Bayesian Networks Learning: A Survey

——Topic Exploration

JIAQI YANG
Department of Computer Science, Graduate Center, CUNY
jyang2@gradcenter.cuny.edu

Review Paper List:

1. Machine learning of Bayesian networks using constraint programming
   -- *Peter van Beek, Hella-Franziska Hoffmann.*

2. Learning Optimal Bayesian Networks Using A* Search
   -- *Changhe Yuan, Brandon Malone, and Xiaojian Wu.*

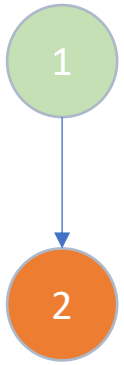
3. Neural Combinatorial Optimization With Reinforcement Learning

   -- *Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, Samy Bengio*

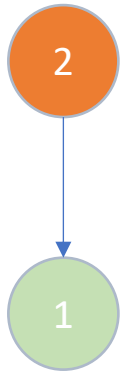
Goal:

- All papers are trying to solve one problem: Find optimal or local optimal Bayesian Networks(DAG).
- Joint Probability distribution: (2 variables for example)

$P(v\_1,v\_2) = P(v\_1)P(v\_2)$



$P(v\_1,v\_2) = P(v\_1)P(v\_2|v\_1)$



$P(v\_1,v\_2) = P(v\_2)\ P(v\_1|v\_2)$

V_1	V_2
0	1
0	1
1	0
1	0
0	1
0	0
1	0

Which One is Better? More Realistic?

Score-based Learning:

- All these papers proposed score-based algorithms.
- Find the highest-scoring network structure
  - Optimal algorithms [2]
  - Approximation algorithms [1, 3]
- All of these are expressed as a sum over the individual variables, e.g.

BDeu	$\sum_{i}^{n}\sum_{j}^{q_i}\log\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}+\sum_{k}^{r_i}\log\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}$
MDL	$\sum_{i}^{n}-LL(X_i\|PA_i)+\frac{\log N}{2}(r_i-1)q_i$
fNML	$\sum_{i}^{n}\sum_{j}^{q_i}\sum_{k}^{r_i}-N_{ijk}\log\frac{N_{ijk}}{N_{ij}}-C(r_i,N_{ij})$

$$Score(G)=\sum_{i}^{n}Score(X_i|PA_i)$$

Score-based Learning

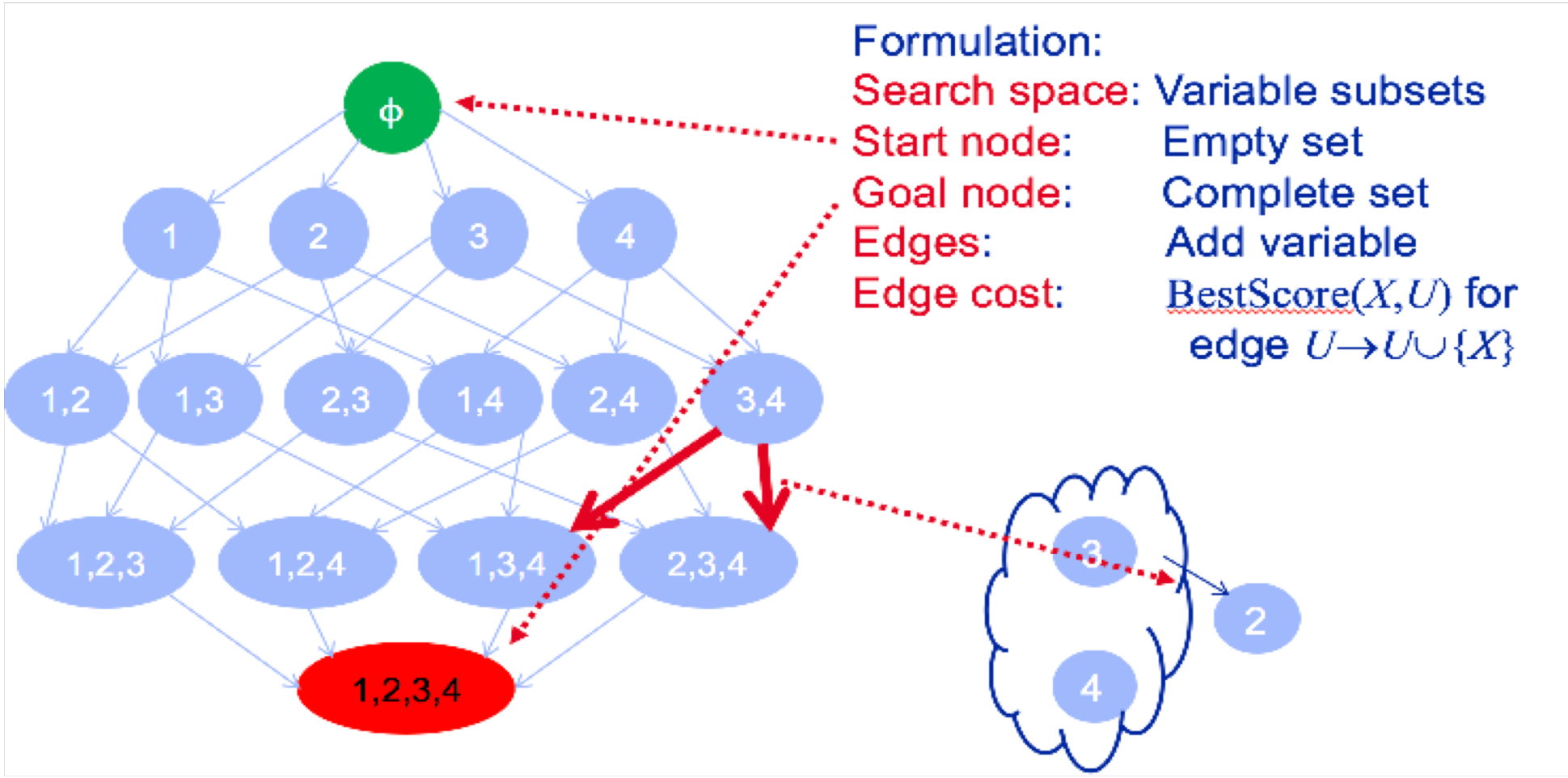
- Find the best parent with best score
- *BestScore (X,Parentsets) = min Score(X|PA$_X$)*

