Graphical Models
Discrete Inference and Learning

MVA

2022 – 2023

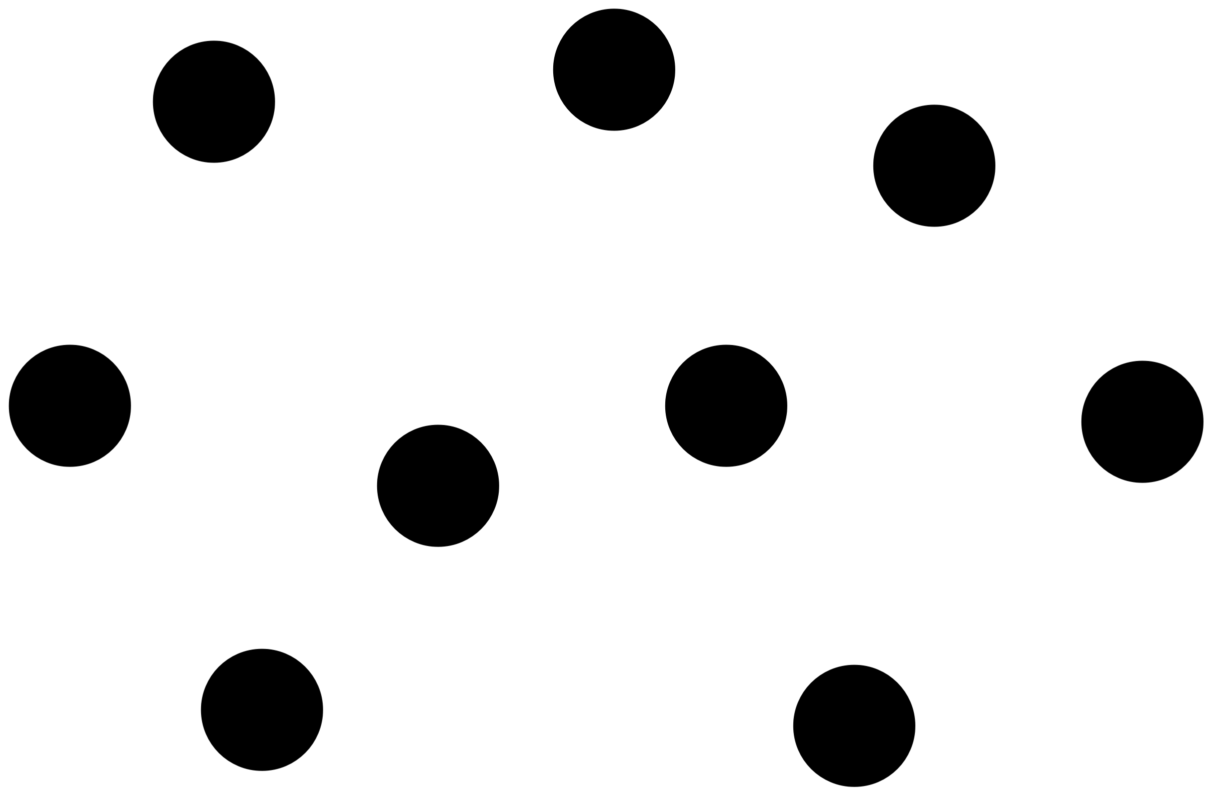
http://thoth.inrialpes.fr/~alahari/disinflearn

Recap

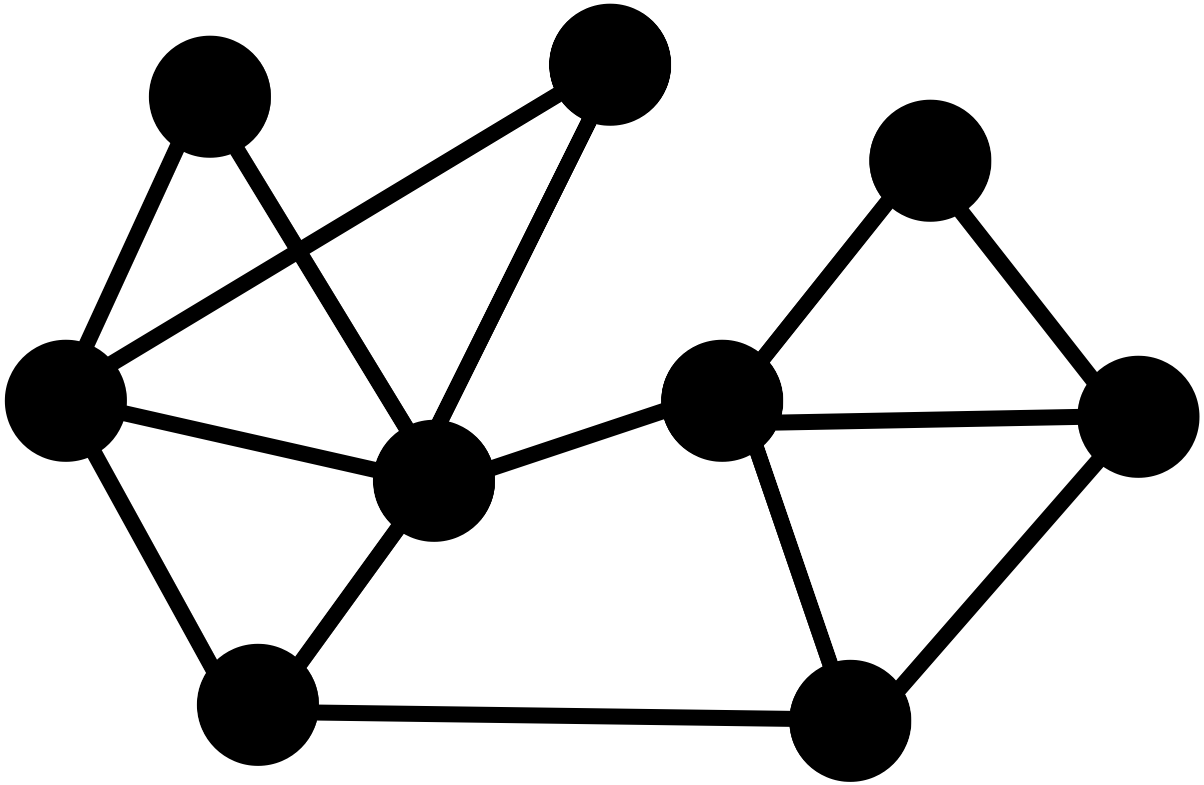
Why Graphs?

Graphs are a general language for describing and analyzing entities with relations/interactions

Slide courtesy: http://cs224w.Stanford.edu



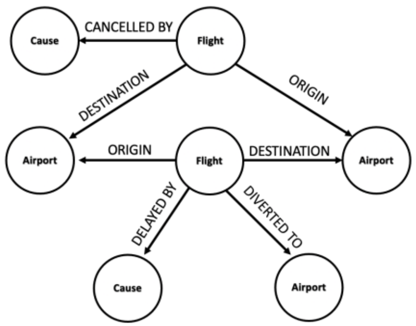
Slide courtesy: http://cs224w.Stanford.edu



Graph

Slide courtesy: http://cs224w.Stanford.edu

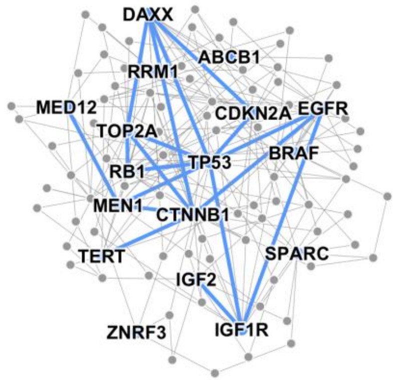
Many Types of Data are Graphs (1)



Event Graphs

Image credit: SalientNetworks

Computer Networks



Disease Pathways

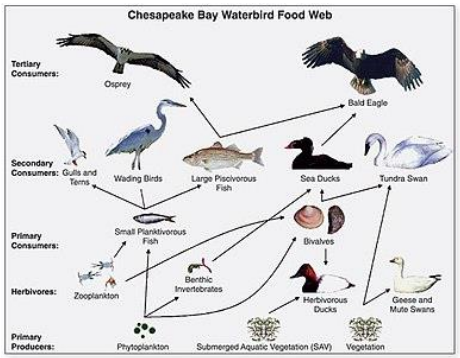


Image credit: Wikipedia

Food Webs

Image credit: Pinterest

Particle Networks

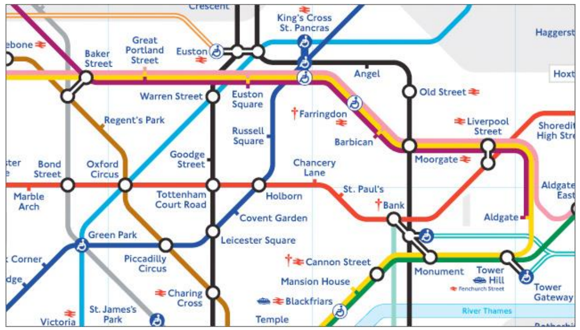


Image credit: visitlondon.com

Underground Networks

Many Types of Data are Graphs (2)

Image credit: Medium

Social Networks

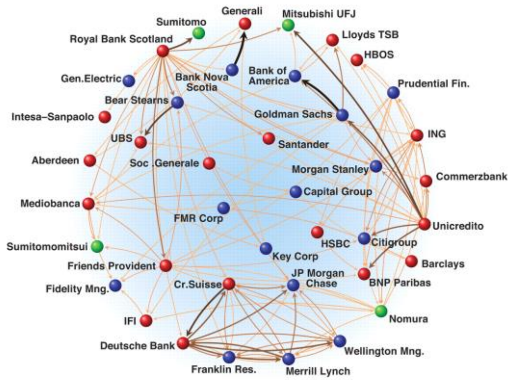
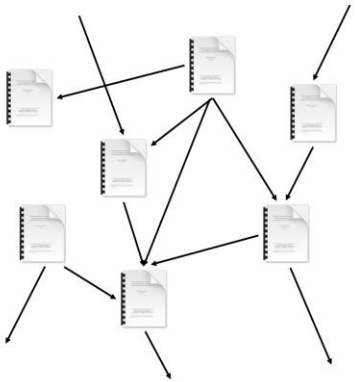


Image credit: Science

Economic Networks

Image credit: Lumen Learning

Communication Networks



Citation Networks

Image credit: Missoula Current News

Internet

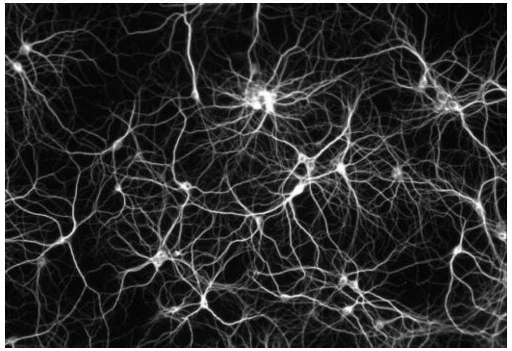


Image credit: The Conversation

Networks of Neurons

Slide courtesy: http://cs224w.Stanford.edu

Many Types of Data are Graphs (3)

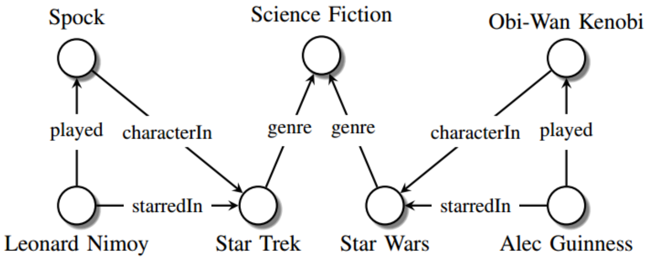


Image credit: Maximilian Nickel et al

Knowledge Graphs

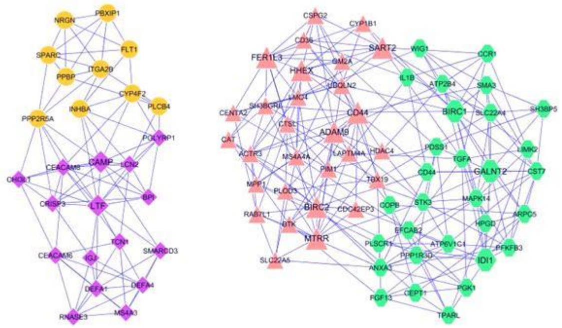


Image credit: ese.wustl.edu

Regulatory Networks

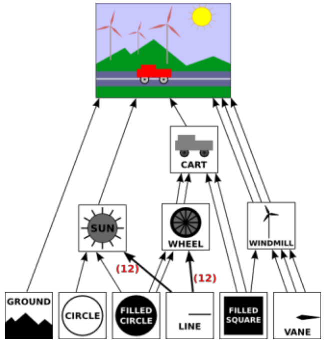


Image credit: math.hws.edu

Scene Graphs

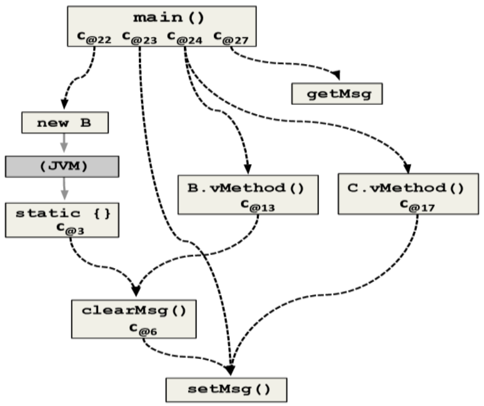


Image credit: ResearchGate

Code Graphs

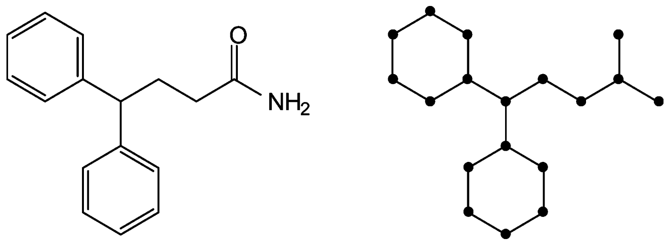


Image credit: MDPI

Molecules

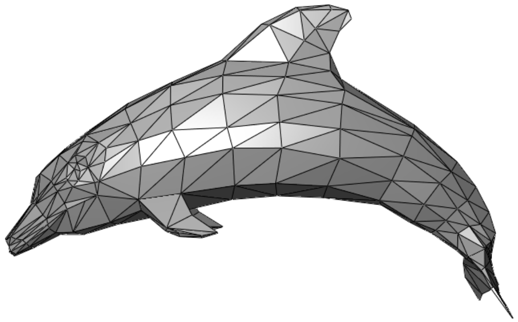


Image credit: Wikipedia

3D Shapes

Slide courtesy: http://cs224w.Stanford.edu

Graphs and Relational Data

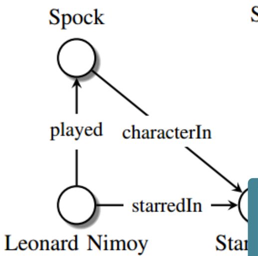
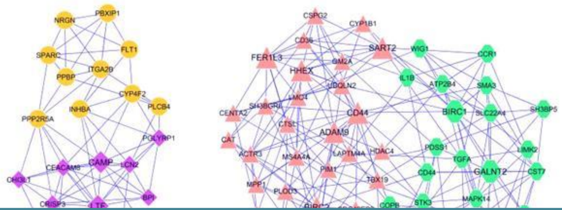
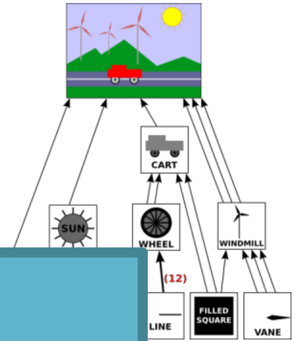


Image cred

Know



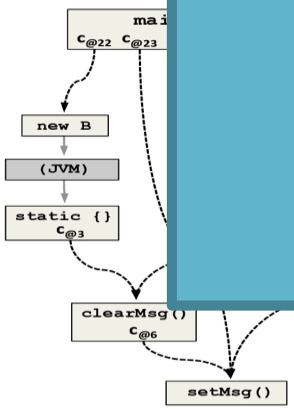


ath.hws.edu

Graphs

Main question:

How do we take advantage of relational structure for better prediction?



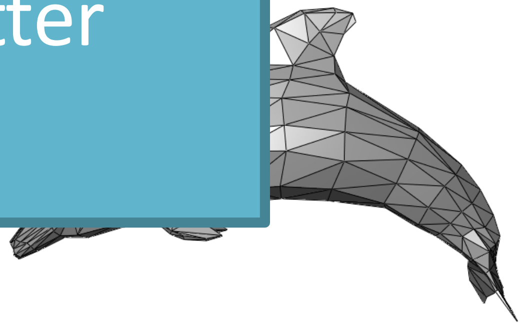


Image credit: ResearchGate

Code Graphs

Image credit: MDPI

Molecules

Image credit: Wikipedia

3D Shapes

Slide courtesy: http://cs224w.Stanford.edu

Graphs: Machine Learning

Complex domains have a rich relational structure, which can be represented as a **relational graph**

By explicitly modeling relationships we achieve better performance!

What have we seen?

- Inference
  - Belief propagation
  - Graph cuts (to be completed)
  - Variational inference
  - Simulation-based inference

Outline

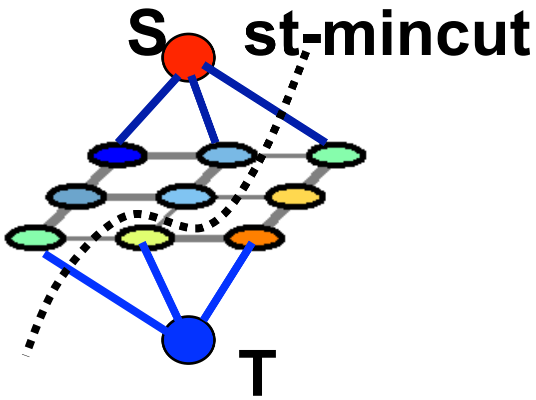
The st-mincut problem

Connection between st-mincut and energy minimization?

What problems can we solve using st-mincut?

st-mincut based Move algorithms

St-mincut and Energy Minimization



↔

Minimizing a Qudratic Pseudoboolean function E(x)

Functions of boolean variables

Pseudoboolean?

$$E: \{0,1\}^n \rightarrow R$$

$$E(y) = \sum_i c_i y_i + \sum_{i,j} c_{ij} y_i(1-y_j)$$

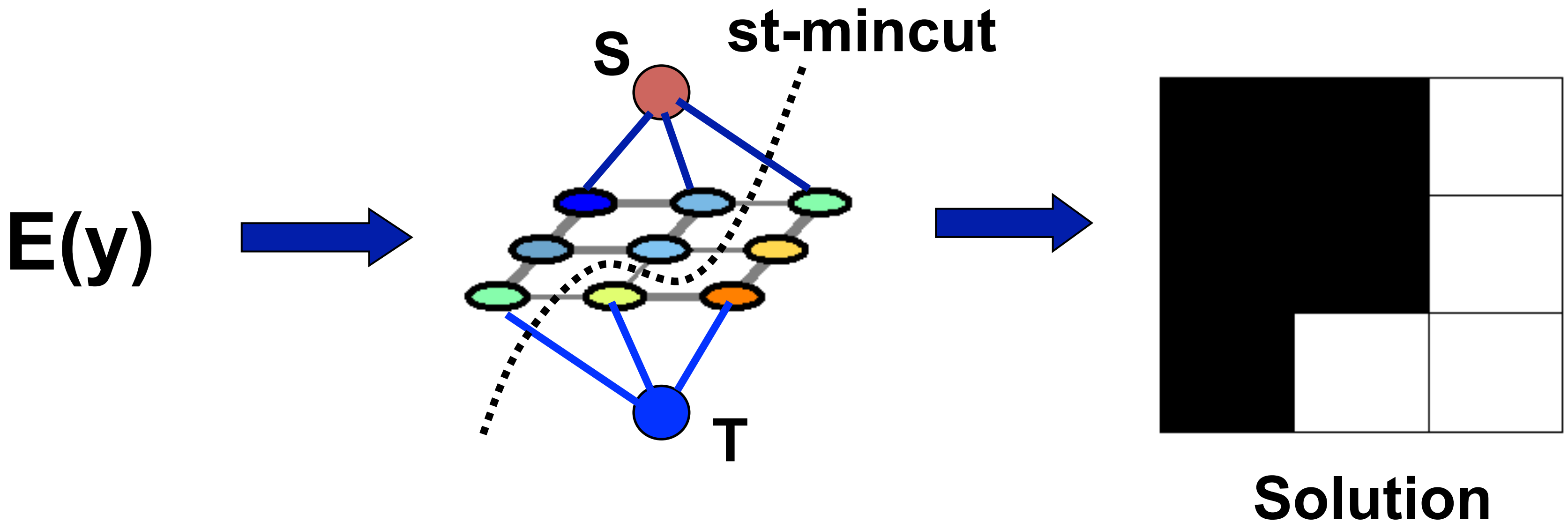
$c_{ij} \geq 0$

Polynomial time st-mincut algorithms require non-negative edge weights

So how does this work?

