0.59
	PSPNet	$9.4\,e^{-4}$	0.030	0.29	0.27	0.01	62.66	85.90	94.42	0.55
	Lytro	$2.1\,e^{-3}$	0.040	0.31	0.26	0.01	55.65	82.00	93.09	1.02
	PSP-LF	$2.7\,e^{-3}$	0.046	0.45	0.46	0.03	39.70	65.56	82.46	0.54
	DFLF	$4.8\,e^{-3}$	0.063	0.59	0.72	0.07	28.64	53.55	71.61	0.65
	VDFF	$7.3\,e^{-3}$	0.080	1.39	0.62	0.05	8.42	19.95	32.68	0.79

Table 1: **Quantitative results.** *DDFFNet-CC3* is the best depth from focus method and provides also better results compared to Lytro, *i.e.* depth from light-field. Metrics are computed on the predicted and the groundtruth disparity maps.

- MSE : $\frac{1}{|\mathcal{M}|}\sum_{p\in\mathcal{M}}\left\|f(\mathcal{S}_p)-D_p\right\|_2^2$
- RMS : $\sqrt{\frac{1}{|\mathcal{M}|}\sum_{p\in\mathcal{M}}\left\|f(\mathcal{S}_p)-D_p\right\|_2^2}$
- log RMS : $\sqrt{\frac{1}{|\mathcal{M}|}\sum_{p\in\mathcal{M}}\left\|\log f(\mathcal{S}_p)-\log D_p\right\|_2^2}$
- Absolute relative : $\frac{1}{|\mathcal{M}|}\sum_{p\in\mathcal{M}}\frac{|f(\mathcal{S}_p)-D_p|}{D_p}$
- Squared relative : $\frac{1}{|\mathcal{M}|}\sum_{p\in\mathcal{M}}\frac{\left\|f(\mathcal{S}_p)-D_p\right\|_2^2}{D_p}$
- Accuracy: % of D_p s.t $\max\left(\frac{f(\mathcal{S}_p)}{D_p},\frac{D_p}{f(\mathcal{S}_p)}\right)=\delta<thr$
- BadPix(τ): $\frac{|\{p\in\mathcal{M}:|f(\mathcal{S}_p)-D_p|>\tau\}|}{|\mathcal{M}|}\cdot 100$
- Bumpiness: $\frac{1}{|\mathcal{M}|}\sum_{p\in\mathcal{M}}\min(0.05,\|\mathrm{H}_\Delta(p)\|_F)\cdot 100$

- ❖ DDFFNet is more accurate by a large margin when compared to classical variational DFF method.
- ❖ Faster on a GPU (orders of magnitude)