  e.g. *BestScore (A,{D,C,E,ϕ}) = min Score(A|PA$_A$)*

A	**{D} 9.6**	**{C} 9.9**	**{E} 10.0**	**{} 15.4**	
B	**{C,D} 12.1**	**{C} 12.2**	**{E} 12.3**	**{} 14.1**	
C	**{E} 3.6**	**{D} 5.2**	**{A,B} 10.9**	**{A} 11.4**	**{} 17.0**
D	**{E} 3.6**	**{C} 5.2**	**{A,B} 10.9**	**{A} 11.4**	**{} 17.0**
E	**{D} 3.7**	**{A} 4.2**	**{A,B} 11.2**	**{C} 11.6**	**{} 17.0**

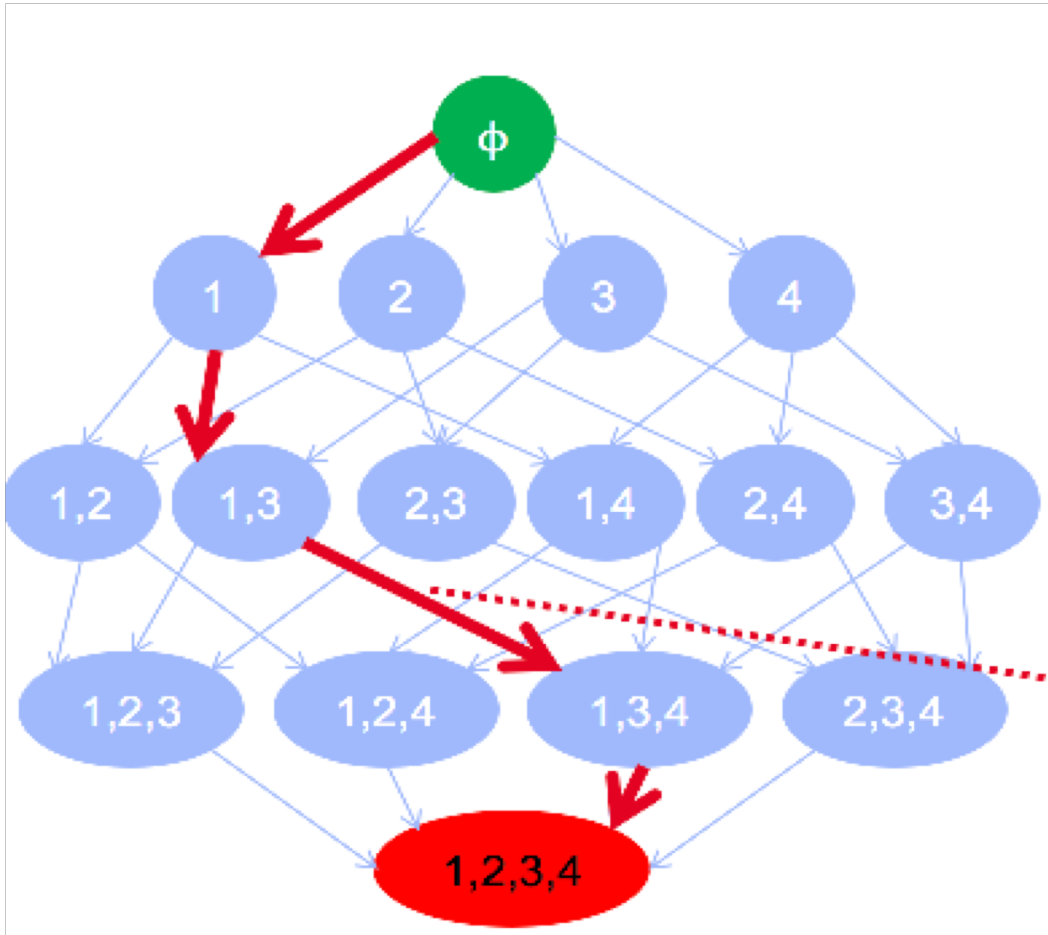
Table from '*Machine Learning of Bayesian networks using constraint programming*'