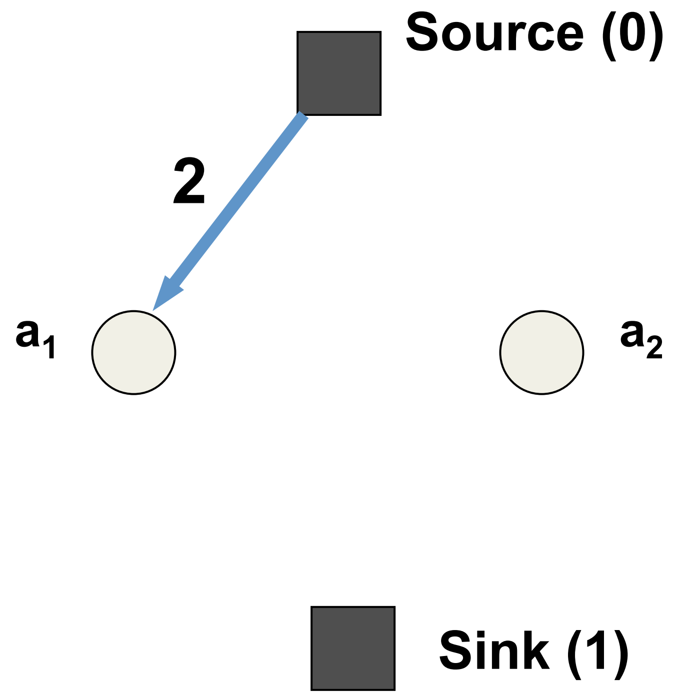
Construct a graph such that:

1. Any st-cut corresponds to an assignment of x
2. The cost of the cut is equal to the energy of x : E(x)



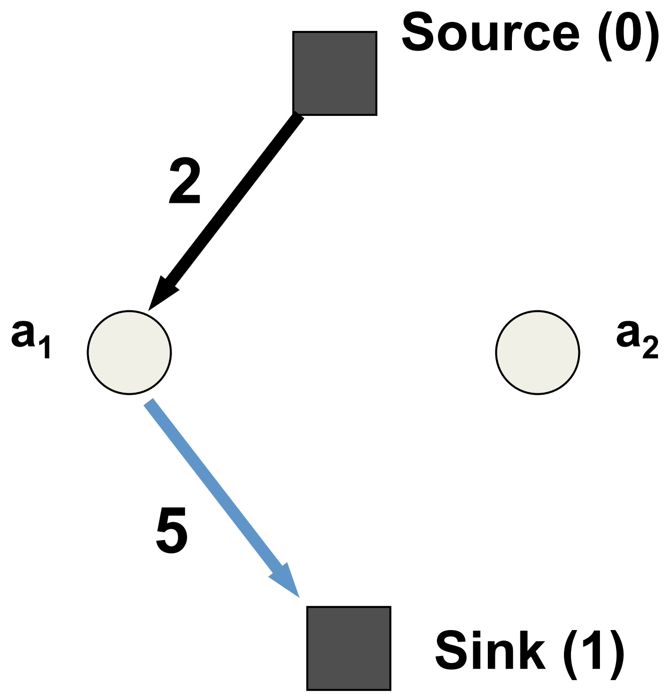
Graph Construction

$E(a_1,a_2) = 2a_1$



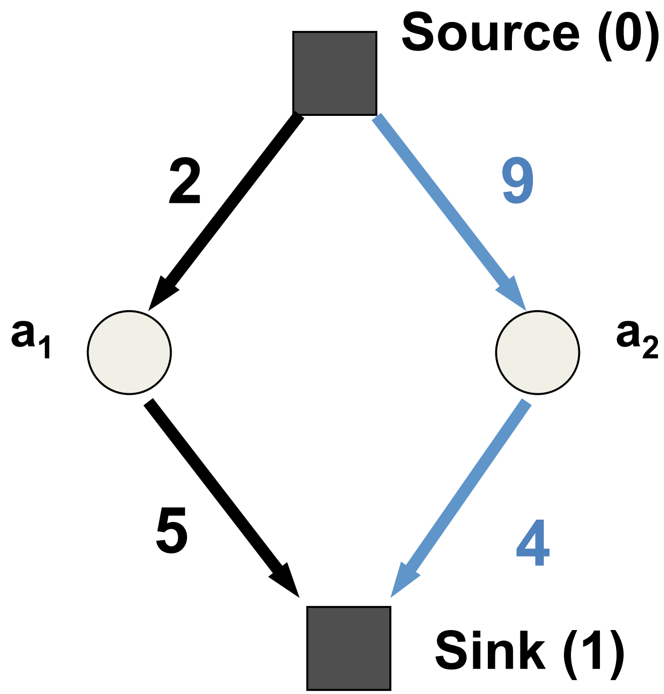
Graph Construction

$E(a_1,a_2) = 2a_1 + 5\bar{a}_1$



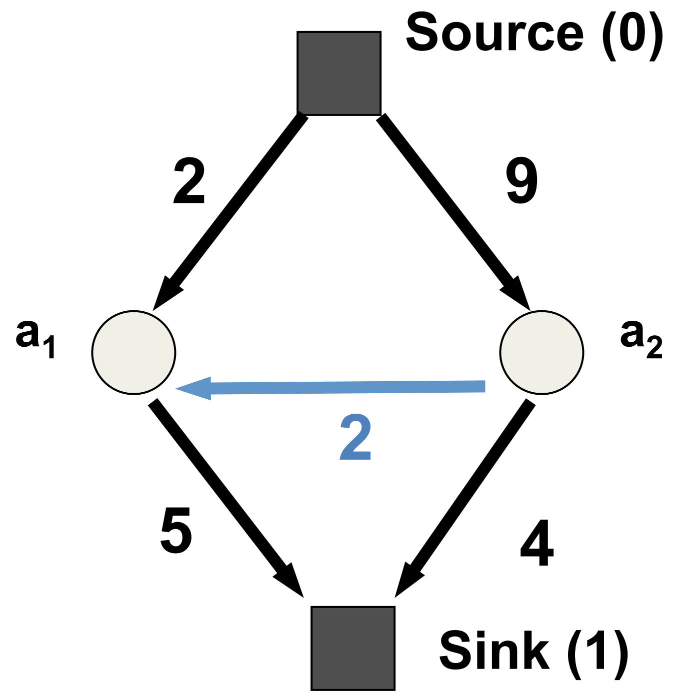
Graph Construction

$E(a_1,a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2$



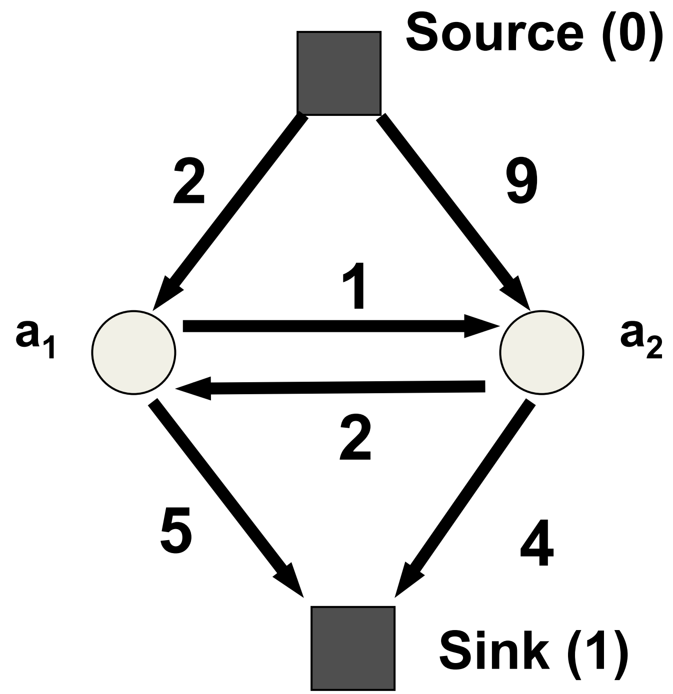
Graph Construction

$E(a_1,a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2$



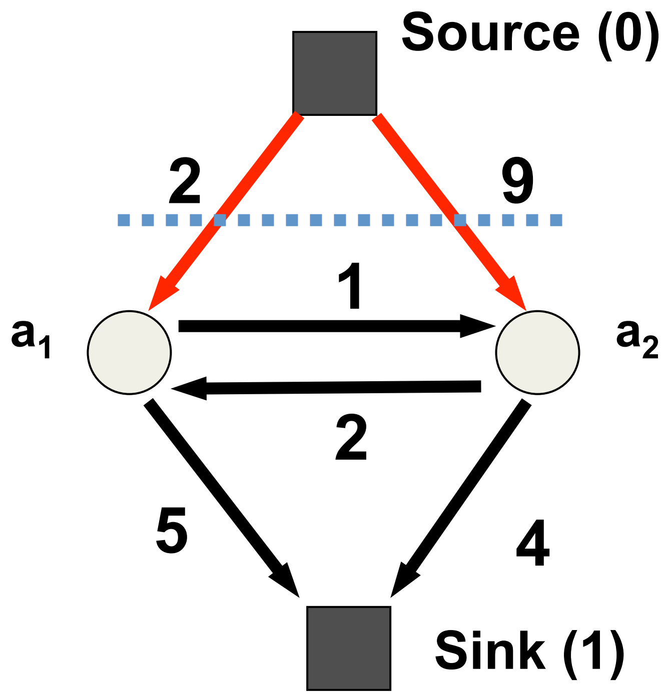
Graph Construction

$$E(a_1,a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2 + \bar{a}_1 a_2$$



Graph Construction

$$E(a_1,a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2 + \bar{a}_1 a_2$$



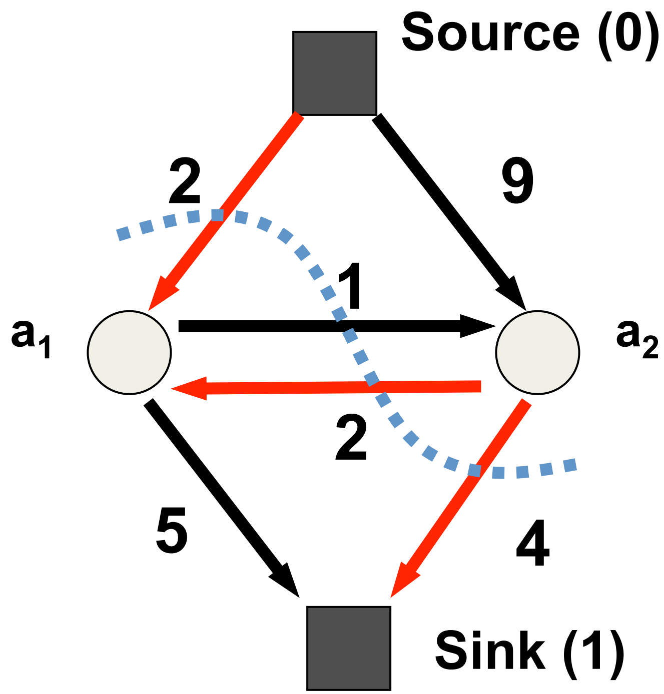
Cost of cut = 11

$a_1 = 1 \quad a_2 = 1$

E (1,1) = 11

Graph Construction

$E(a_1,a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2 + \bar{a}_1 a_2$



st-mincut cost = 8

$a_1 = 1 \quad a_2 = 0$

$E(1,0) = 8$

Energy Function Reparameterization

Two functions E_1 and E_2 are reparameterizations if

$$E_1(\mathbf{x}) = E_2(\mathbf{x}) \text{ for all } \mathbf{x}$$

For instance:

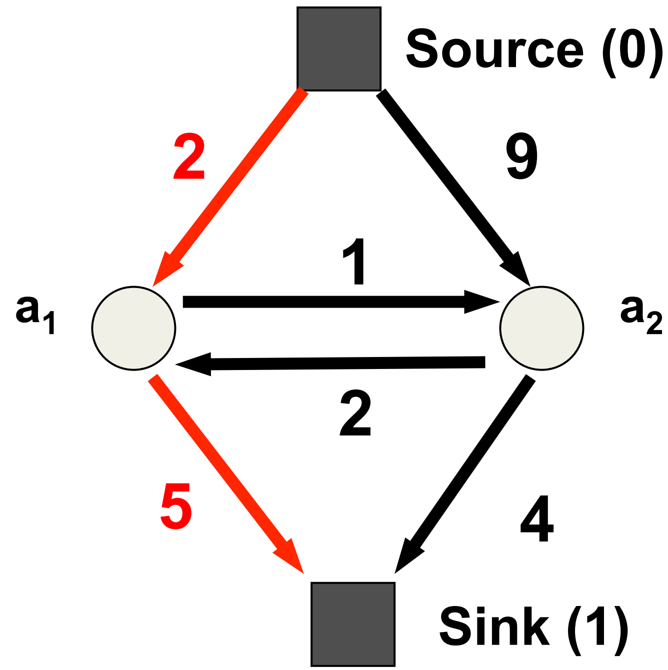
$\mathbf{E_1(a_1) = 1 + 2a_1 + 3\bar{a}_1}$

$\mathbf{E_2(a_1) = 3 + \bar{a}_1}$

a_1	$\bar{a}_1$	$1 + 2a_1 + 3\bar{a}_1$	$3 + \bar{a}_1$
0	1	4	4
1	0	3	3

Flow and Reparametrization

$E(a_1,a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2 + \bar{a}_1 a_2$



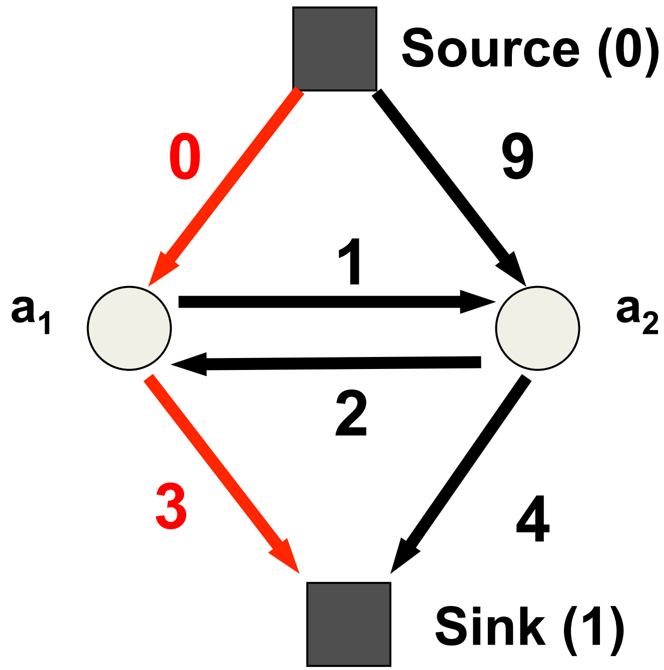
$2a_1 + 5\bar{a}_1$

$= 2(a_1+\bar{a}_1) + 3\bar{a}_1$

$= 2 + 3\bar{a}_1$

Flow and Reparametrization

$E(a_1,a_2) = 2 + 3\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2 + \bar{a}_1a_2$



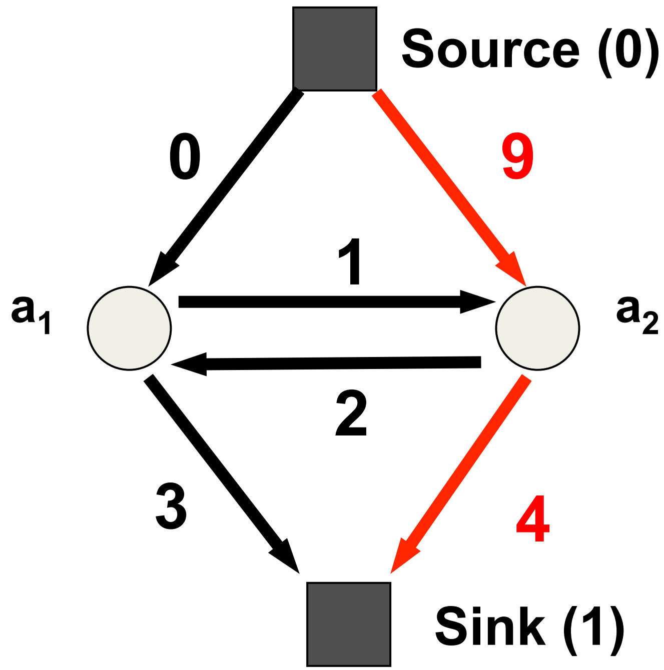
$2a_1 + 5\bar{a}_1$

$= 2(a_1+\bar{a}_1) + 3\bar{a}_1$

$= 2 + 3\bar{a}_1$

Flow and Reparametrization

$E(a_1,a_2) = 2 + 3\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2 + \bar{a}_1 a_2$



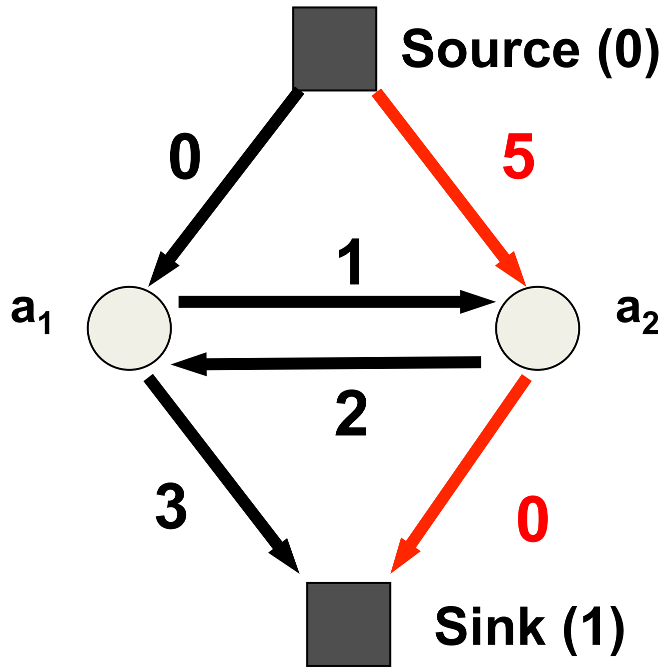
$9a_2 + 4\bar{a}_2$

$= 4(a_2+\bar{a}_2) + 5\bar{a}_2$

$= 4 + 5\bar{a}_2$

Flow and Reparametrization

$E(a_1,a_2) = 2 + 3\bar{a}_1 + 5a_2 + 4 + 2a_1\bar{a}_2 + \bar{a}_1 a_2$



$9a_2 + 4\bar{a}_2$

$= 4(a_2+\bar{a}_2) + 5\bar{a}_2$

$= 4 + 5\bar{a}_2$

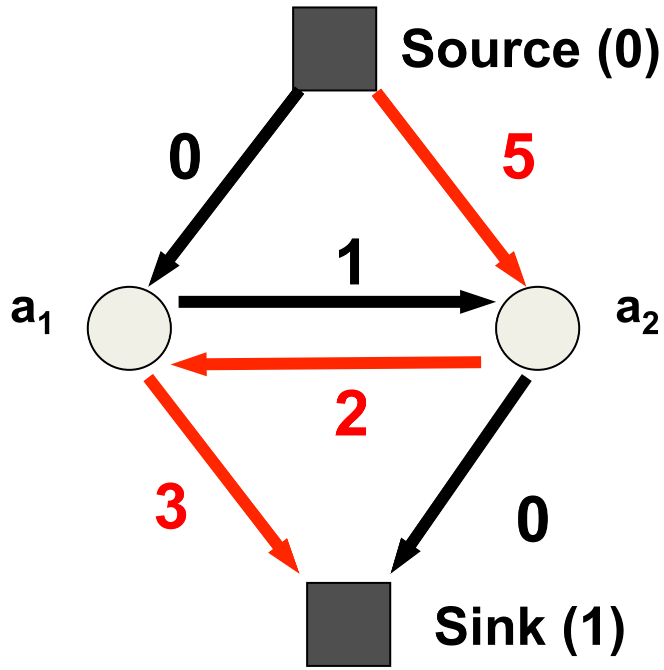
Flow and Reparametrization

$E(a_1,a_2) = 6 + 3\bar{a}_1 + 5a_2 + 2a_1\bar{a}_2 + \bar{a}_1a_2$

$3\bar{a}_1 + 5a_2 + 2a_1\bar{a}_2$

$= 2(\bar{a}_1+a_2+a_1\bar{a}_2) + \bar{a}_1+3a_2$

$= 2(1+\bar{a}_1a_2) + \bar{a}_1+3a_2$



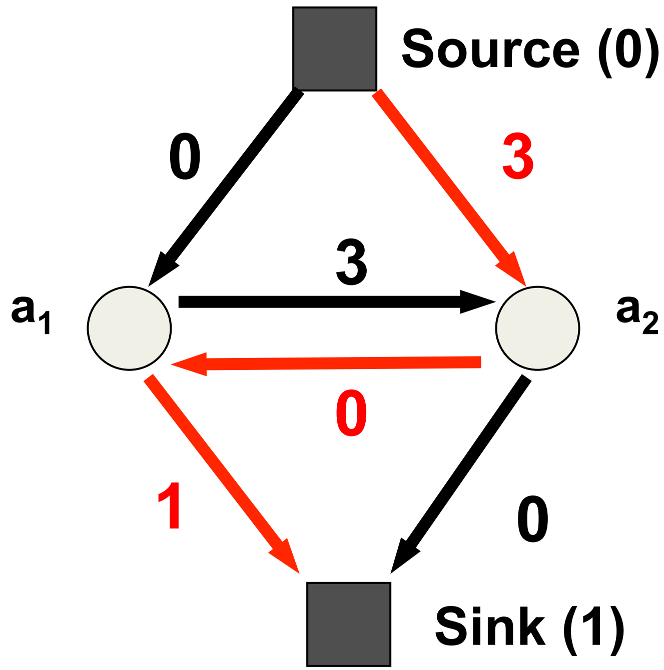
$F1 = \bar{a}_1+a_2+a_1\bar{a}_2$

$F2 = 1+\bar{a}_1a_2$

a_1	a_2	F1	F2
0	0	1	1
0	1	2	2
1	0	1	1
1	1	1	1

Flow and Reparametrization

$E(a_1,a_2) = 8 + \bar{a}_1 + 3a_2 + 3\bar{a}_1 a_2$



$3\bar{a}_1 + 5a_2 + 2a_1\bar{a}_2$

$= 2(\bar{a}_1 + a_2 + a_1\bar{a}_2) + \bar{a}_1 + 3a_2$

$= 2(1 + \bar{a}_1 a_2) + \bar{a}_1 + 3a_2$

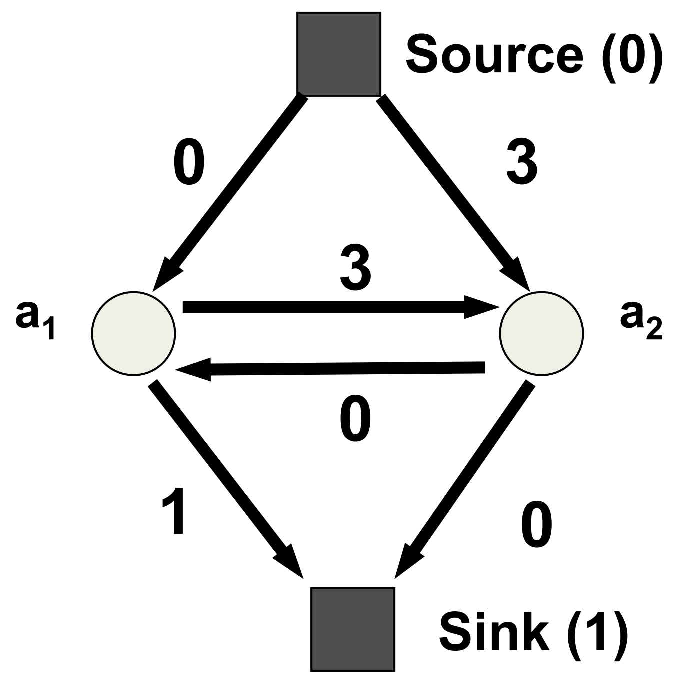
$F1 = \bar{a}_1 + a_2 + a_1\bar{a}_2$

$F2 = 1 + \bar{a}_1 a_2$

a_1	a_2	F1	F2
0	0	1	1
0	1	2	2
1	0	1	1
1	1	1	1

Flow and Reparametrization

$E(a_1,a_2) = 8 + \bar{a}_1 + 3a_2 + 3\bar{a}_1 a_2$



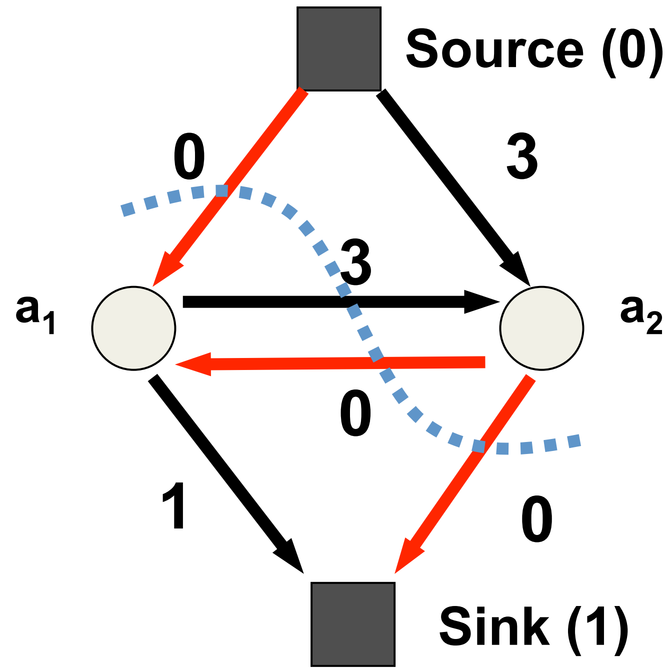
No more augmenting paths possible

Flow and Reparametrization

$E(a_1,a_2) = 8 + \bar{a}_1 + 3a_2 + 3\bar{a}_1 a_2$ → Residual Graph (positive coefficients)

Total Flow

bound on the optimal solution



st-mincut cost = 8

$a_1 = 1 \quad a_2 = 0$

$E(1,0) = 8$

Inference of the optimal solution becomes trivial because the bound is tight

Example: Image Segmentation

$$E(y) = \sum_i c_i y_i + \sum_{i,j} c_{ij} y_i(1-y_j)$$

$E: \{0,1\}^n \rightarrow R$
$0 \rightarrow$ fg
$1 \rightarrow$ bg

Global Minimum (y*)

$$y^* = \arg\min_y E(y)$$

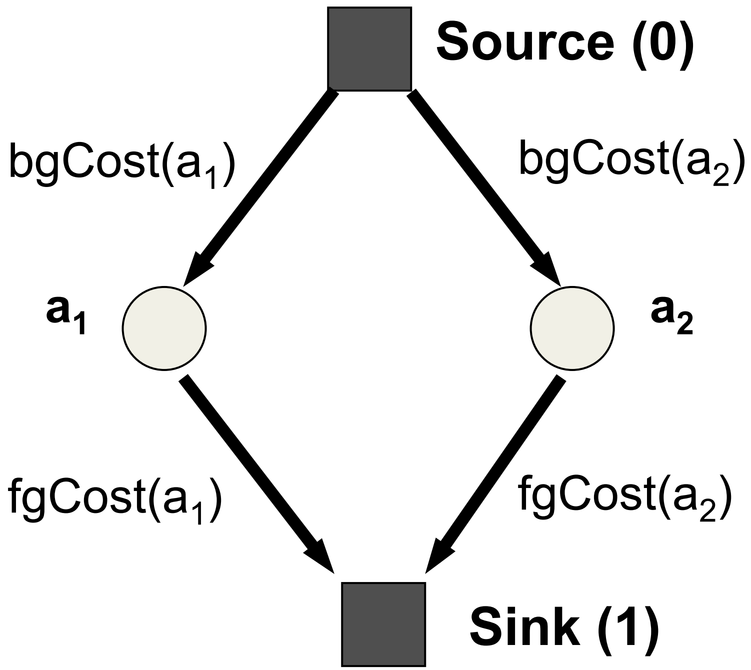
How to minimize E(x)?