VDFF: Variational Depth from Focus Reconstruction, TIP, 2015

PSPNet: Pyramid Scene Parsing Network, CVPR, 2017

Deep Depth from Defocus

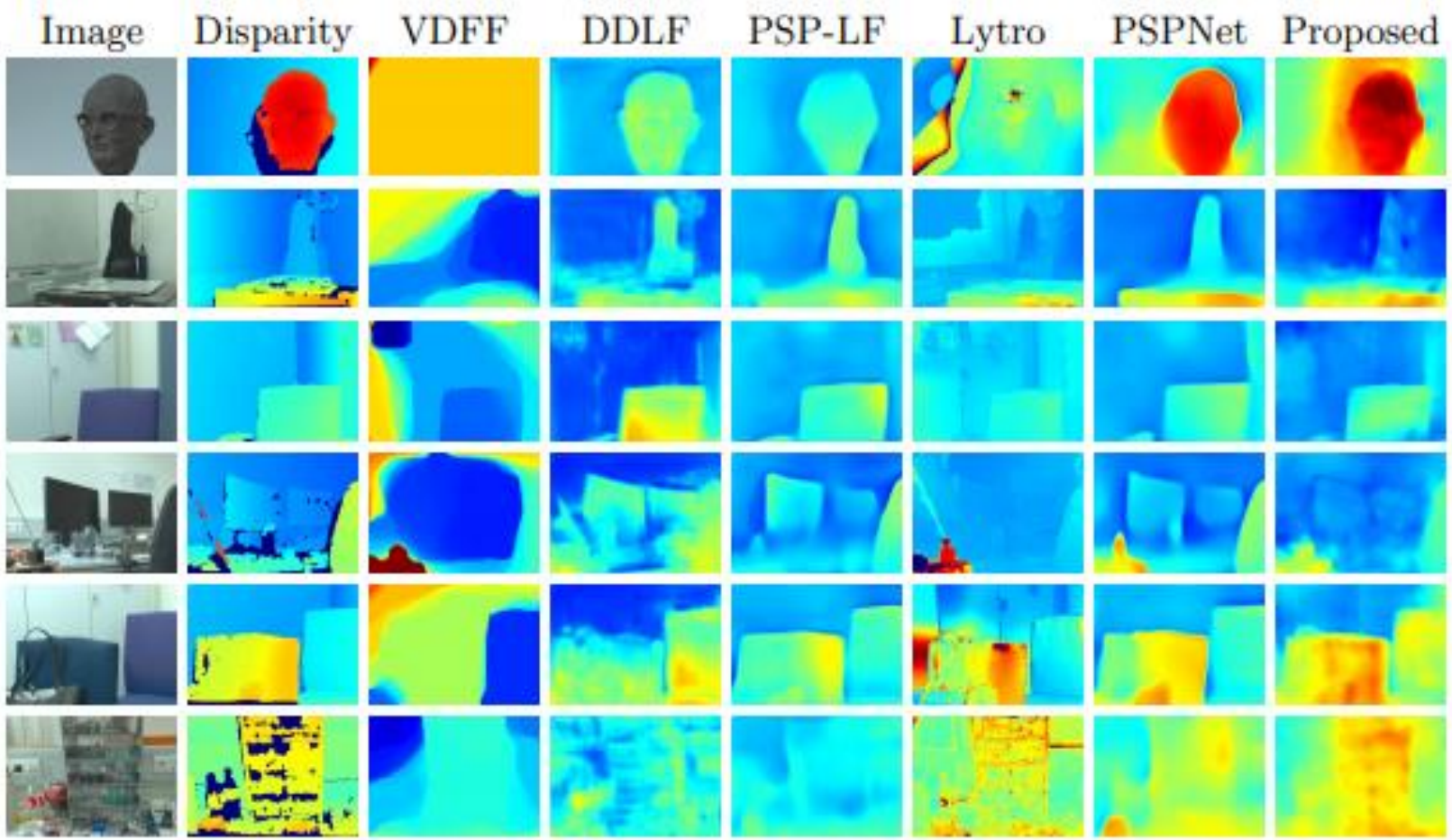


Fig. 6: **Qualitative results of the DDFFNet versus state-of-the-art methods.** Results are normalized by the maximum disparity. Warmer colors represent closer distances. Best viewed in color.

Deep Depth from Defocus

- What is the network learning?
  - Giving the stack of image to the network at the same time helps the network identify which pixel is the sharpest
  - The regression layer then regresses the depth from sharpness (focus)

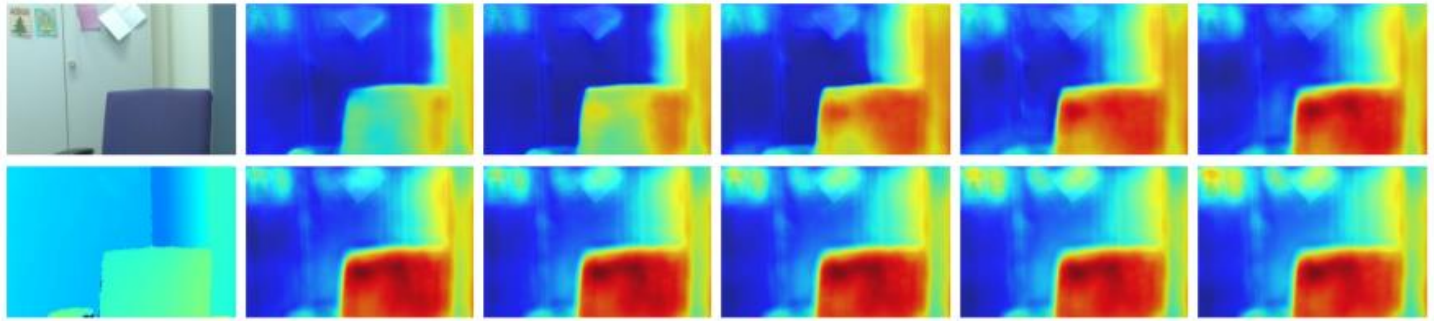


Fig. 2: **Activation heat maps** for the refocused images in a focal stack. First column: *top:* center sub-aperture image, *bottom:* groundtruth disparity map. The rest of the columns show from left to right, top to bottom, how the activations on the focal stack images evolve. Note how the activations slowly shift to the closest object (the chair) as we advance in the focal stack.

- Code and dataset are publicly available: https://github.com/soyers/ddff-pytorch
- More details can be found here: https://hazirbas.com/projects/ddff/