Graph search



[Yuan, Malone, Wu, IJCAI-11]

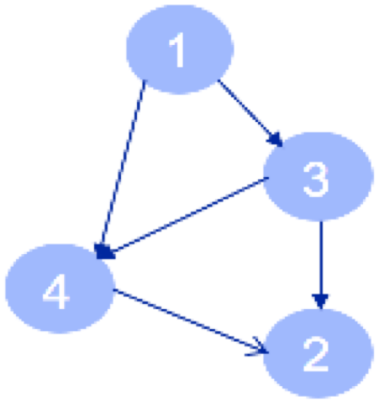
Graph search



Formulation:
Search space: Variable subsets
Start node: Empty set
Goal node: Complete set
Edges: Add variable
Edge cost: $\text{BestScore}(X,U)$ for edge $U \rightarrow U \cup \{X\}$

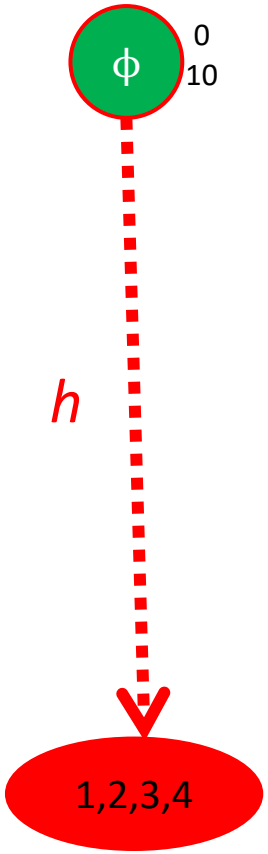
Task: find the shortest path between start and goal nodes

1,3,4,2



[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



Expand the nodes in the order of quality: *f=g+h*

g(*U*) = *Score*(*U*)

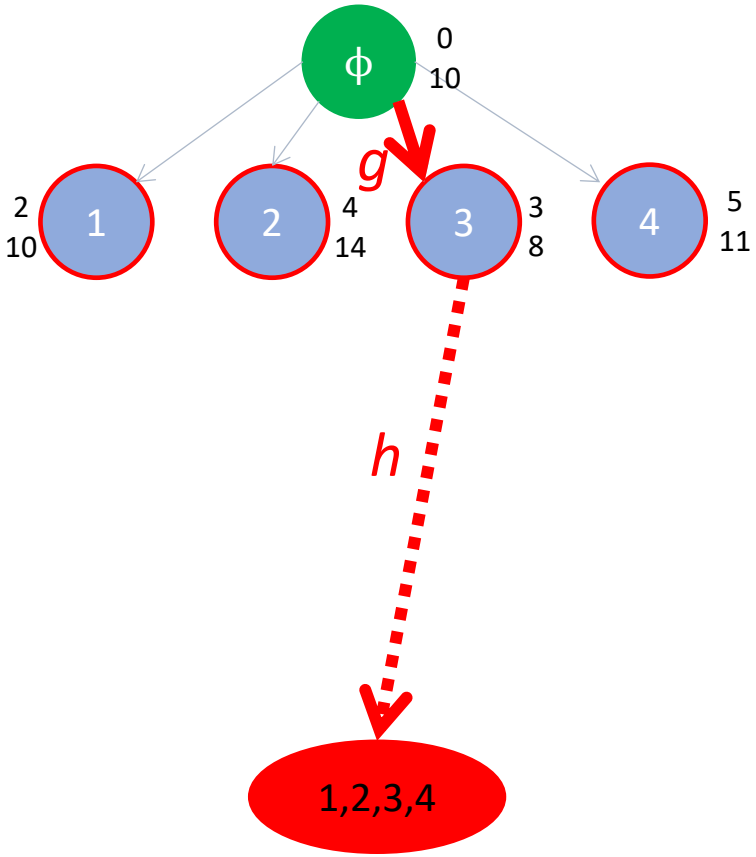
h(*U*) = *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes (In the openlist)
No outline:	closed nodes (In the closelist)