How does the code look like?

```
Graph *g;

For all pixels p

        /* Add a node to the graph */
        nodeID(p) = g->add_node();

        /* Set cost of terminal edges */
        set_weights(nodeID(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q
        add_weights(nodeID(p), nodeID(q),  cost);
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```

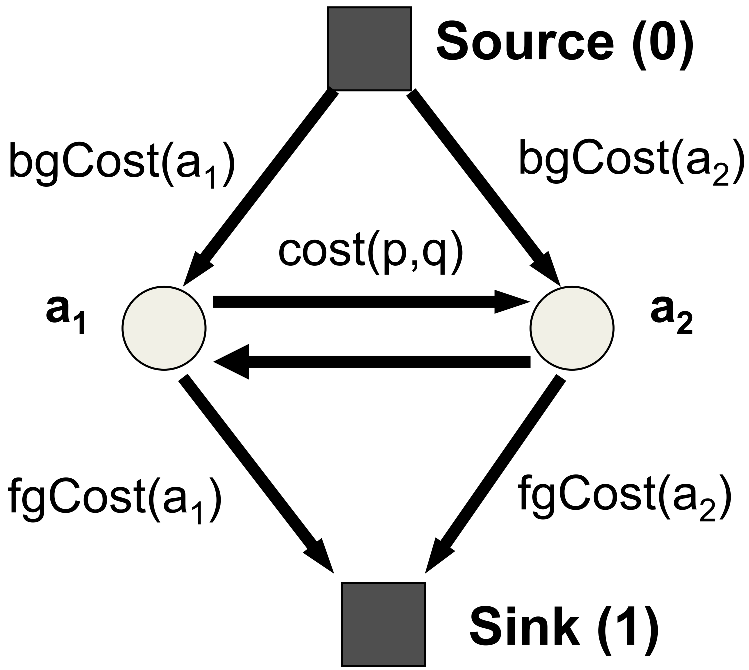
How does the code look like?

```
Graph *g;

For all pixels p

        /* Add a node to the graph */
        nodeID(p) = g->add_node();

        /* Set cost of terminal edges */
        set_weights(nodeID(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q
        add_weights(nodeID(p), nodeID(q),  cost);
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```



How does the code look like?

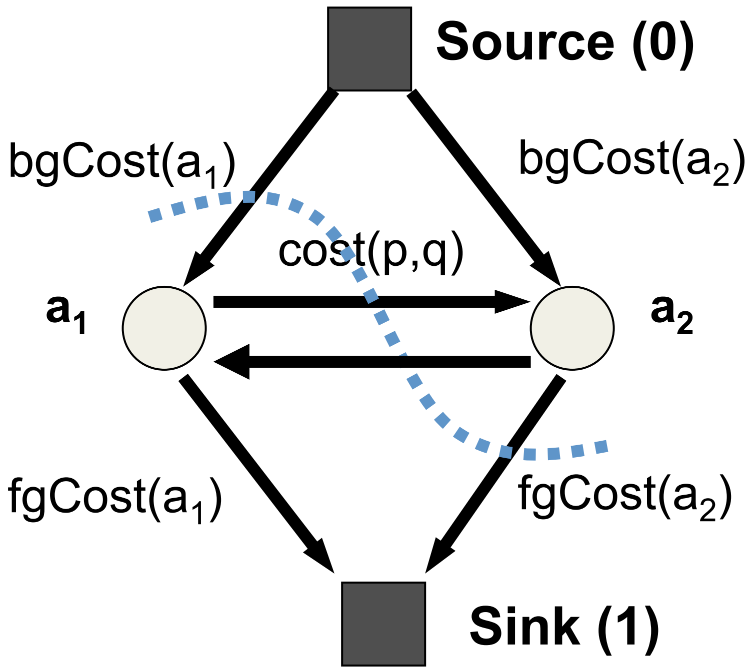
```
Graph *g;

For all pixels p

        /* Add a node to the graph */
        nodeID(p) = g->add_node();

        /* Set cost of terminal edges */
        set_weights(nodeID(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q
        add_weights(nodeID(p), nodeID(q),  cost(p,q));
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```



How does the code look like?

```
Graph *g;

For all pixels p

        /* Add a node to the graph */
        nodeID(p) = g->add_node();

        /* Set cost of terminal edges */
        set_weights(nodeID(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q
        add_weights(nodeID(p), nodeID(q),  cost(p,q));
end
```

```
g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```



$a_1 = bg \;\; a_2 = fg$

Outline

The st-mincut problem

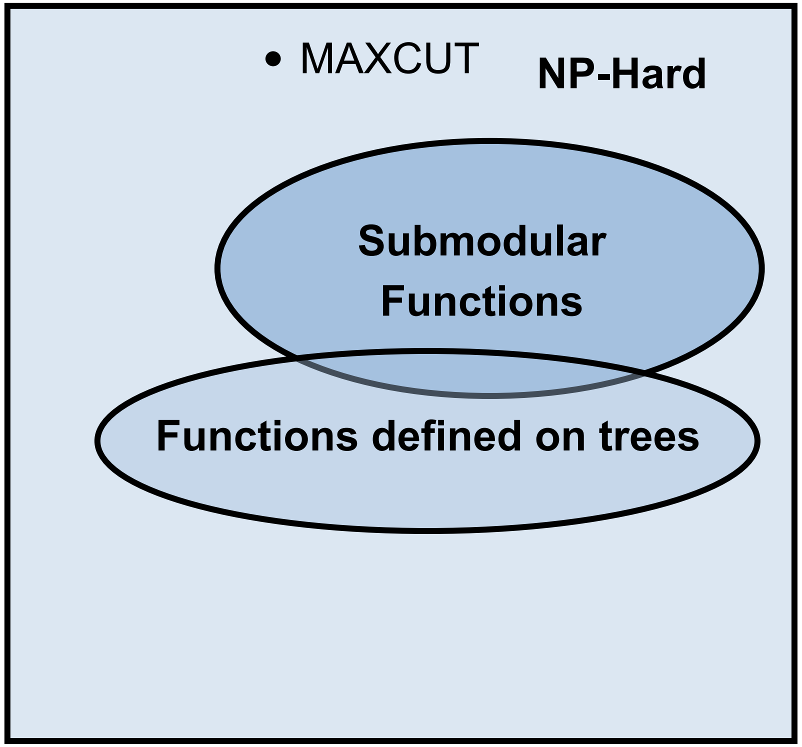
Connection between st-mincut and energy minimization?

What problems can we solve using st-mincut?

st-mincut based Move algorithms

Minimizing Energy Functions

- **General Energy Functions**
  - NP-hard to minimize
  - Only approximate minimization possible
- **Easy energy functions**
  - Solvable in polynomial time
  - Submodular ~ $O(n^6)$



Space of Function Minimization Problems

Minimizing Submodular Functions

- **Minimizing general submodular functions**
  - $O(n^5 Q + n^6)$ where Q is function evaluation time **[Orlin, IPCO 2007]**
- **Symmetric submodular functions**
  - $E(\mathbf{y}) = E(\mathbf{1} - \mathbf{y})$
  - $O(n^3)$ **[Queyranne 1998]**
- **Quadratic pseudoboolean**
  - Can be transformed to st-mincut
  - One node per variable ($O(n^3)$ complexity)
  - Very low empirical running time

Submodular Pseudoboolean Functions

Function defined over boolean vectors $\mathbf{y} = \{y_1, y_2, \ldots. y_n\}$

Definition

- All functions for one boolean variable ($f: \{0,1\} \rightarrow \mathbb{R}$) are submodular

- A function of two boolean variables ($f: \{0,1\}^2 \rightarrow \mathbb{R}$) is submodular if

$$f(0,1) + f(1,0) \geq f(0,0) + f(1,1)$$

- A general pseudoboolean function $f : 2^n \rightarrow \mathbb{R}$ is submodular if all its projections f^p are submodular i.e.

$$f^p(0,1) + f^p(1,0) \geq f^p(0,0) + f^p(1,1)$$

Quadratic Submodular Pseudoboolean Functions

$$E(y) = \sum_i \theta_i (y_i) + \sum_{i,j} \theta_{ij} (y_i, y_j)$$

For all ij $\quad \theta_{ij}(0,1) + \theta_{ij}(1,0) \geq \theta_{ij}(0,0) + \theta_{ij}(1,1)$

↕ **Equivalent (transformable)**

$$E(y) = \sum_i c_i y_i + \sum_{i,j} c_{ij} y_i(1-y_j)$$

$c_{ij} \geq 0$

i.e. all submodular QPBFs are st-mincut solvable

How are they equivalent?

$A = \theta_{ij}(0,0)$ $\quad$ $B = \theta_{ij}(0,1)$ $\quad$ $C = \theta_{ij}(1,0)$ $\quad$ $D = \theta_{ij}(1,1)$

$$\begin{array}{c|cc} y_i \backslash y_j & 0 & 1 \\ \hline 0 & A & B \\ 1 & C & D \end{array} = A + \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & C\text{-}A & C\text{-}A \end{array} + \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & D\text{-}C \\ 1 & 0 & D\text{-}C \end{array} + \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & B+C\text{-}A\text{-}D \\ 1 & 0 & 0 \end{array}$$

if y_i=1 add C-A $\quad$ **if y_j = 1 add D-C**

$$\theta_{ij}(y_i,y_j) = \theta_{ij}(0,0)$$
$$+ (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, y_i + (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, y_j$$
$$+ (\theta_{ij}(1,0) + \theta_{ij}(0,1) - \theta_{ij}(0,0) - \theta_{ij}(1,1))\,(1-y_i)\, y_j$$

B+C-A-D $\geq$ 0 is true from the submodularity of θ_{ij}

How are they equivalent?

$A = \theta_{ij}(0,0)$	$B = \theta_{ij}(0,1)$	$C = \theta_{ij}(1,0)$	$D = \theta_{ij}(1,1)$

y_i \ y_j	0	1
0	A	B
1	C	D

= **A** +

	0	1
0	0	0
1	C-A	C-A

if y_i=1 add C-A

+

	0	1
0	0	D-C
1	0	D-C

if y_j = 1 add D-C

+

	0	1
0	0	B+C-A-D
1	0	0

$$\theta_{ij}(y_i,y_j) = \theta_{ij}(0,0)$$
$$+ (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, y_i + (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, y_j$$
$$+ (\theta_{ij}(1,0) + \theta_{ij}(0,1) - \theta_{ij}(0,0) - \theta_{ij}(1,1))\, (1-y_i)\, y_j$$

$B+C-A-D \geq 0$ is true from the submodularity of θ_{ij}

How are they equivalent?

$A = \theta_{ij}(0,0)$ $B = \theta_{ij}(0,1)$ $C = \theta_{ij}(1,0)$ $D = \theta_{ij}(1,1)$

y_i \ y_j	0	1
0	A	B
1	C	D

= A +

	0	1
0	0	0
1	C-A	C-A

if y_i=1 add C-A

+

	0	1
0	0	D-C
1	0	D-C

if y_j = 1 add D-C

+

	0	1
0	0	B+C-A-D
1	0	0

$$\theta_{ij}(y_i,y_j) = \theta_{ij}(0,0) + (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, y_i + (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, y_j + (\theta_{ij}(1,0) + \theta_{ij}(0,1) - \theta_{ij}(0,0) - \theta_{ij}(1,1))\,(1-y_i)\, y_j$$

$B+C-A-D \geq 0$ is true from the submodularity of θ_{ij}

How are they equivalent?

$A = \theta_{ij}(0,0)$ $B = \theta_{ij}(0,1)$ $C = \theta_{ij}(1,0)$ $D = \theta_{ij}(1,1)$

y_i \ y_j	0	1
0	A	B
1	C	D

= A +

	0	1
0	0	0
1	C-A	C-A

if y_i=1 add C-A

+

	0	1
0	0	D-C
1	0	D-C

if y_j = 1 add D-C

+

	0	1
0	0	B+C-A-D
1	0	0

$$\theta_{ij}(y_i,y_j) = \theta_{ij}(0,0) + (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, y_i + (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, y_j + (\theta_{ij}(1,0) + \theta_{ij}(0,1) - \theta_{ij}(0,0) - \theta_{ij}(1,1))\,(1-y_i)\, y_j$$

$B+C-A-D \geq 0$ is true from the submodularity of θ_{ij}

How are they equivalent?

$A = \theta_{ij}(0,0)$ $B = \theta_{ij}(0,1)$ $C = \theta_{ij}(1,0)$ $D = \theta_{ij}(1,1)$

y_i \ y_j	0	1
0	A	B
1	C	D

= A +

	0	1
0	0	0
1	C-A	C-A

+

	0	1
0	0	D-C
1	0	D-C

+

	0	1
0	0	B+C-A-D
1	0	0

if y_i=1 add C-A **if y_j = 1 add D-C**

$$\theta_{ij}(y_i,y_j) = \theta_{ij}(0,0)$$
$$+ (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, y_i + (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, y_j$$
$$+ (\theta_{ij}(1,0) + \theta_{ij}(0,1) - \theta_{ij}(0,0) - \theta_{ij}(1,1))\,(1-y_i)\, y_j$$

B+C-A-D ≥ 0 is true from the submodularity of θ_{ij}

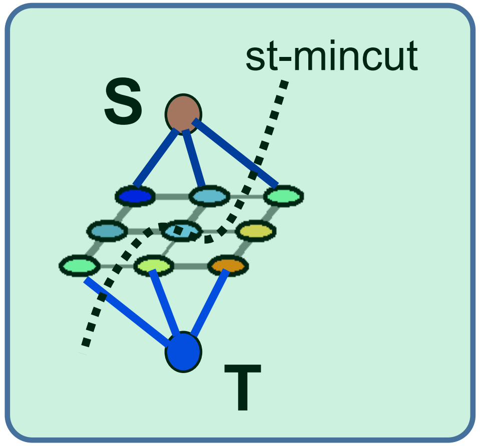
Quadratic Submodular Pseudoboolean Functions

y in $\{0,1\}^n$

$$E(y) = \sum_i \theta_i(y_i) + \sum_{i,j} \theta_{ij}(y_i, y_j)$$

For all ij $\quad \theta_{ij}(0,1) + \theta_{ij}(1,0) \geq \theta_{ij}(0,0) + \theta_{ij}(1,1)$

Equivalent (transformable)



Recap

- **Exact minimization of Submodular QBFs using graph cuts**

- **Obtaining partially optimal solutions of non-submodular QBFs using graph cuts**

Outline

The st-mincut problem

Connection between st-mincut and energy minimization?

What problems can we solve using st-mincut?

st-mincut based Move algorithms

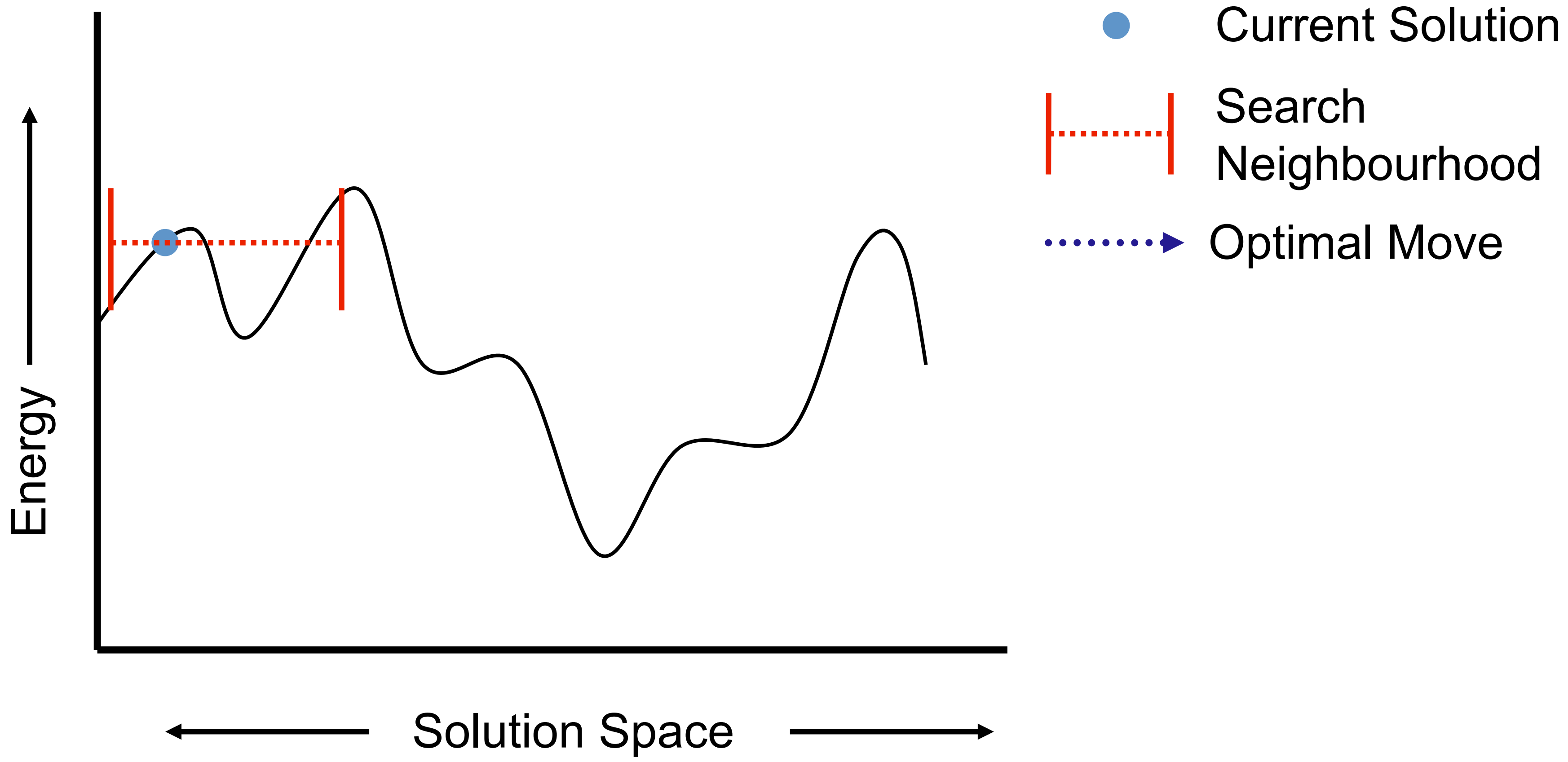
St-mincut based Move algorithms

$$E(\mathbf{y}) = \sum_{i} \theta_i (y_i) + \sum_{i,j} \theta_{ij} (y_i, y_j)$$

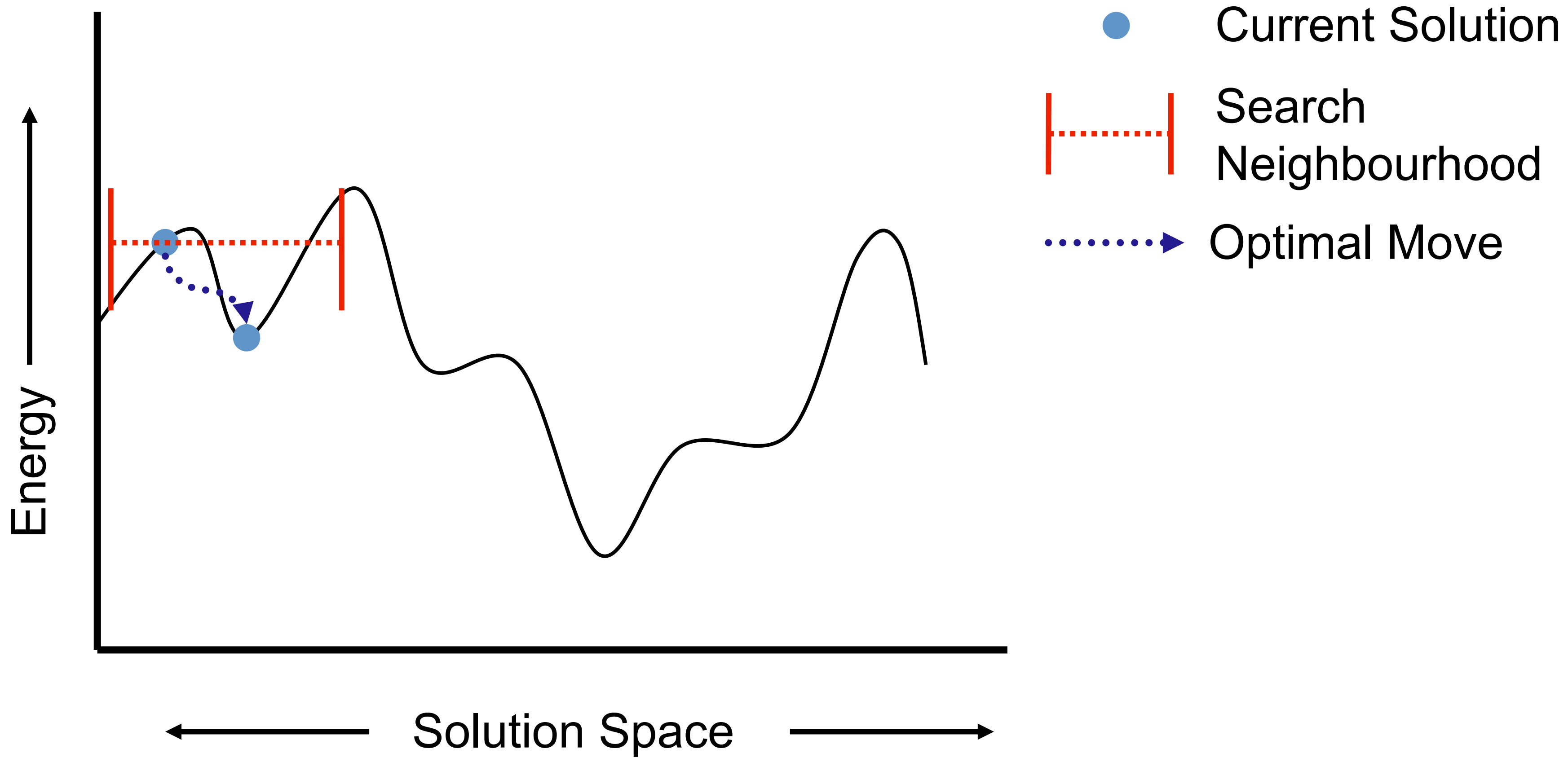
$y \in$ Labels $L = \{l_1, l_2, \dots, l_k\}$

- Commonly used for solving **non-submodular** multi-label problems
- Extremely efficient and produce good solutions
- Not Exact: Produce local optima

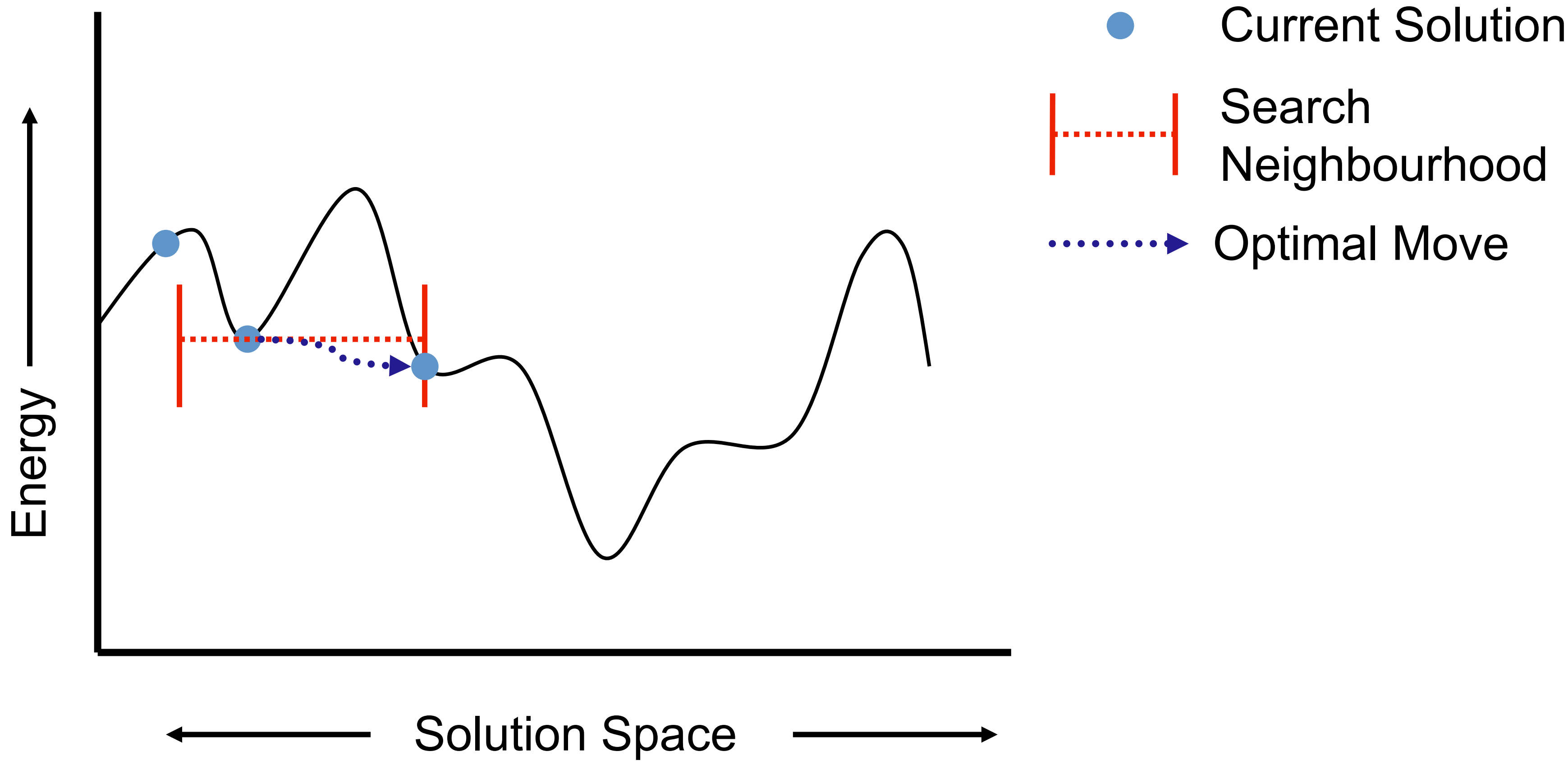
Move Making Algorithms



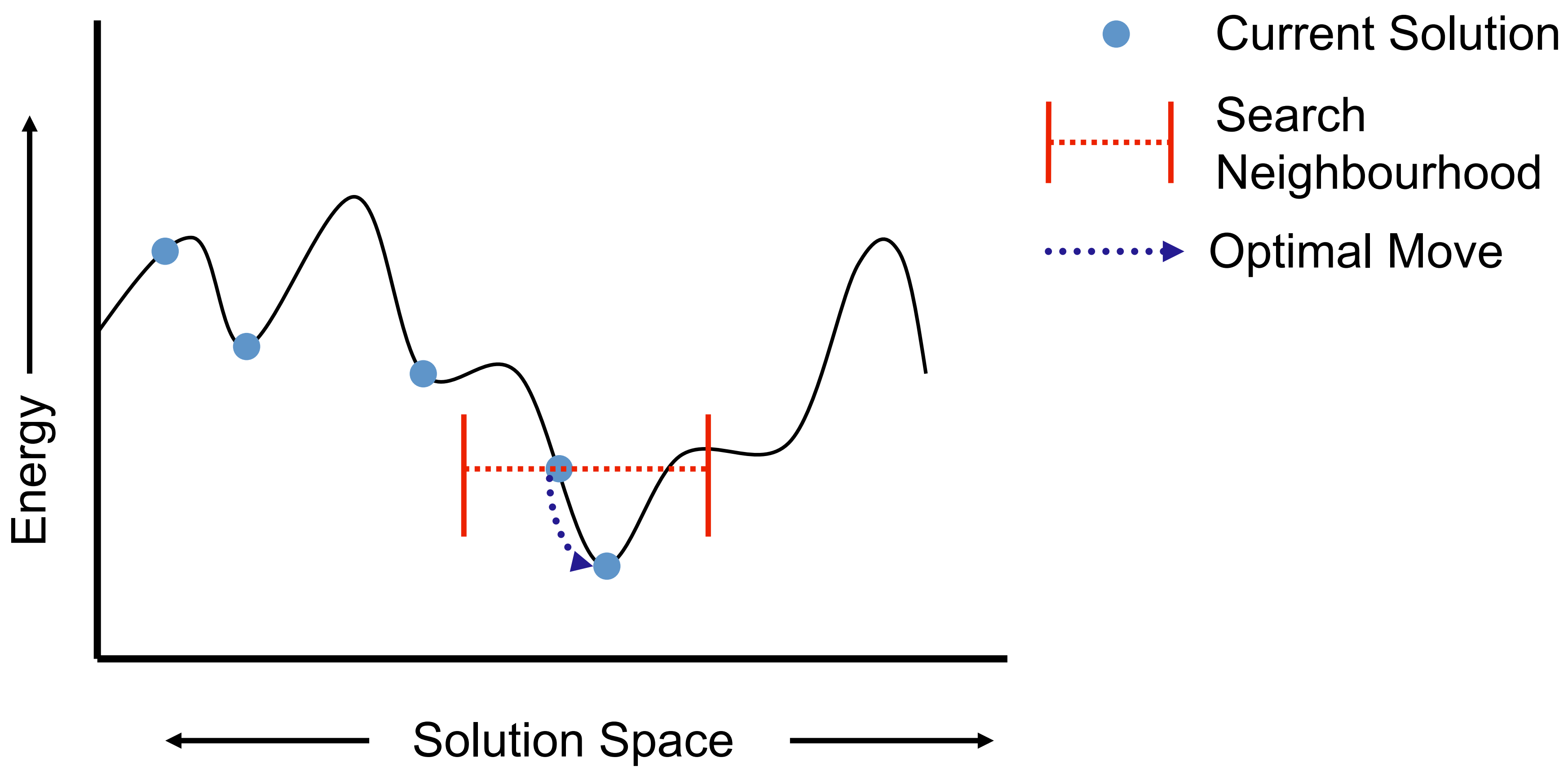
Move Making Algorithms



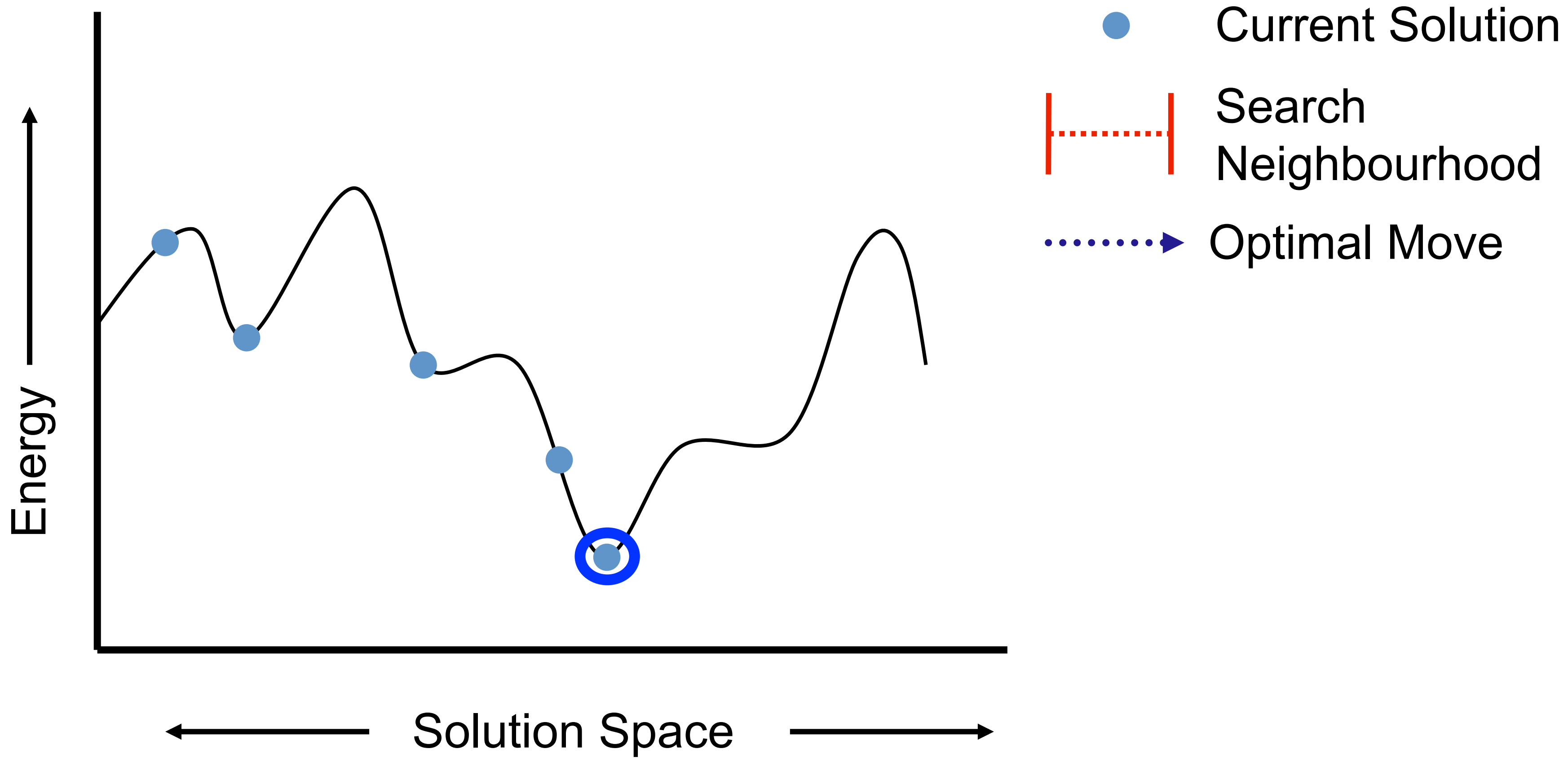
Move Making Algorithms



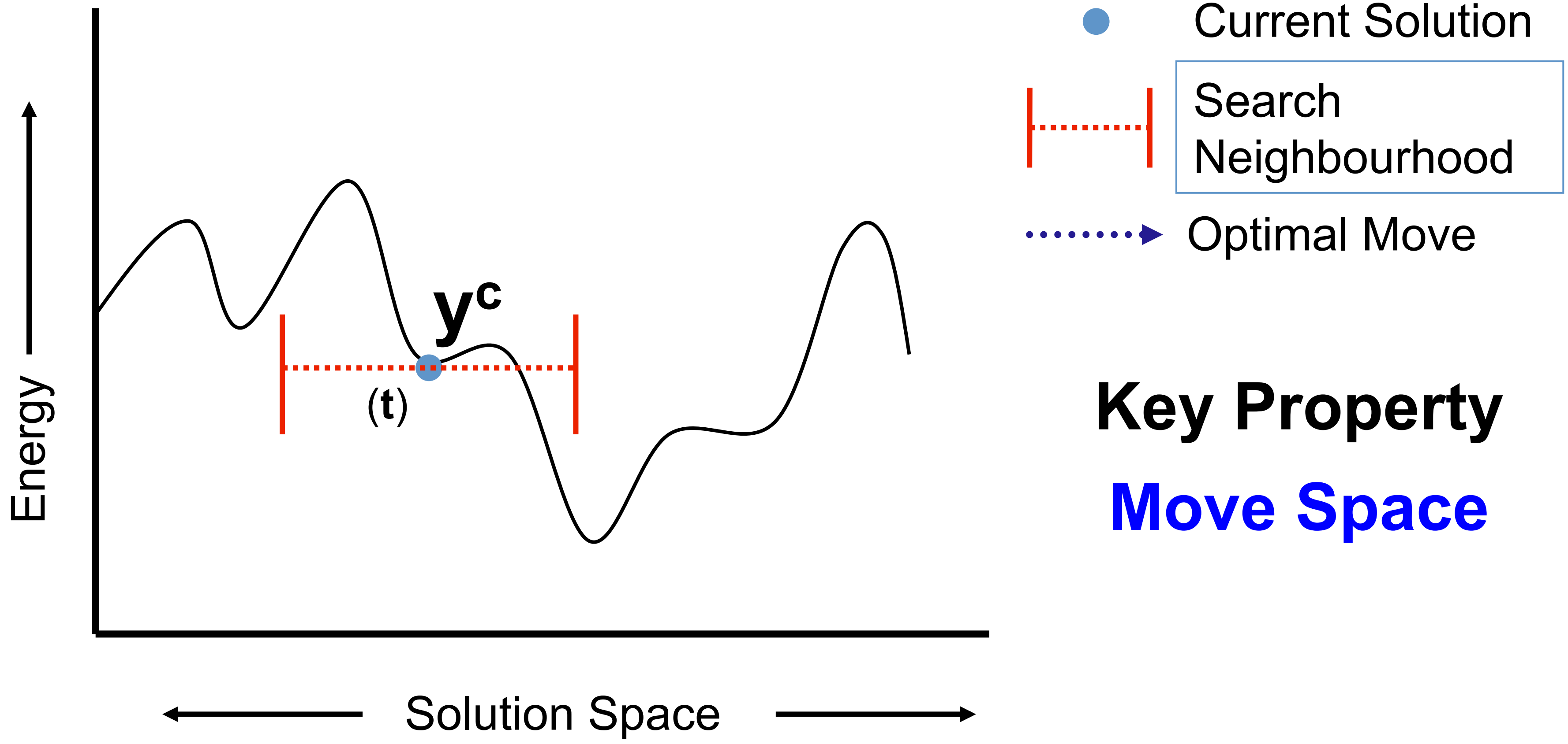
Move Making Algorithms



Move Making Algorithms



Computing the Optimal Move



Key Property

Move Space

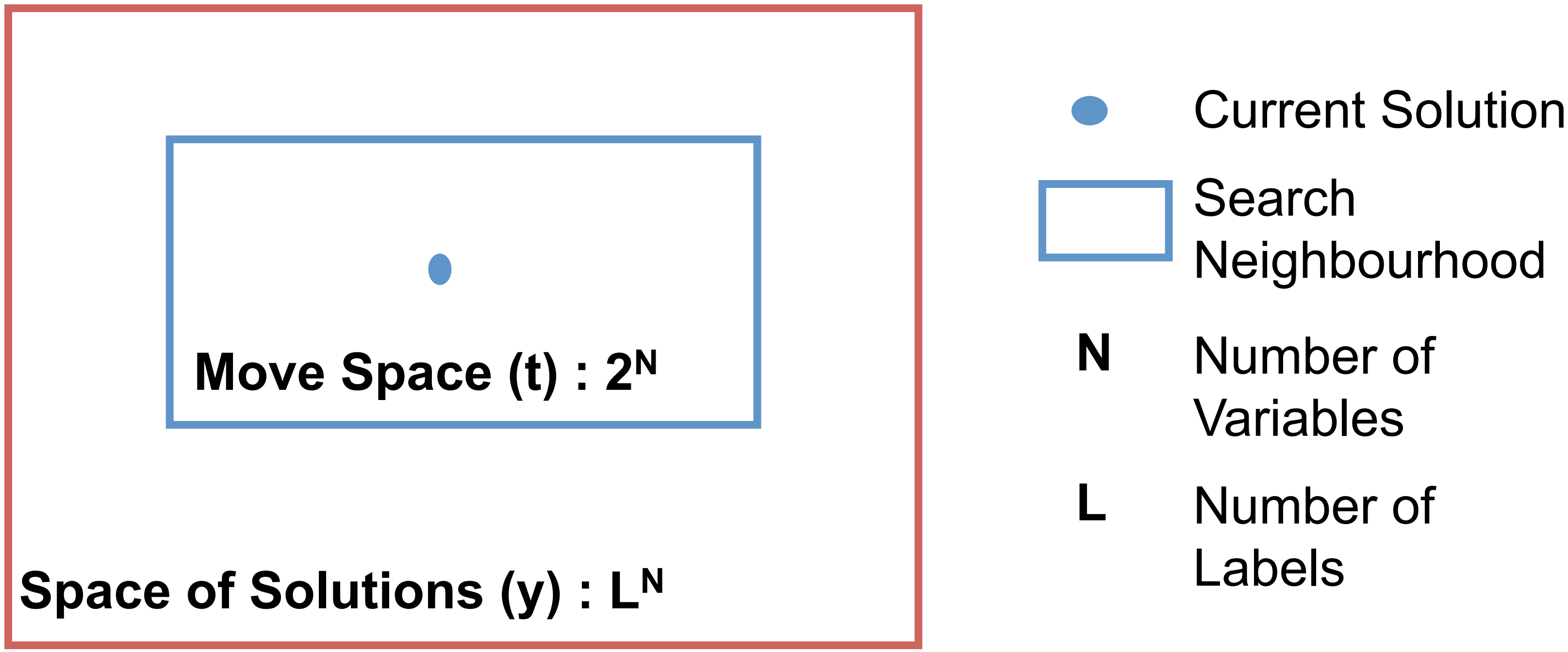
Bigger move space →
- **Better solutions**
- **Finding the optimal move hard**

Moves using Graph Cuts

Expansion and Swap move algorithms

[Boykov Veksler and Zabih, PAMI 2001]

- **Makes a series of changes to the solution (moves)**
- **Each move results in a solution with smaller energy**

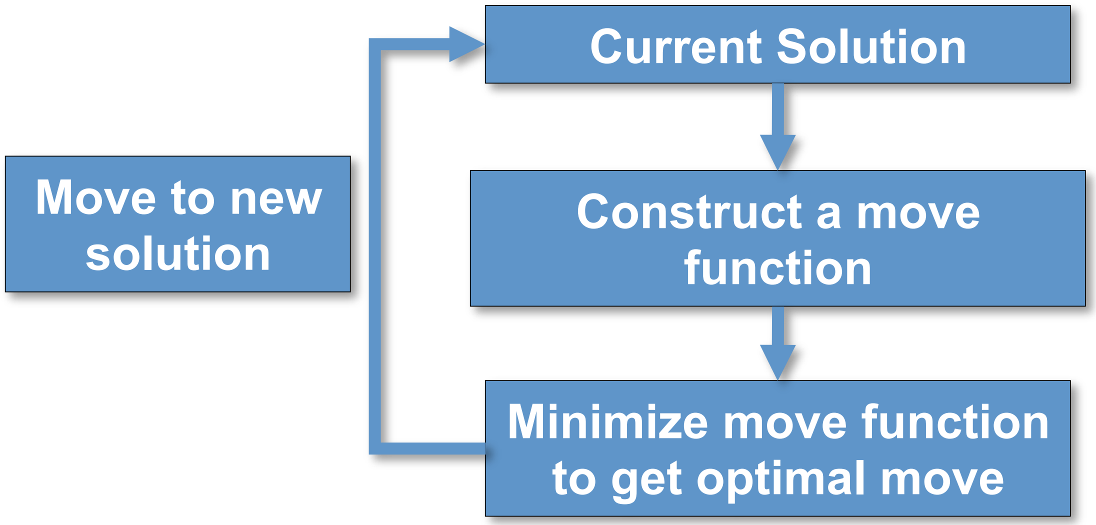


Moves using Graph Cuts

Expansion and Swap move algorithms

[Boykov Veksler and Zabih, PAMI 2001]

- **Makes a series of changes to the solution (moves)**
- **Each move results in a solution with smaller energy**



How to minimize move functions?