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



Expand the nodes in the order of quality: *f=g+h*

g(*U*) = *Score*(*U*)

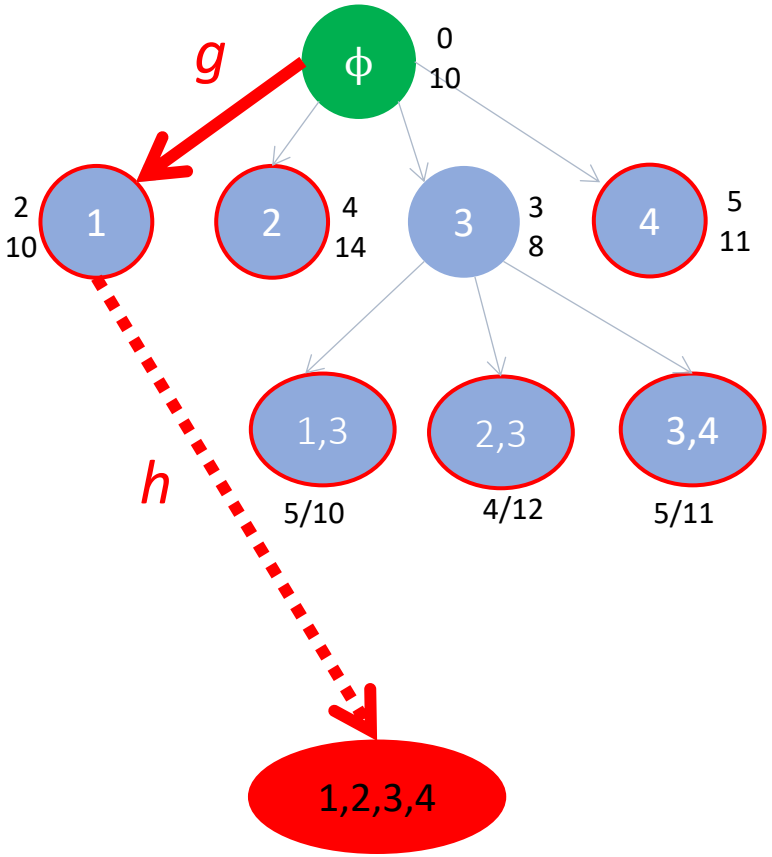
h(*U*) = *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes (In the openlist)
No outline:	closed nodes (In the closelist)

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



Expand the nodes in the order of quality: *f=g+h*

g(*U*) = *Score*(*U*)