General Binary Moves

$$y = t\, y^1 + (1 - t)\, y^2$$

New solution — Current Solution — Second solution

$$E_m(t) = E(t\, y^1 + (1 - t)\, y^2)$$

Minimize over move variables t to get the optimal move

Move energy is a submodular QPBF (Exact Minimization Possible)

Boykov, Veksler and Zabih, PAMI 2001

Expansion Move

- Variables take label α or retain current label

→ Tree
Ground
House
Sky

Status: Initialize with Tree

[Boykov, Veksler, Zabih]

Expansion Move

- Variables take label α or retain current label

Status: Expand Ground

[Boykov, Veksler, Zabih]

Expansion Move

- Variables take label *α* or retain current label

Tree
Ground
→ House
Sky

Status: Expand House

[Boykov, Veksler, Zabih]

Expansion Move

- Variables take label *α* or retain current label

Tree
Ground
House
Sky

Status: Expand Sky

[Boykov, Veksler, Zabih]

Expansion Move

- Variables take label *α* or retain current label

- Move energy is submodular if:
  - Unary Potentials: Arbitrary
  - Pairwise potentials: Metric

$$\theta_{ij}(l_a, l_b) \geq 0$$

$$\theta_{ij}(l_a, l_b) = 0 \quad \text{iff} \quad a = b$$

Semi metric

Examples: Potts model, Truncated linear

Cannot solve truncated quadratic

[Boykov, Veksler, Zabih]

Expansion Move

- Variables take label *α* or retain current label

- Move energy is submodular if:
  - Unary Potentials: Arbitrary
  - Pairwise potentials: Metric

$$\theta_{ij}(l_a,l_b) + \theta_{ij}(l_b,l_c) \geq \theta_{ij}(l_a,l_c)$$

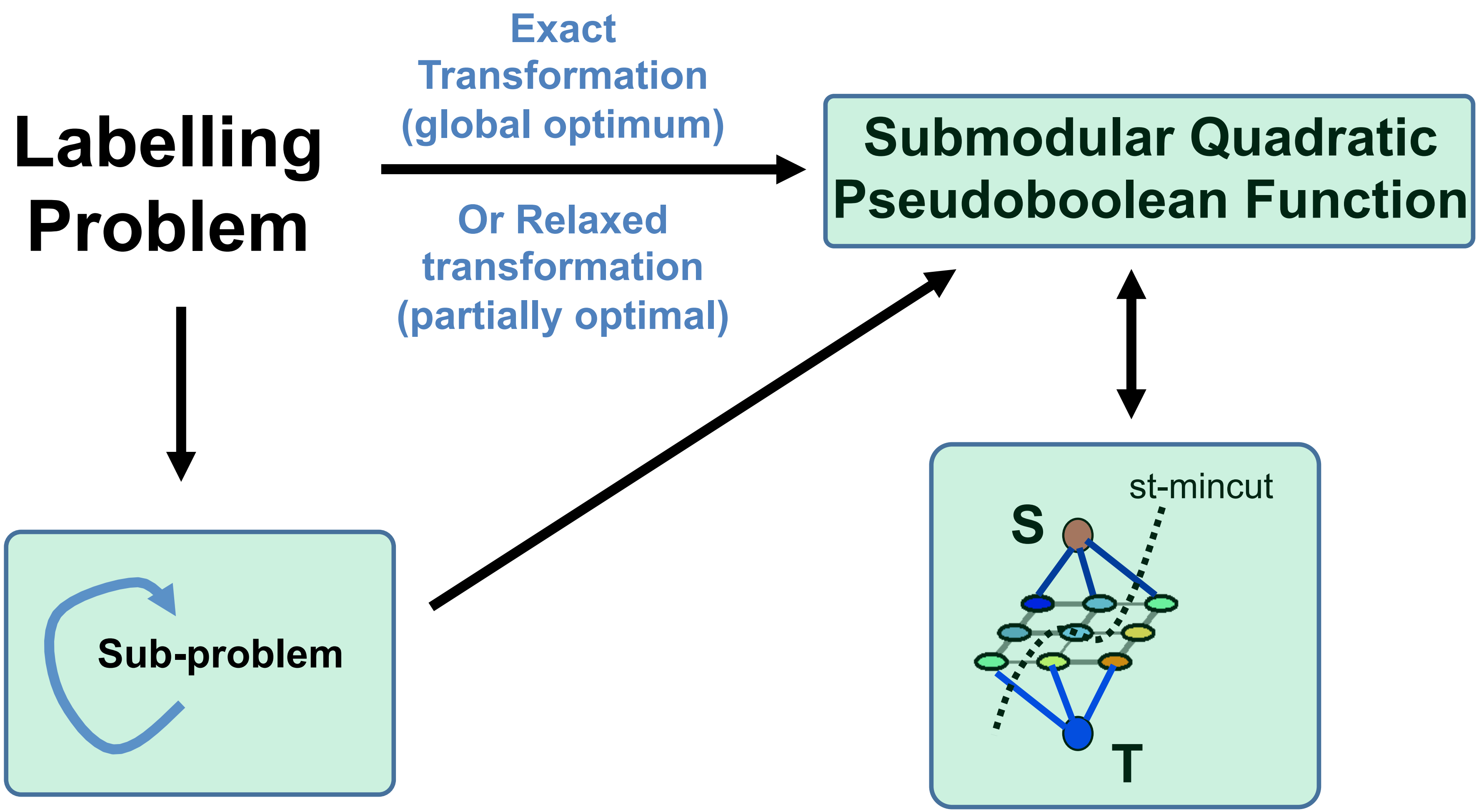
Triangle Inequality

Examples: Potts model, Truncated linear

Cannot solve truncated quadratic

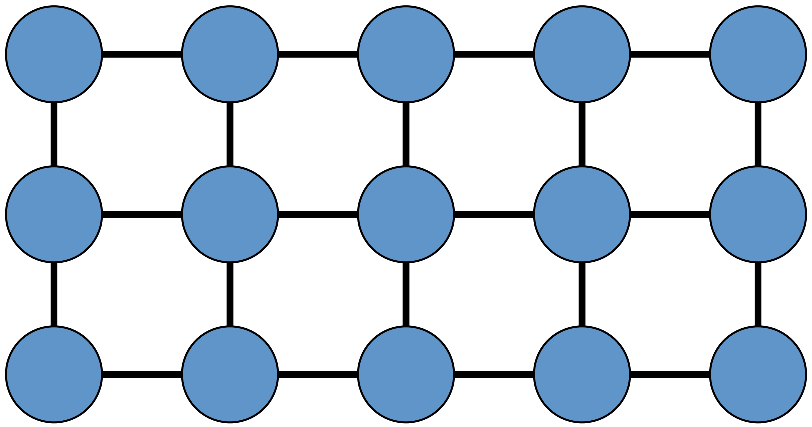
[Boykov, Veksler, Zabih]

Summary



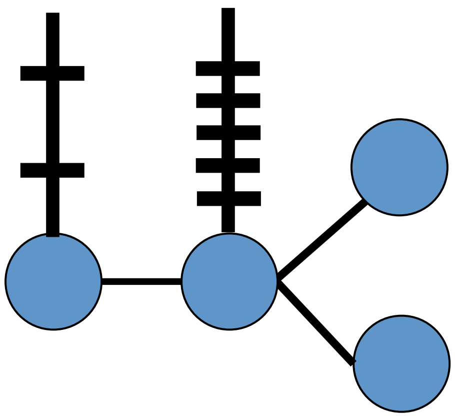
Move making algorithms

Where do we stand ?



Grid graph -
“submodular”: Use graph cuts
“metric”: Use expansion

otherwise: Use TRW,
dual decomposition,
relaxation



Chain/Tree, 2/multi-label: Use BP

What have we seen?

- Inference
  - Belief propagation
  - Graph cuts
  - Variational inference
  - Simulation-based inference
- **Learning**

Outline

- Supervised Learning
- Probabilistic Methods
- Loss-based Methods

Image Classification

Which city is this?

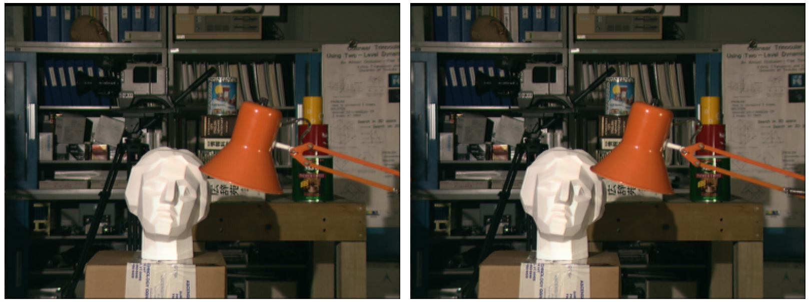
Input: **d** Output: $\mathbf{x} \in \{1,2,\ldots,h\}$

CRF training

- Stereo matching:
  - Z: left, right image
  - X: disparity map

Goal of training:
estimate proper **w**

$f:$



Z

X

$$f = \arg\min_{\mathbf{x}} \mathrm{MRF}_G(\mathbf{x}; \mathbf{u}, \mathbf{h})$$

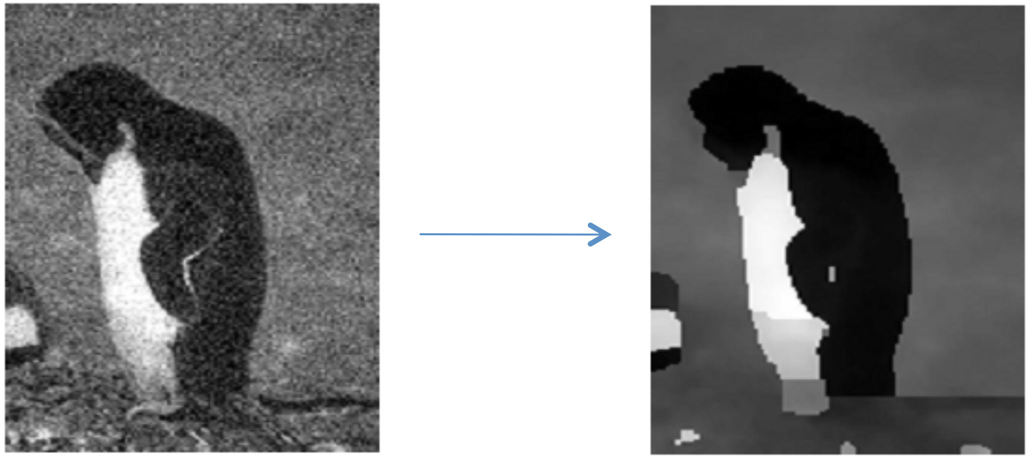
parameterized by **w**

CRF training

- Denoising:
  - Z: noisy input image
  - X: denoised output image

Goal of training:
estimate proper **w**

$f:$



Z X

$$f = \arg\min_{\mathbf{x}} \mathrm{MRF}_G(\mathbf{x}; \mathbf{u}, \mathbf{h})$$

parameterized by **w** ($\mathbf{u}$, $\mathbf{h}$)

CRF training (some further notation)

$$\mathrm{MRF}_G(\mathbf{x};\mathbf{u}^k,\mathbf{h}^k) = \sum_p u_p^k(x_p) + \sum_c h_c^k(\mathbf{x}_c)$$

$$u_p^k(x_p) = \mathbf{w}^T g_p(x_p, \mathbf{z}^k), \;\; h_c^k(\mathbf{x}_c) = \mathbf{w}^T g_c(\mathbf{x}_c, \mathbf{z}^k)$$

vector valued feature functions

$$\mathrm{MRF}_G(\mathbf{x};\mathbf{w},\mathbf{z}^k) = \mathbf{w}^T \left(\sum_p g_p(x_p, \mathbf{z}^k) + \sum_c g_c(\mathbf{x}_c, \mathbf{z}^k) \right) = \mathbf{w}^T g(\mathbf{x}, \mathbf{z}^k)$$

Learning formulations

Risk minimization

$$\hat{\mathbf{x}}^k = \arg\min_{\mathbf{x}} \mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k)$$

$$\min_{\mathbf{w}} \sum_{k=1}^{K} \Delta\left(\mathbf{x}^k, \hat{\mathbf{x}}^k\right)$$

K training samples $\left\{(\mathbf{x}^k, \mathbf{z}^k)\right\}_{k=1}^{K}$

Regularized Risk minimization

$$\hat{\mathbf{x}}^k = \arg\min_{\mathbf{x}} \mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k)$$

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_{k=1}^{K} \Delta\left(\mathbf{x}^k, \hat{\mathbf{x}}^k\right)$$

$$R(\mathbf{w}) = ||\mathbf{w}||^2, \;\; ||\mathbf{w}||_1, \;\; \text{etc.}$$

Regularized Risk minimization

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_{k=1}^{K} L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right)$$

↑

Replace Δ(.) with easier to handle upper bound L_G
(e.g., convex w.r.t. **w**)

↑

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_{k=1}^{K} \Delta\left(\mathbf{x}^k, \hat{\mathbf{x}}^k\right)$$

Choice 1: Hinge loss

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_{k=1}^{K} L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right)$$

$$L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right) = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) - \min_{\mathbf{x}}\left(\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k) - \Delta(\mathbf{x}, \mathbf{x}^k)\right)$$

- Upper bounds Δ(.)
- Leads to **max-margin learning**

Max-margin learning

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_k \xi_k$$

subject to the constraints:

$$\mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) \leq \mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k) - \Delta(\mathbf{x}, \mathbf{x}^k) + \xi_k$$

energy of ground truth — any other energy — desired margin — slack

Max-margin learning

CONSTRAINED

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_k \xi_k$$

subject to the constraints:

$$\mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) \leq \mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k) - \Delta(\mathbf{x}, \mathbf{x}^k) + \xi_k$$

or equivalently

UNCONSTRAINED

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_k \xi_k$$

$$\xi_k = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) - \min_{\mathbf{x}} \left(\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k) - \Delta(\mathbf{x}, \mathbf{x}^k) \right)$$

Choice 2: logistic loss

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_{k=1}^{K} L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right)$$

$$L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right) = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) + \underbrace{\log \sum_{\mathbf{x}} e^{-\mathrm{MRF}_G(\mathbf{x};\mathbf{w},\mathbf{z}^k)}}_{\text{partition function}}$$

- Can be shown to lead to **maximum likelihood learning**

Max-margin vs Maximum-likelihood

max-margin

$$L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right) = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) - \min_{\mathbf{x}} \left(\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k) - \Delta(\mathbf{x}, \mathbf{x}^k)\right)$$

$$L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right) = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) + \log \sum_{\mathbf{x}} e^{-\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k)}$$

maximum likelihood

Max-margin vs Maximum-likelihood

max-margin

$$L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right) = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) + \max_{\mathbf{x}} \left(-\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k) + \Delta(\mathbf{x}, \mathbf{x}^k)\right)$$

soft-max

$$L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right) = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) + \log \sum_{\mathbf{x}} e^{-\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k)}$$

maximum likelihood

Solving the learning formulations

Maximum-likelihood learning

$$\min_{\mathbf{w}} \frac{\mu}{2}||\mathbf{w}||^2 + \sum_{k=1}^{K} L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right)$$

$$L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right) = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) + \underbrace{\log \sum_{\mathbf{x}} e^{-\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k)}}_{\text{partition function}}$$

- Differentiable & convex
- Global optimum via gradient descent, for example

Maximum-likelihood learning

$$\min_{\mathbf{w}} \frac{\mu}{2}||\mathbf{w}||^2 + \sum_{k=1}^{K} L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right)$$

$$L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right) = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) + \log \sum_{\mathbf{x}} e^{-\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k)}$$

gradient ⟶ $\nabla_{\mathbf{w}} = \mathbf{w} + \sum_k \left(g(\mathbf{x}^k, \mathbf{z}^k) - \sum_{\mathbf{x}} p(\mathbf{x}|w, \mathbf{z}^k) g(\mathbf{x}, \mathbf{z}^k) \right)$

Recall that: $\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k) = \mathbf{w}^T g(\mathbf{x}, \mathbf{z}^k)$

Maximum-likelihood learning

$$\min_{\mathbf{w}} \frac{\mu}{2}||\mathbf{w}||^2 + \sum_{k=1}^{K} L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right)$$

$$L_G\left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right) = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) + \log \sum_{\mathbf{x}} e^{-\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k)}$$

gradient $\longrightarrow \nabla_{\mathbf{w}} = \mathbf{w} + \sum_k \left(g(\mathbf{x}^k, \mathbf{z}^k) - \underbrace{\sum_{\mathbf{x}} p(\mathbf{x}|w, \mathbf{z}^k) g(\mathbf{x}, \mathbf{z}^k)} \right)$

- Requires MRF probabilistic inference
- **NP-hard** (exponentially many $\mathbf{x}$): approximation via loopy-BP ?

Max-margin learning (UNCONSTRAINED)

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_{k=1}^{K} L_G \left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right)$$

$$L_G \left(\mathbf{x}^k, \mathbf{z}^k; \mathbf{w}\right) = \mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) - \min_{\mathbf{x}} \left(\mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k) - \Delta(\mathbf{x}, \mathbf{x}^k)\right)$$

- Convex but non-differentiable
- Global optimum via **subgradient method**

Max-margin learning (CONSTRAINED)

$$\min_{\mathbf{w}} \frac{\mu}{2}||\mathbf{w}||^2 + \sum_k \xi_k$$

subject to the constraints:

$$\mathrm{MRF}_G(\mathbf{x}^k; \mathbf{w}, \mathbf{z}^k) \leq \mathrm{MRF}_G(\mathbf{x}; \mathbf{w}, \mathbf{z}^k) - \Delta(\mathbf{x}, \mathbf{x}^k) + \xi_k$$

linear in $\mathbf{w}$

- Quadratic program (great!)
- But exponentially many constraints (not so great)

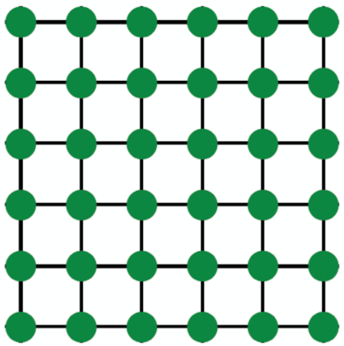
Max-margin learning (CONSTRAINED)

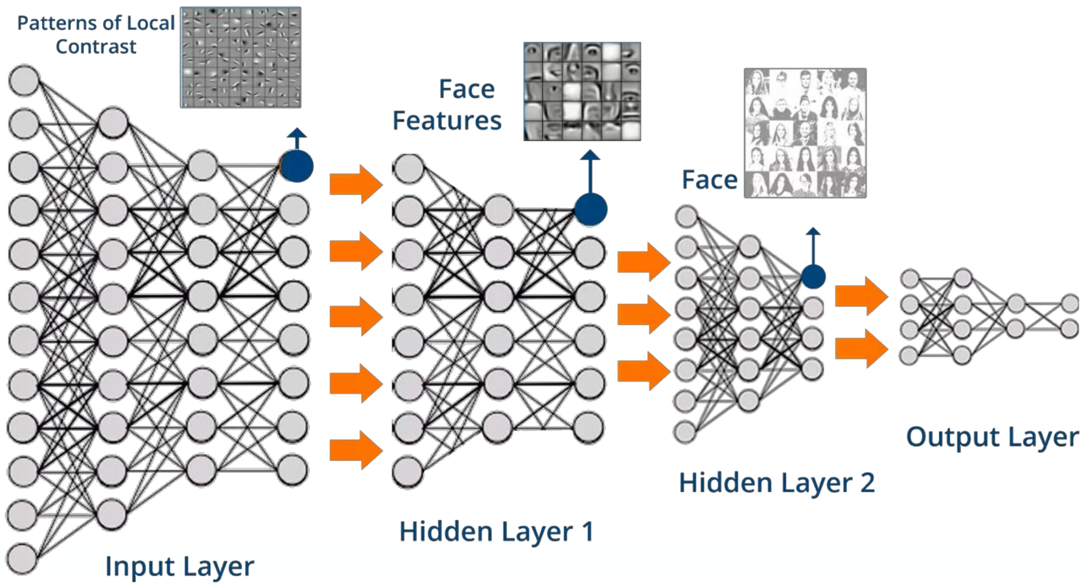
- What if we use only a small number of constraints?
  - Resulting QP can be solved
  - But solution may be infeasible
- **Constraint generation** to the rescue
  - only few constraints **active** at optimal solution !!
    (variables much fewer than constraints)
  - Given the active constraints, rest can be ignored
  - Then let us try to find them!

What have we seen?

- Inference
  - Belief propagation
  - Graph cuts
  - Variational inference
  - Simulation-based inference
- Learning

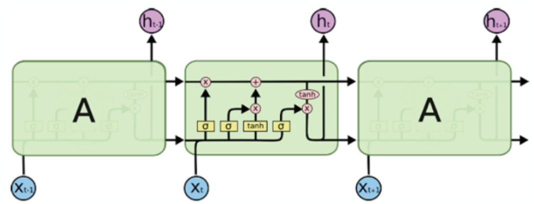
Today: Modern ML Toolbox





Images

Text/Speech



Modern deep learning toolbox is designed for simple sequences & grids

Doubt thou the stars are fire;
Doubt that the sun doth move;
Doubt truth to be a liar;
But never doubt I love...

Text

Audio signals

Images

Slide courtesy: http://cs224w.Stanford.edu

Modern deep learning toolbox is designed for sequences & grids

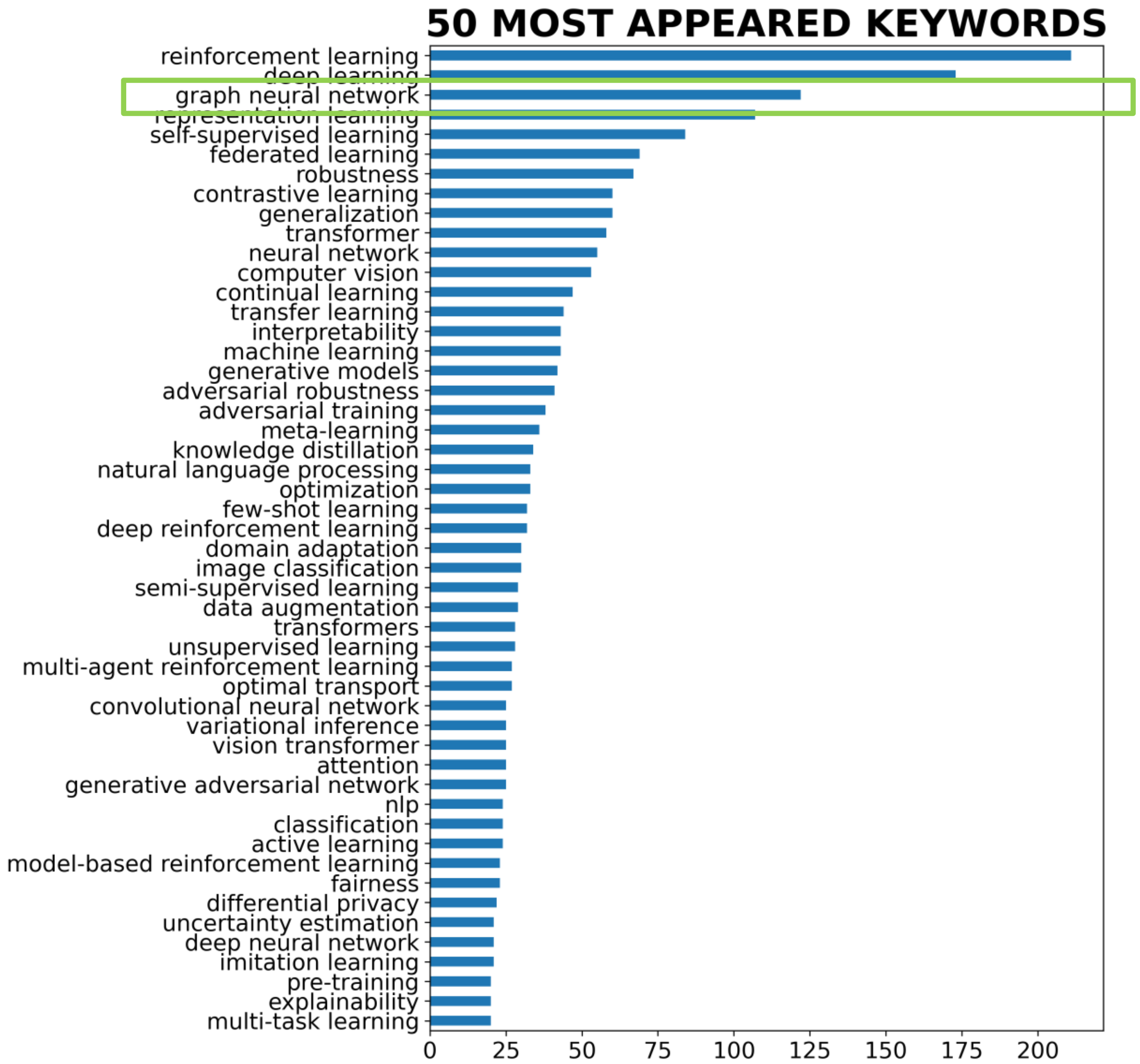
Not everything can be represented as a sequence or a grid

How can we develop neural networks that are much more broadly applicable?

New frontiers beyond classic neural networks that only learn on images and sequences

Hot subfield in ML

ICLR 2022 keywords

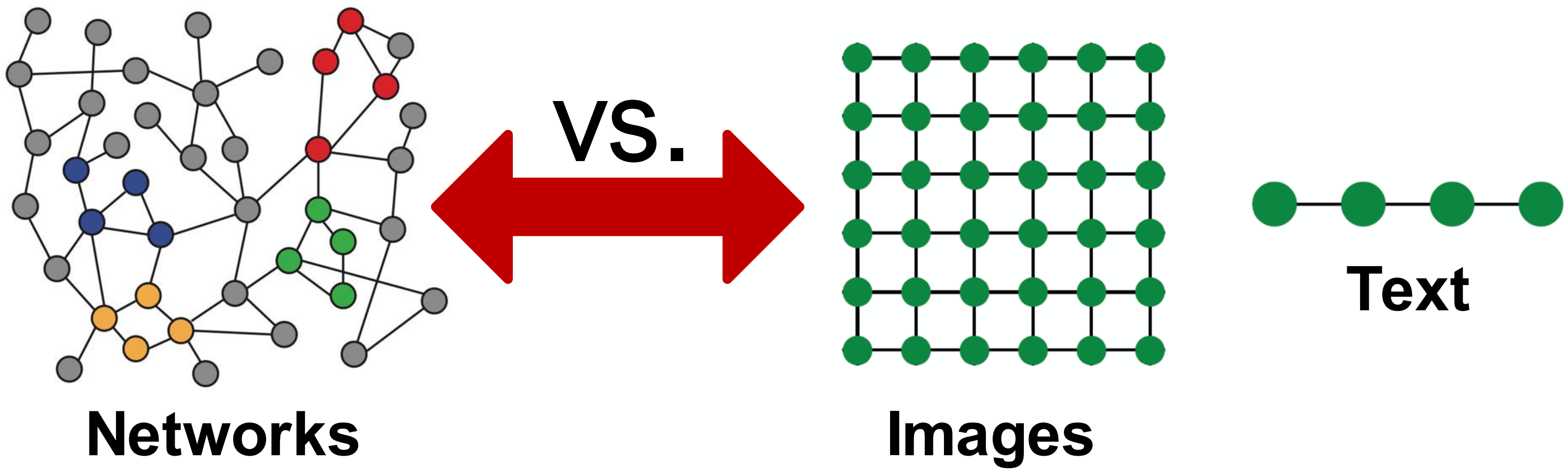


Slide courtesy: http://cs224w.Stanford.edu

Why is Graph Deep Learning Hard?

Networks are complex.

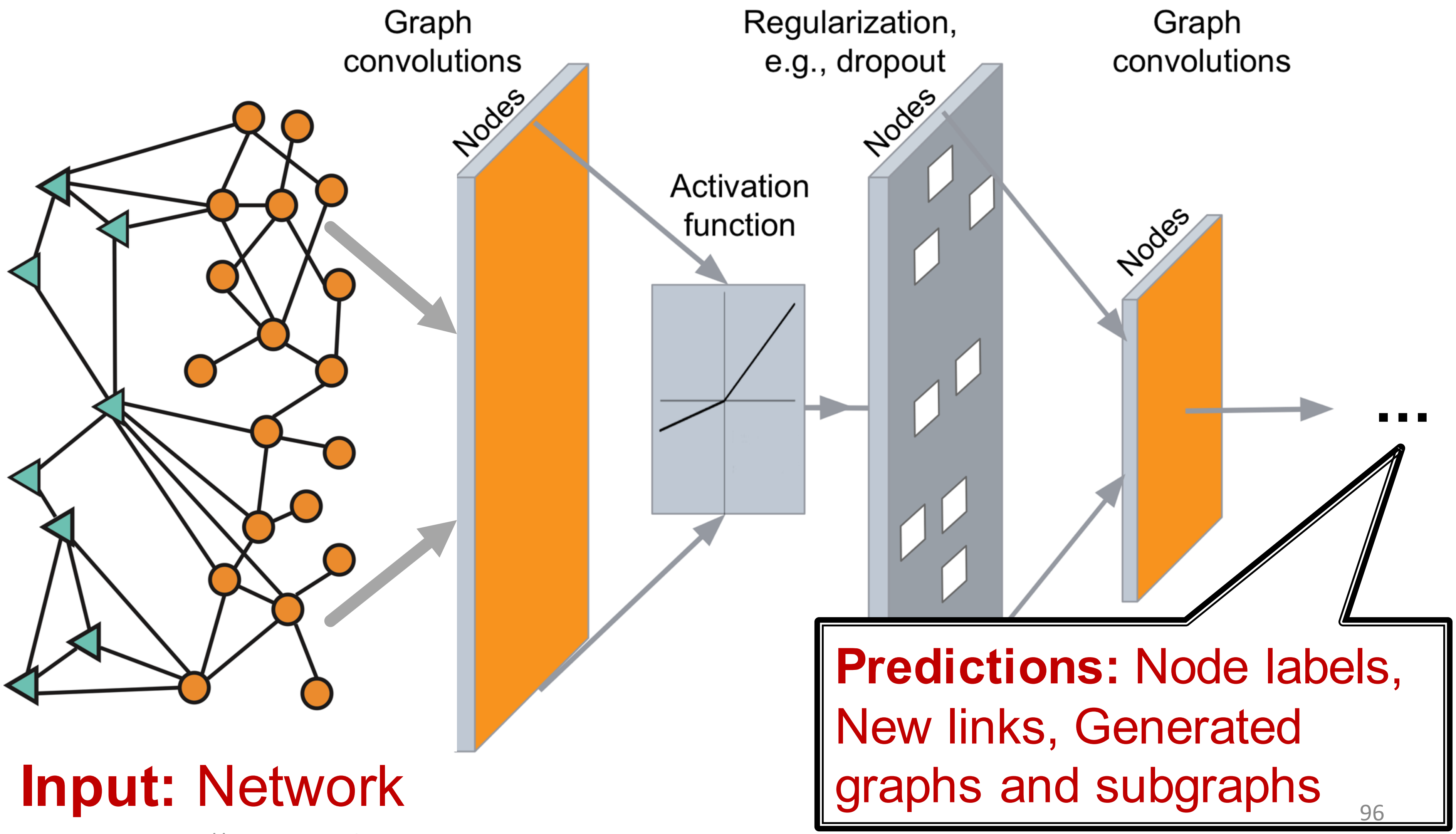
- Arbitrary size and complex topological structure (*i.e.*, no spatial locality like grids)



- No fixed node ordering or reference point
- Often dynamic and have multimodal features

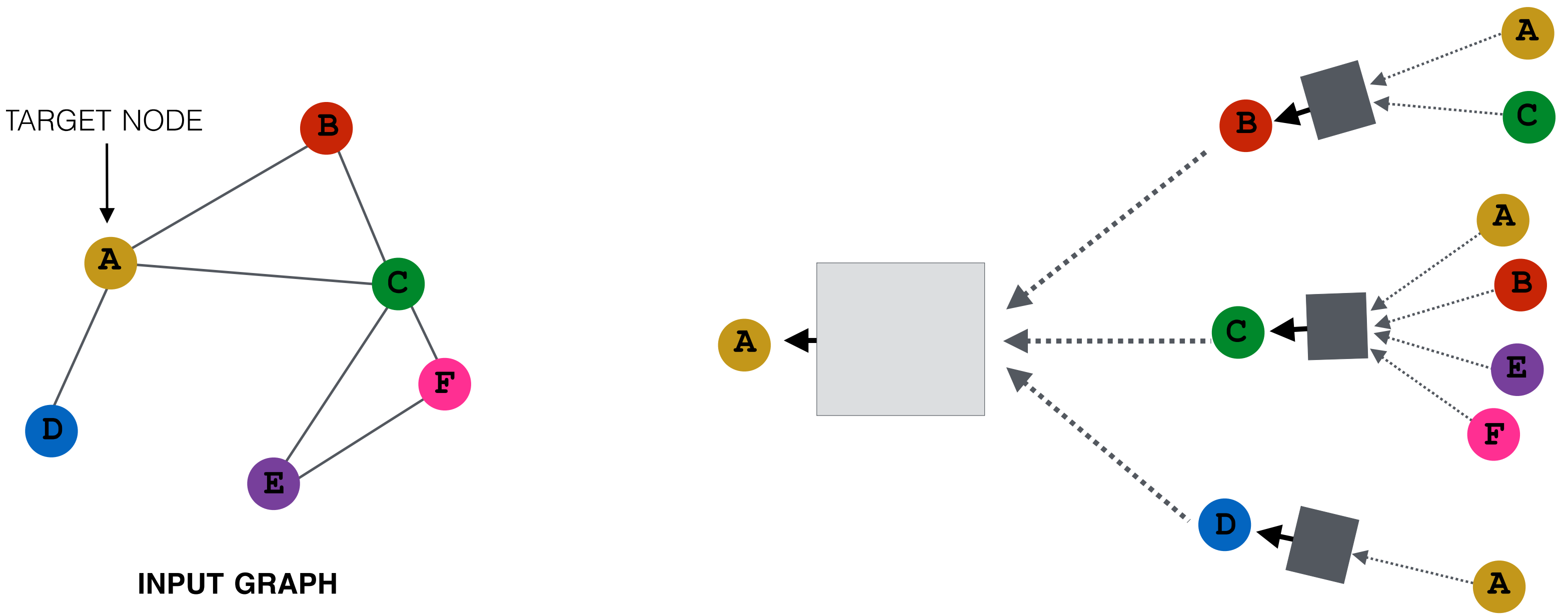
Slide courtesy: http://cs224w.Stanford.edu

ML withGraphs



Slide courtesy: http://cs224w.Stanford.edu

Graph Neural Networks

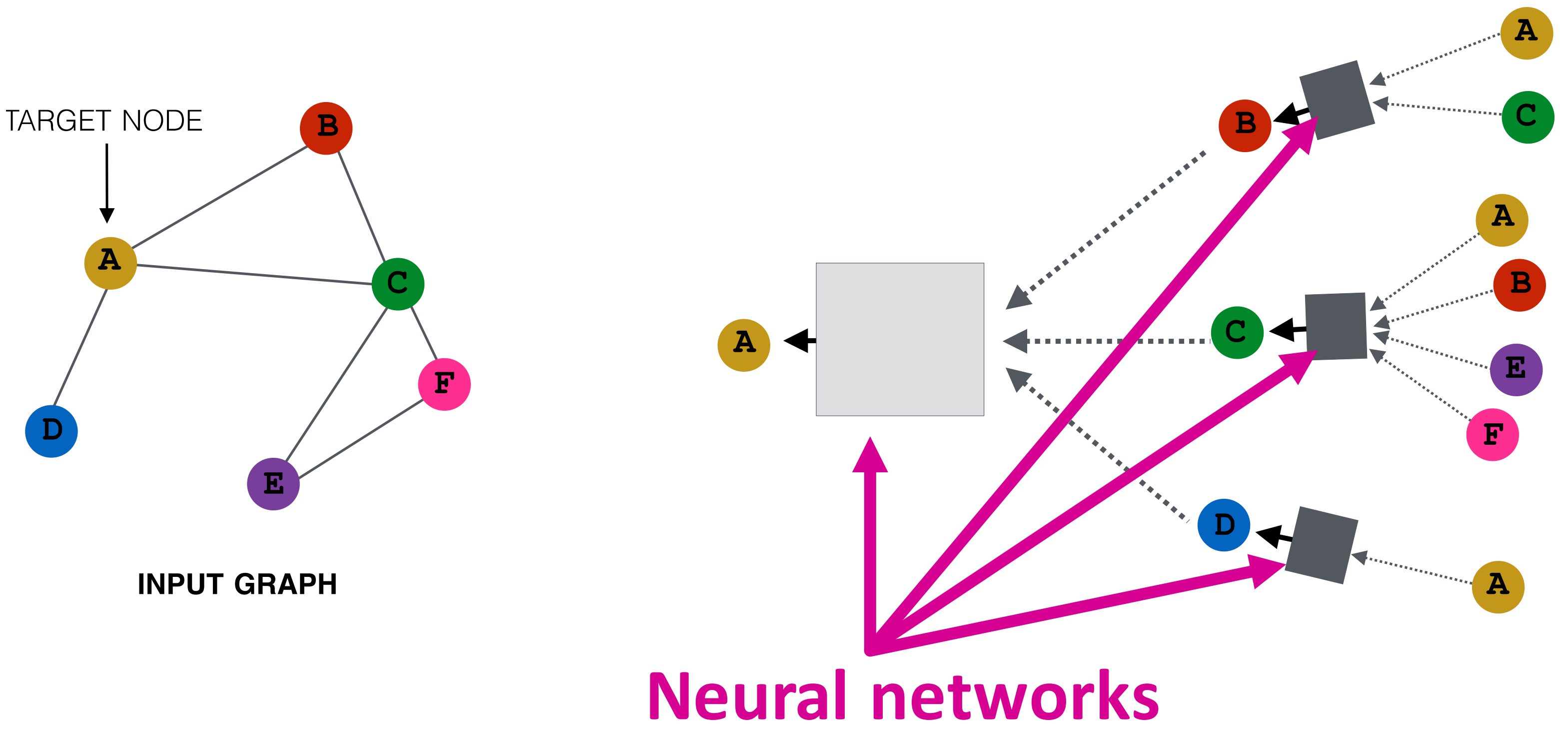


Each node defines a computation graph

- Each edge in this graph is a transformation/aggregation function

Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

Graph Neural Networks



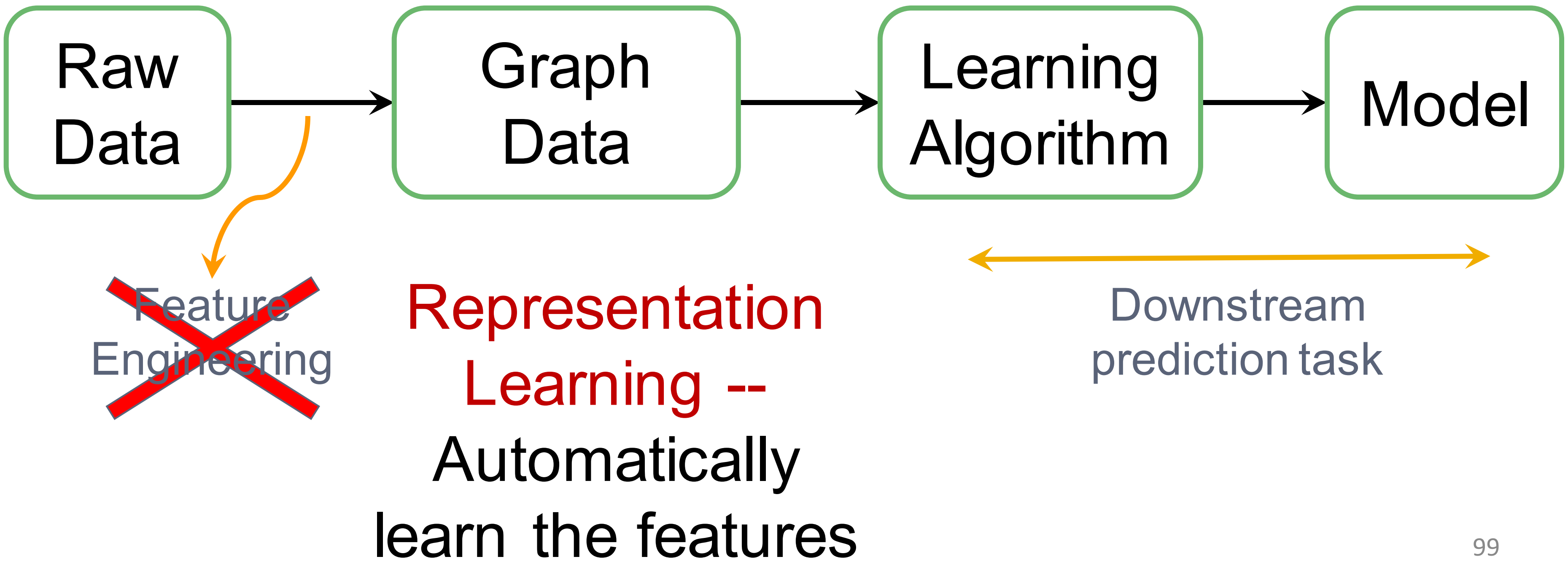
Intuition: Nodes aggregate information from their neighbors using neural networks

Slide courtesy: http://cs224w.Stanford.edu

Inductive Representation Learning on Large Graphs. W. Hamilton, R. Ying, J. Leskovec. NIPS, 2017.

Representation Learning

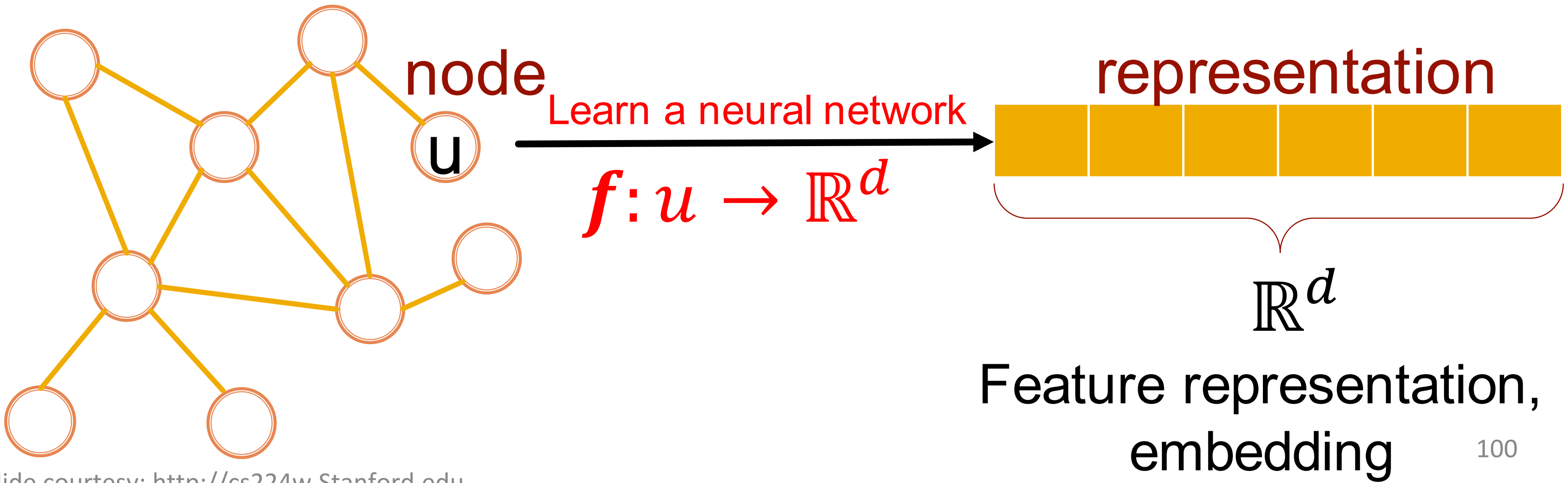
(Supervised) Machine Learning Lifecycle:
This feature, that feature. **Every single time!**



Slide courtesy: http://cs224w.Stanford.edu

Representation Learning

Map nodes to d-dimensional **embeddings** such that similar nodes in the network are embedded close together



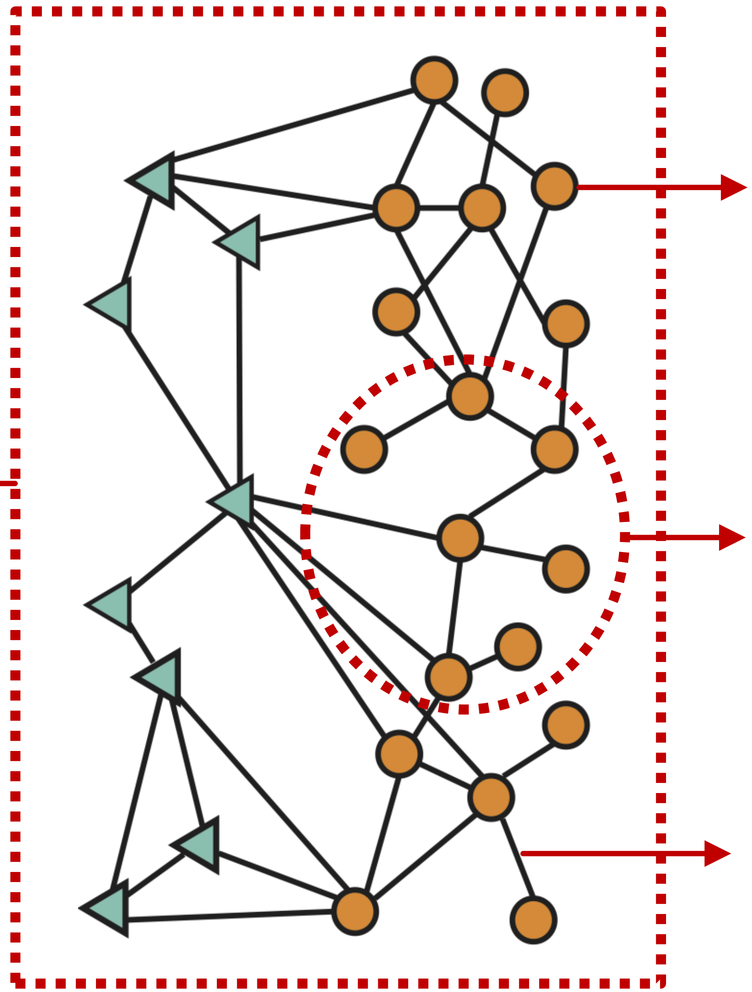
Slide courtesy: http://cs224w.Stanford.edu

ML for Graph data

- Traditional methods
- Node embeddings
- Graph neural networks
- Applications

Different Types of Tasks

Graph-level prediction,
Graph generation



Node level

Community (subgraph) level

Edge-level

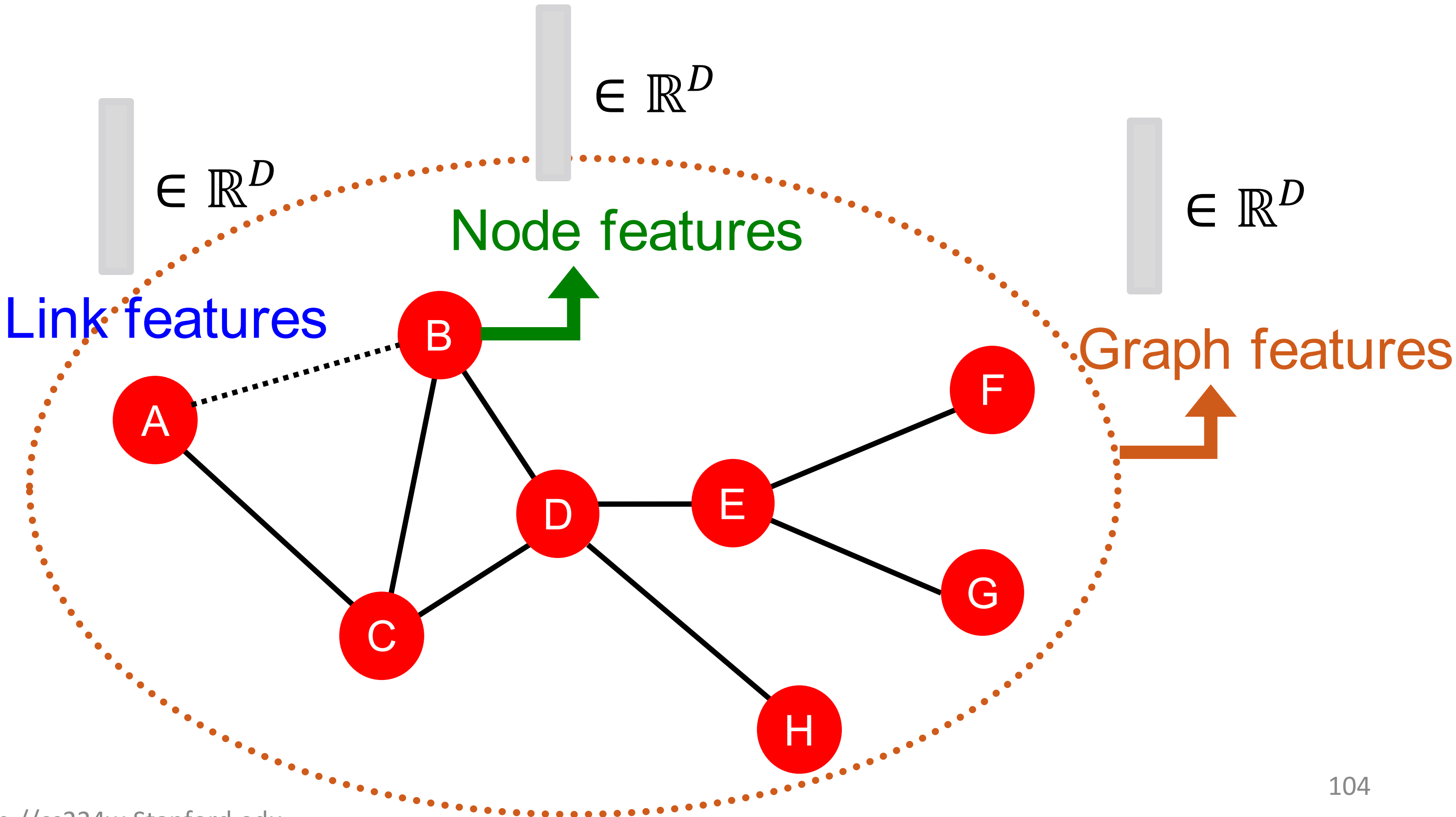
Slide courtesy: http://cs224w.Stanford.edu

Classic Graph ML Tasks

- **Node classification**: Predict a property of a node
  - **Example:** Categorize online users / items
- **Link prediction**: Predict whether there are missing links between two nodes
  - **Example:** Knowledge graph completion
- **Graph classification**: Categorize different graphs
  - **Example:** Molecule property prediction
- **Clustering**: Detect if nodes form a community
  - **Example:** Social circle detection
- **Other tasks**:
  - **Graph generation**: Drug discovery
  - **Graph evolution**: Physical simulation

Traditional ML Pipeline

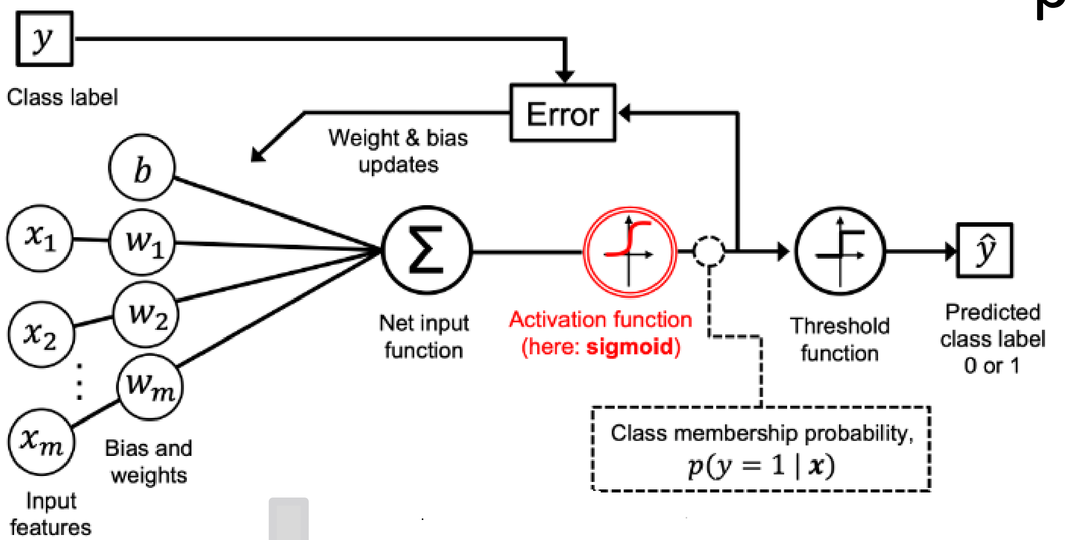
- Design features for nodes/links/graphs
- Obtain features for all training data

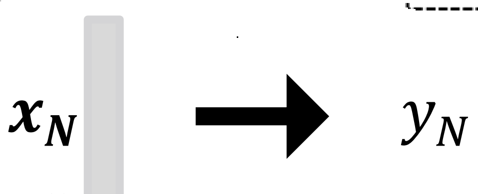


Slide courtesy: http://cs224w.Stanford.edu

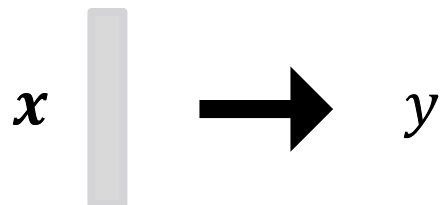
Traditional ML Pipeline

- **Train an ML model:**
  - Logistic Regression
  - Random forest
  - Neural network, etc.





- **Apply the model:**
  - Given a new node/link/graph, obtain its features and make a prediction



Slide courtesy: http://cs224w.Stanford.edu

Machine Learning in Graphs

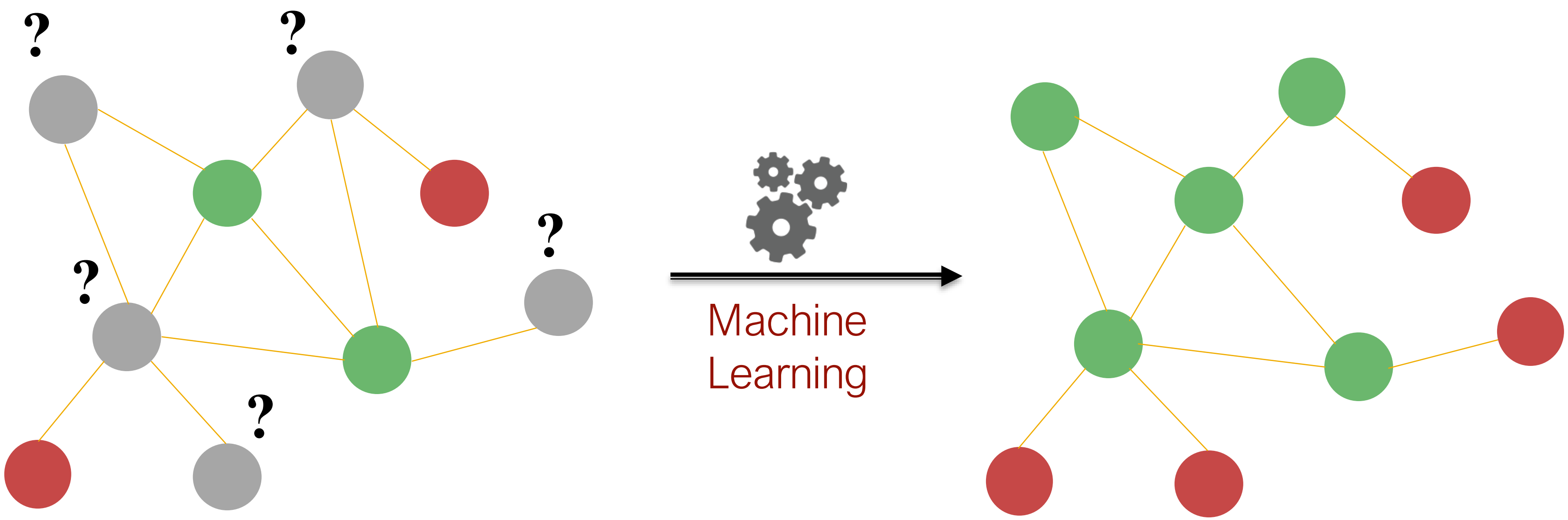
Goal: Make predictions for a set of objects

Design choices:

- **Features:** *d*-dimensional vectors $\boldsymbol{x}$
- **Objects:** Nodes, edges, sets of nodes, entire graphs
- **Objective function:**
  - What task are we aiming to solve?

Slide courtesy: http://cs224w.Stanford.edu

Node-Level Tasks



Node classification

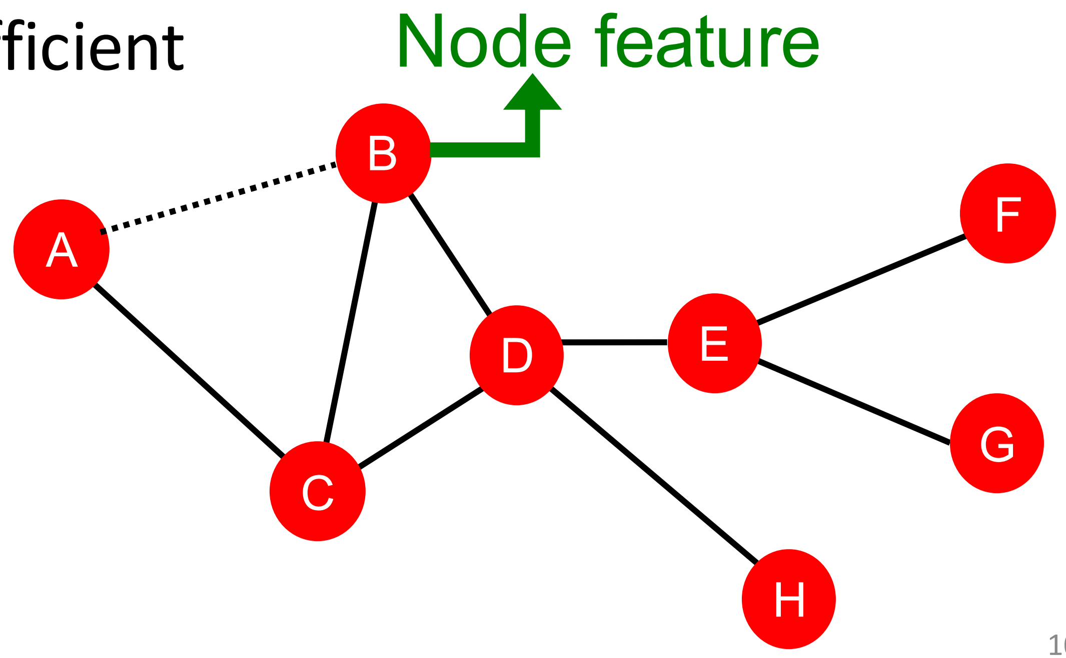
ML needs features.

Slide courtesy: http://cs224w.Stanford.edu

Node-Level Features: Overview

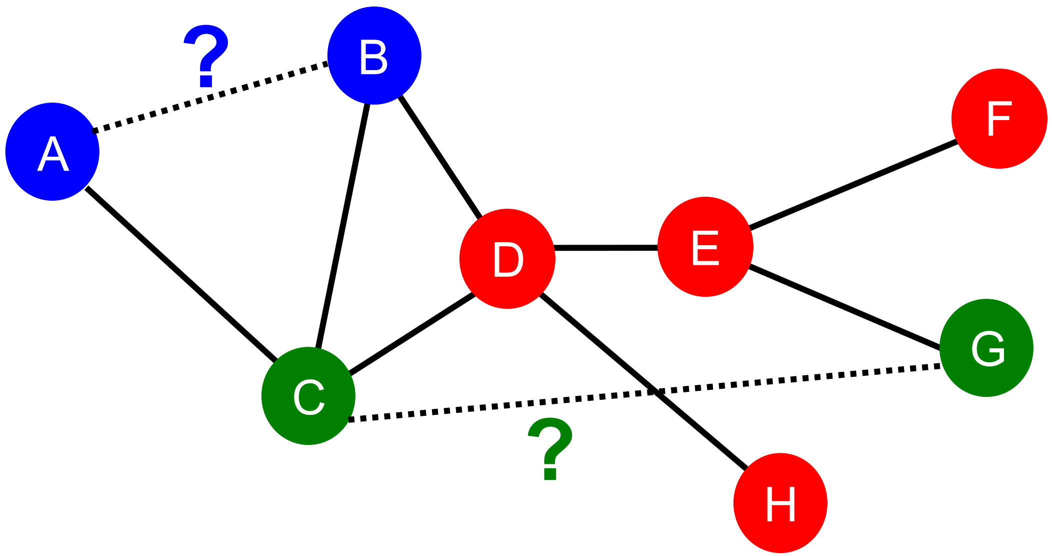
Goal: Characterize the structure and position of a node in the network:

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets



Link-Level Prediction Task: Recap

- The task is to predict **new links** based on the existing links.
- At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted.
- The key is to design features for **a pair of nodes**.

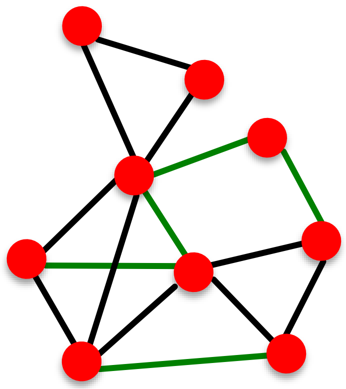


Slide courtesy: http://cs224w.Stanford.edu

Link Prediction as a Task

Two formulations of the link prediction task:

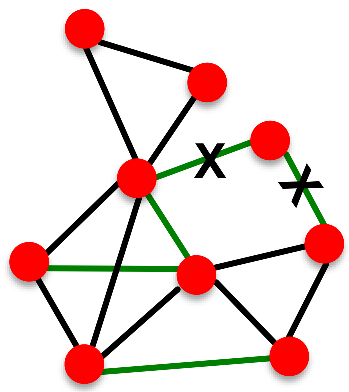
- **1) Links missing at random:**
  - Remove a random set of links and then aim to predict them
- **2) Links over time:**
  - Given $G[t_0, t'_0]$ a graph defined by edges up to time t'_0, **output a ranked list L** of edges (not in $G[t_0, t'_0]$) that are predicted to appear in time $G[t_1, t'_1]$
  - **Evaluation:**
    - $n = |E_{new}|$: # new edges that appear during the test period $[t_1, t'_1]$
    - Take top n elements of L and count correct edges



$G[t_0, t'_0]$
$G[t_1, t'_1]$

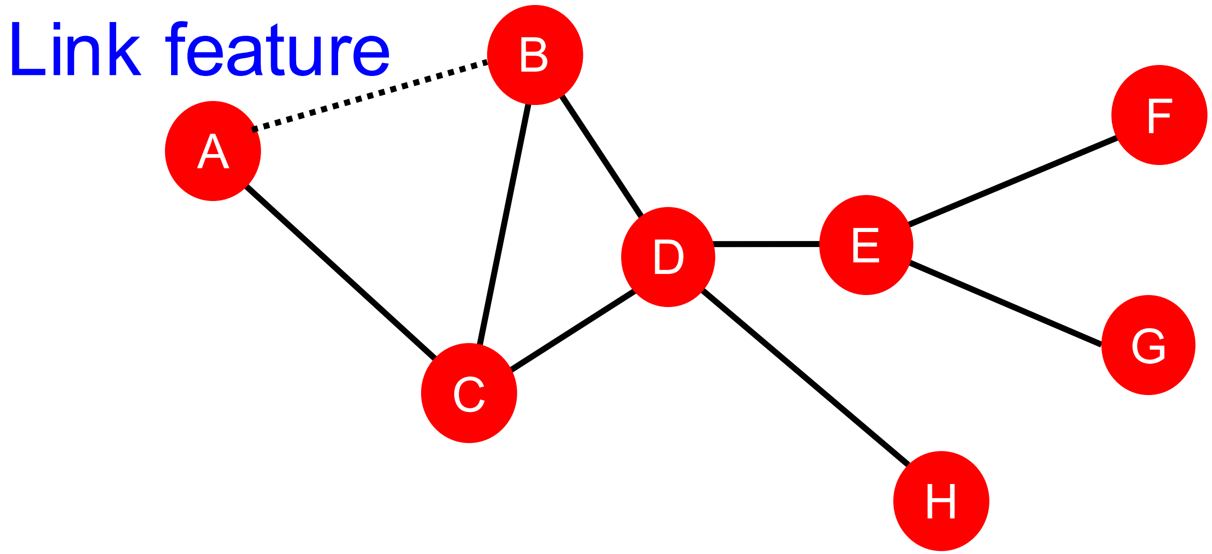
Link Prediction via Proximity

- **Methodology:**
  - For each pair of nodes *(x,y)* compute score *c(x,y)*
    - For example, *c(x,y)* could be the # of common neighbors of *x* and *y*
  - Sort pairs *(x,y)* by the decreasing score *c(x,y)*
  - **Predict top n pairs as new links**
  - **See which of these links actually appear in $G[t_1, t_1']$**



Link-Level Features: Overview

- Distance-based feature
- Local neighborhood overlap
- Global neighborhood overlap



Slide courtesy: http://cs224w.Stanford.edu

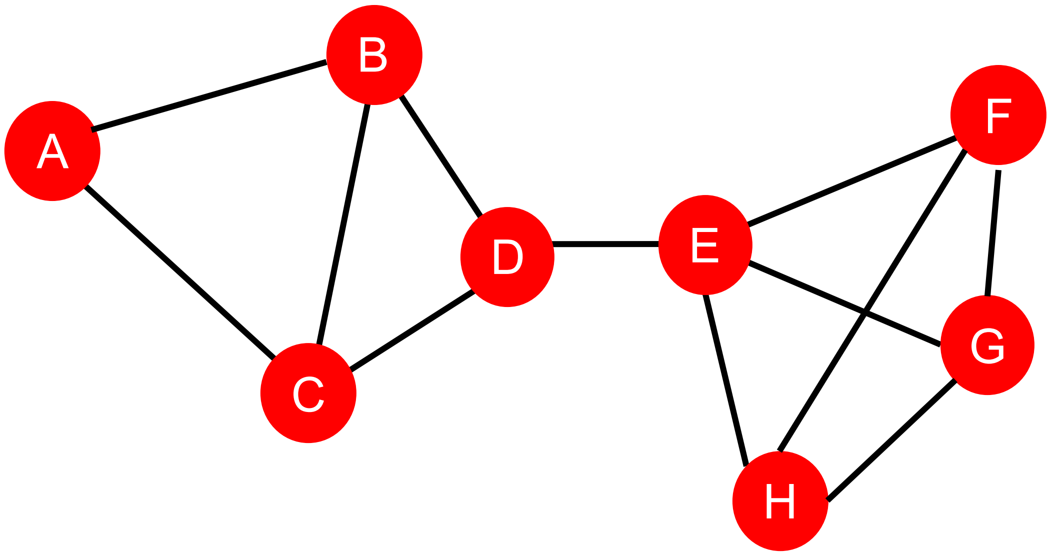
Link-Level Features: Summary

- **Distance-based features:**
  - Uses the shortest path length between two nodes but does not capture how neighborhood overlaps.
- **Local neighborhood overlap:**
  - Captures how many neighboring nodes are shared by two nodes.
  - Becomes zero when no neighbor nodes are shared.
- **Global neighborhood overlap:**
  - Uses global graph structure to score two nodes.
  - Katz index counts #walks of all lengths between two nodes.

Graph-Level Features

- **Goal:** We want features that characterize the structure of an entire graph.

- **For example:**



Background: Kernel Methods

- Kernel methods are widely-used for traditional ML for graph-level prediction.
- **Idea: Design kernels instead of feature vectors.**
- **A quick introduction to Kernels:**
  - Kernel $K(G, G') \in \mathbb{R}$ measures similarity b/w data
  - Kernel matrix $\boldsymbol{K} = \big(K(G, G')\big)_{G,G'}$ must always be positive semidefinite (i.e., has positive eigenvalues)
  - There exists a feature representation $\phi(\cdot)$ such that $K(G, G') = \phi(\mathrm{G})^{\mathrm{T}}\phi(G')$
  - Once the kernel is defined, off-the-shelf ML model, such as kernel SVM, can be used to make predictions.

Graph-Level Features: Overview

- **Graph Kernels:** Measure similarity between two graphs:
  - Graphlet Kernel [1]
  - Weisfeiler-Lehman Kernel [2]
  - Other kernels are also proposed in the literature (beyond the scope of this lecture)
    - Random-walk kernel
    - Shortest-path graph kernel
    - And many more…

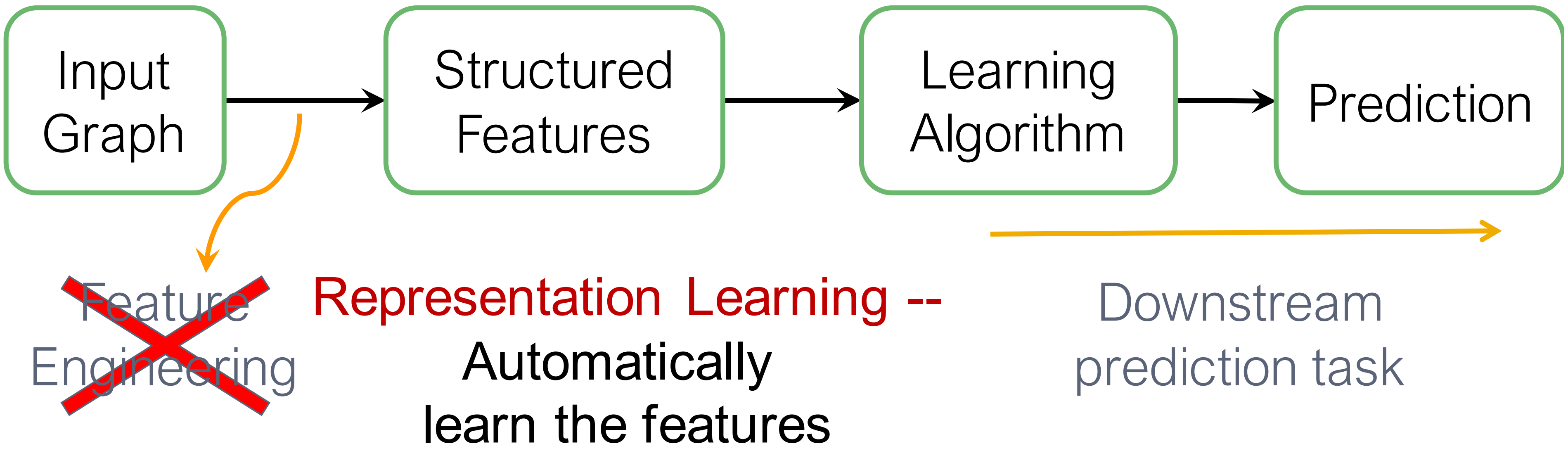
[1] Shervashidze, Nino, et al. "Efficient graphlet kernels for large graph comparison." Artificial Intelligence and Statistics. 2009.
[2] Shervashidze, Nino, et al. "Weisfeiler-lehman graph kernels." Journal of Machine Learning Research 12.9 (2011).

Graph-Level Features: Summary

- **Graphlet Kernel**
  - Graph is represented as **Bag-of-graphlets**
  - **Computationally expensive**
- **Weisfeiler-Lehman Kernel**
  - Apply K-step color refinement algorithm to enrich node colors
    - Different colors capture different K-hop neighborhood structures
  - Graph is represented as **Bag-of-colors**
  - **Computationally efficient**
  - Closely related to Graph Neural Networks (as we will see!)

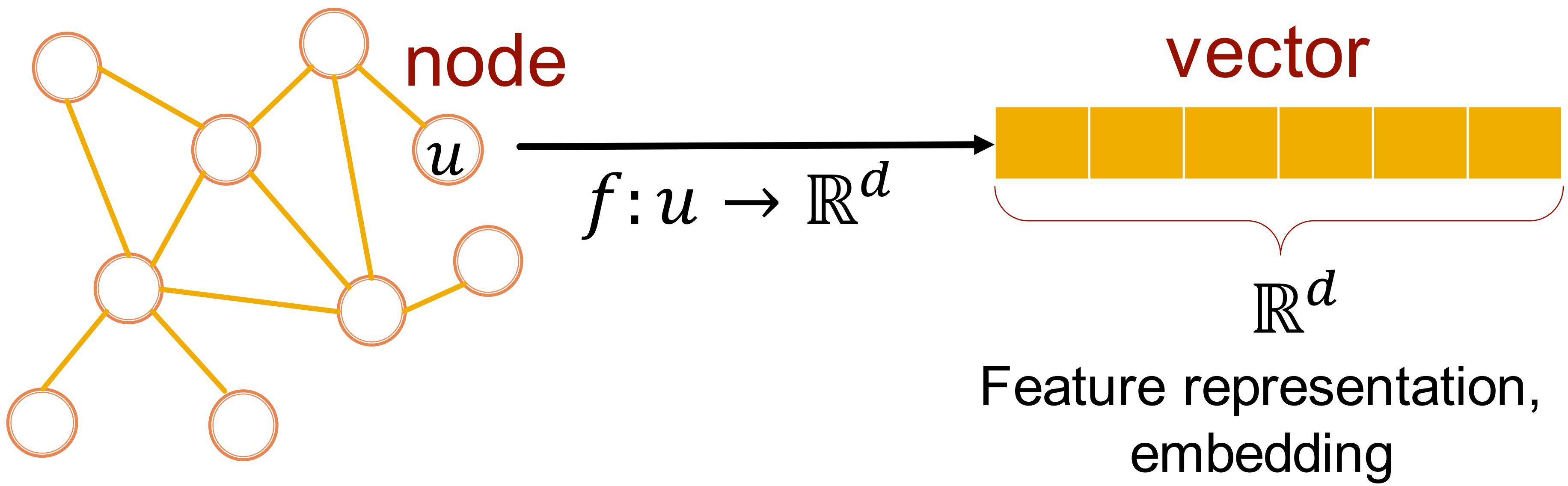
Graph Representation Learning

Graph Representation Learning alleviates the need to do feature engineering every single time.



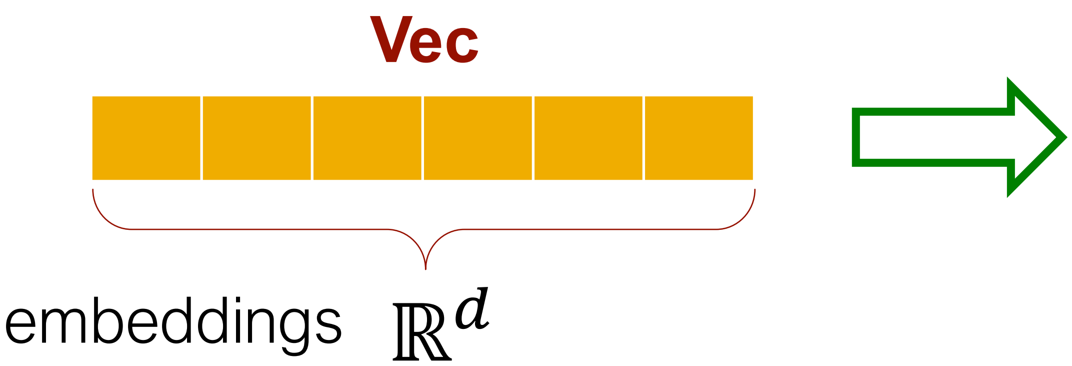
Graph Representation Learning

Goal: Efficient task-independent feature learning for machine learning with graphs!



Why Embedding?

- **Task: Map nodes into an embedding space**
  - Similarity of embeddings between nodes indicates their similarity in the network. For example:
    - Both nodes are close to each other (connected by an edge)
  - Encode network information
  - Potentially used for many downstream predictions

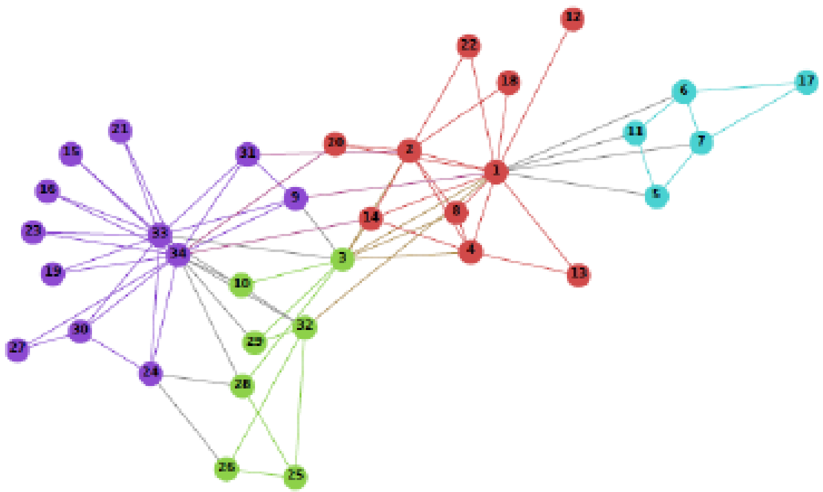


Tasks

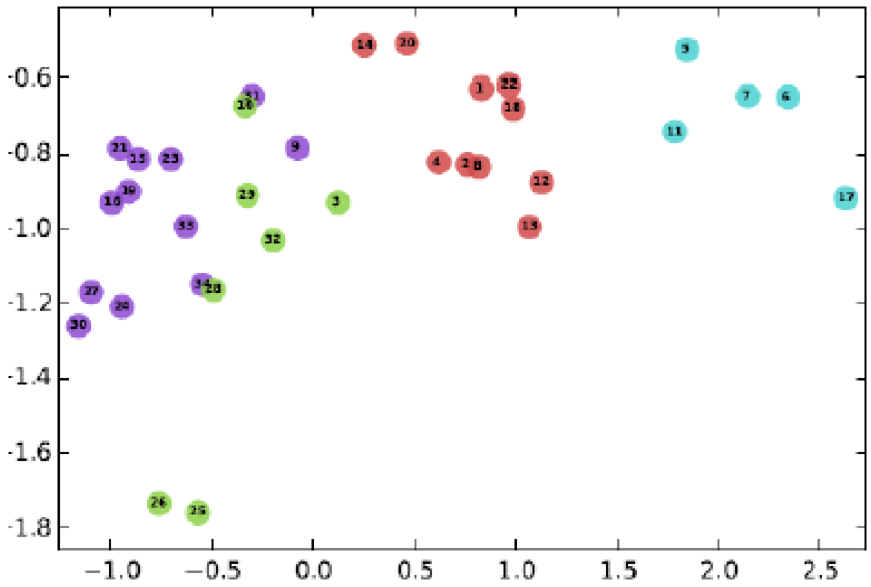
- Node classification
- Link prediction
- Graph classification
- Anomalous node detection
- Clustering
- ….

Example Node Embedding

- **2D embedding of nodes of the Zachary's Karate Club network:**



Input

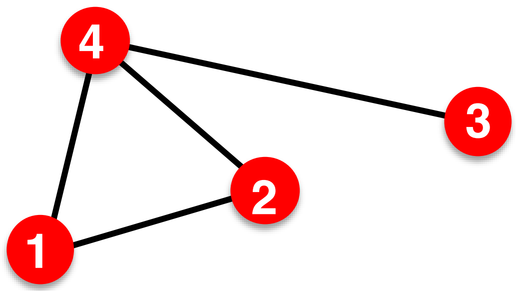


Output

Image from: [Perozzi et al](). DeepWalk: Online Learning of Social Representations. *KDD 2014.*

Setup

- **Assume we have a graph G:**
  - V is the vertex set.
  - $\mathbf{A}$ is the adjacency matrix (assume binary).
  - **For simplicity: No node features or extra information is used**

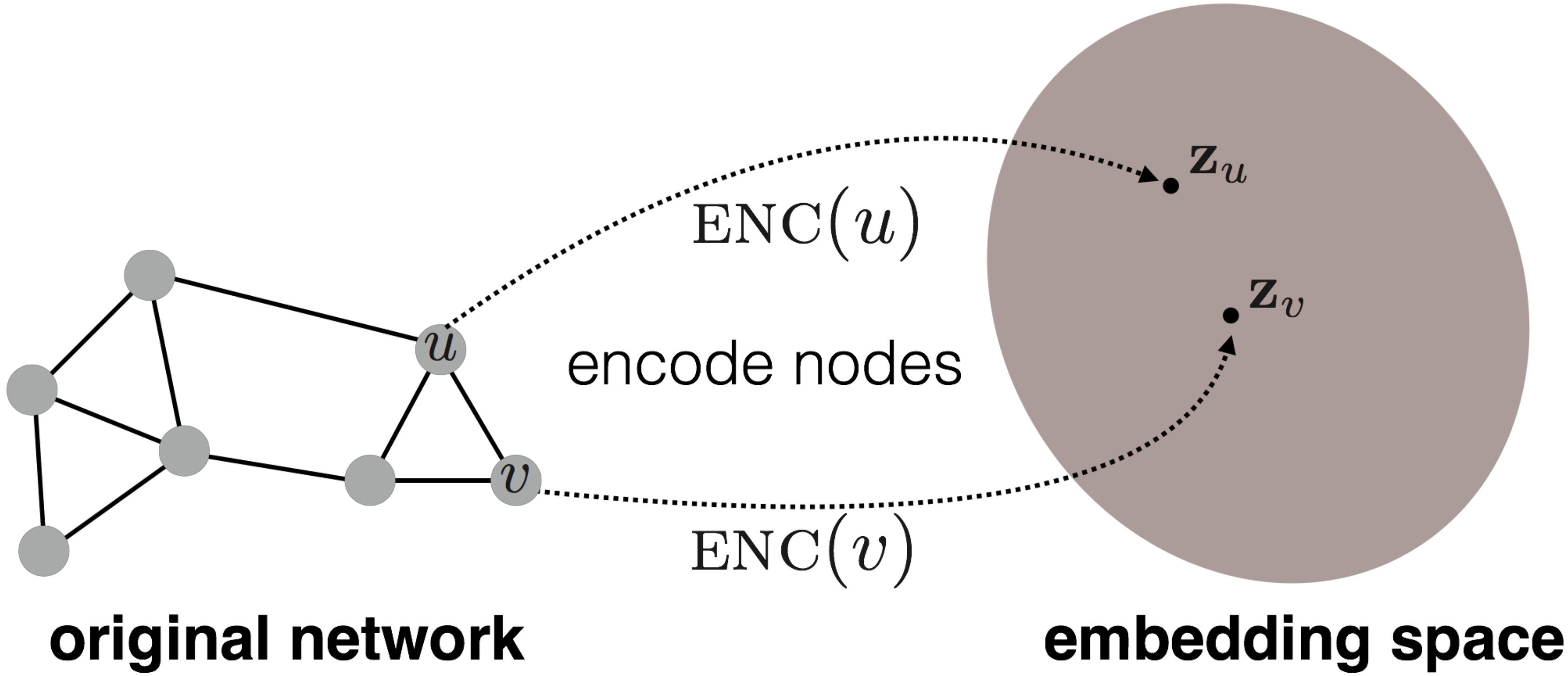


V: {1, 2, 3, 4}

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Embedding Nodes

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph

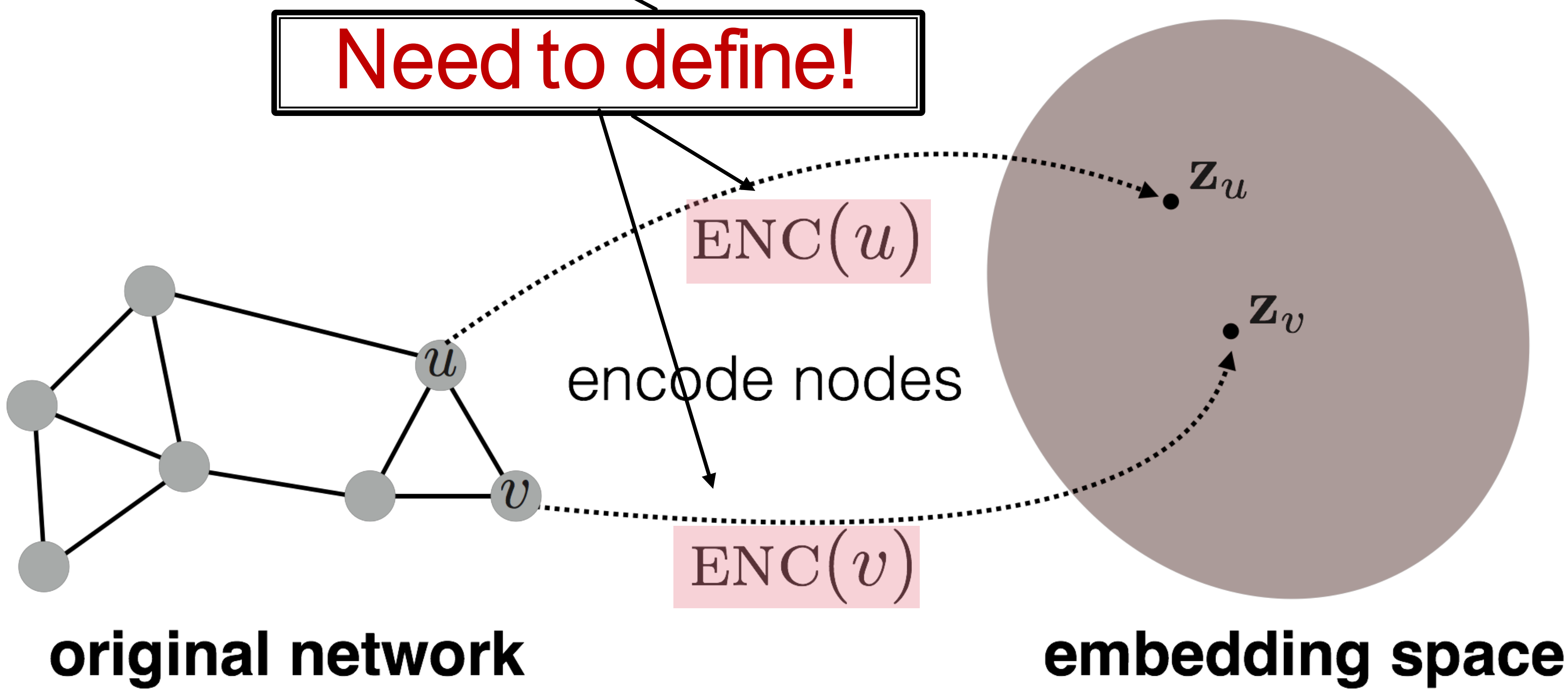


Slide courtesy: http://cs224w.Stanford.edu

Embedding Nodes

Goal: $\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$

in the original network — Similarity of the embedding



Slide courtesy: http://cs224w.Stanford.edu

Learning Node Embeddings

1. **Encoder** maps from nodes to embeddings
2. **Define a node similarity function** (i.e., a measure of similarity in the original network)
3. **Decoder** DEC maps from embeddings to the similarity score
4. **Optimize the parameters of the encoder so that:**

$$\text{DEC}(\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u)$$

$$\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$$

in the original network — Similarity of the embedding

Two Key Components

- **Encoder:** maps each node to a low-dimensional vector

$$\text{ENC}(v) = \mathbf{z}_v$$

d-dimensional embedding

node in the input graph

- **Similarity function:** specifies how the relationships in vector space map to the relationships in the original network

$$\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$$

Decoder

Similarity of u and v in the original network

dot product between node embeddings

"Shallow" Encoding

Simplest encoding approach: **Encoder is just an embedding-lookup**

Each node is assigned a unique embedding vector
(i.e., we directly optimize the embedding of each node)

Many methods: DeepWalk, node2vec

Framework Summary

- **Encoder + Decoder Framework**
  - Shallow encoder: embedding lookup
  - Parameters to optimize: $\mathbf{Z}$ which contains node embeddings $\mathbf{z}_u$ for all nodes $u \in V$
  - We will cover deep encoders (GNNs) in Lecture 6
  - **Decoder:** based on node similarity.
  - **Objective:** maximize $\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$ for node pairs (u, v) that are **similar**

How to Define Node Similarity?

- Key choice of methods is **how they define node similarity.**
- Should two nodes have a similar embedding if they…
  - are linked?
  - share neighbors?
  - have similar “structural roles”?
- There are also random walk based approaches

Note on Node Embeddings

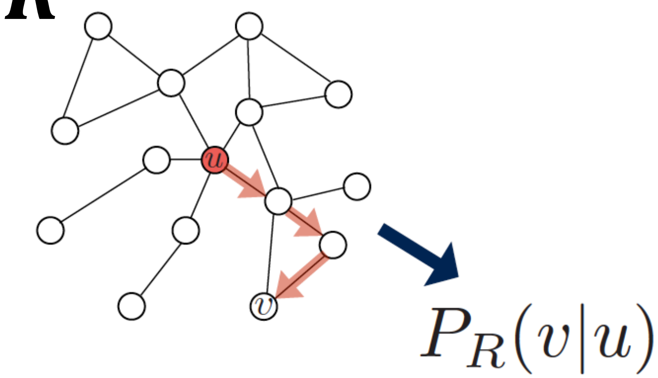
- This is **unsupervised/self-supervised** way of learning node embeddings.
  - We are **not** utilizing node labels
  - We are **not** utilizing node features
  - The goal is to directly estimate a set of coordinates (i.e., the embedding) of a node so that some aspect of the network structure (captured by DEC) is preserved.
- These embeddings are **task independent**
  - They are not trained for a specific task but can be used for any task.

Random-Walk Embeddings

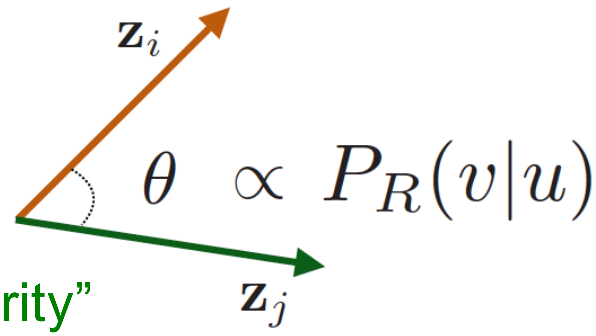
$$\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v \approx$$ probability that u and v co-occur on a random walk over the graph

Random-Walk Embeddings

1. **Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R**



2. **Optimize embeddings to encode these random walk statistics:**



Similarity in embedding space (Here: dot product=$\cos(\theta)$) encodes random walk "similarity"

Why Random Walks?

1. **Expressivity:** Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information
   Idea: if random walk starting from node u visits v with high probability, u and v are similar (high-order multi-hop information)

2. **Efficiency:** Do not need to consider all node pairs when training; only need to consider pairs that co-occur on random walks

Unsupervised Feature Learning

- Intuition: Find embedding of nodes in d-dimensional space that preserves similarity
- Idea: Learn node embedding such that nearby nodes are close together in the network
- Given a node u, how do we define nearby nodes?
  - $N_R(u)$ … neighbourhood of u obtained by some random walk strategy R

Slide courtesy: http://cs224w.Stanford.edu

Feature Learning as Optimization

- Given $G = (V, E)$,
- Our goal is to learn a mapping $f: u \to \mathbb{R}^d$: $f(u) = \mathbf{z}_u$

- Log-likelihood objective:

$$\max_{f} \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$

  - $N_R(u)$ is the neighborhood of node u by strategy R

- Given node u, we want to learn feature representations that are predictive of the nodes in its random walk neighborhood $N_R(u)$.

Random Walk Optimization

1. Run **short fixed-length random walks** starting from each node u in the graph using some random walk strategy R.
2. For each node u collect $N_R(u)$, the multiset* of nodes visited on random walks starting from u.
3. Optimize embeddings according to: Given node u, predict its neighbors $N_{\mathrm{R}}(u)$.

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$ ⟹ Maximum likelihood objective

*$N_R(u)$ can have repeat elements since nodes can be visited multiple times on random walks

Summary so far

- **Core idea:** Embed nodes so that distances in embedding space reflect node similarities in the original network.
- **Different notions of node similarity:**
  - Naïve: similar if two nodes are connected
  - Neighborhood overlap
  - Random walk approaches