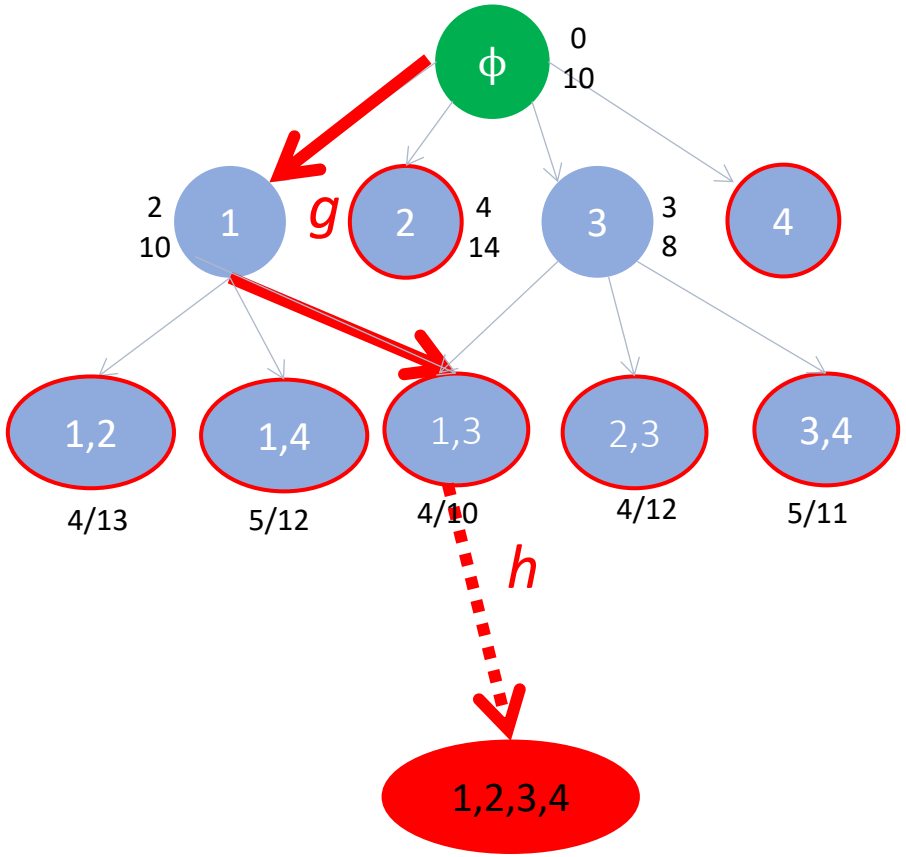
h(*U*) = *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes (In the openlist)
No outline:	closed nodes (In the closelist)

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



Expand the nodes in the order of quality: *f=g+h*

g(*U*) = *Score*(*U*)

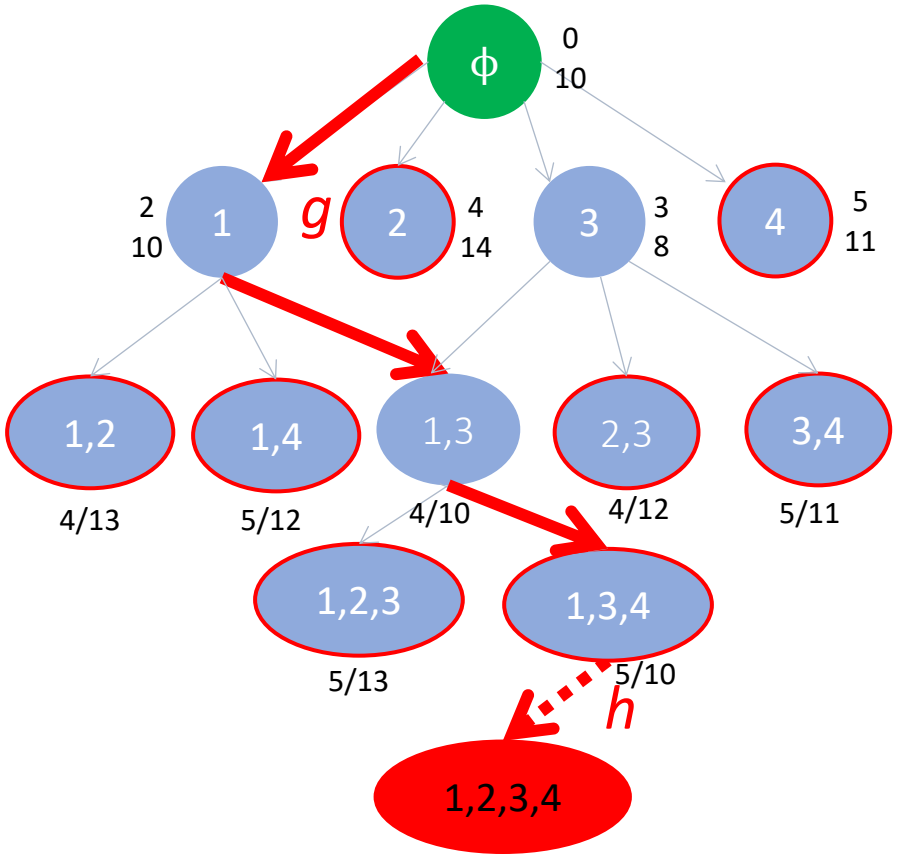
h(*U*) = *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes (In the openlist)
No outline:	closed nodes (In the closelist)

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



Expand the nodes in the order of quality: *f=g+h*

g(*U*) = *Score*(*U*)

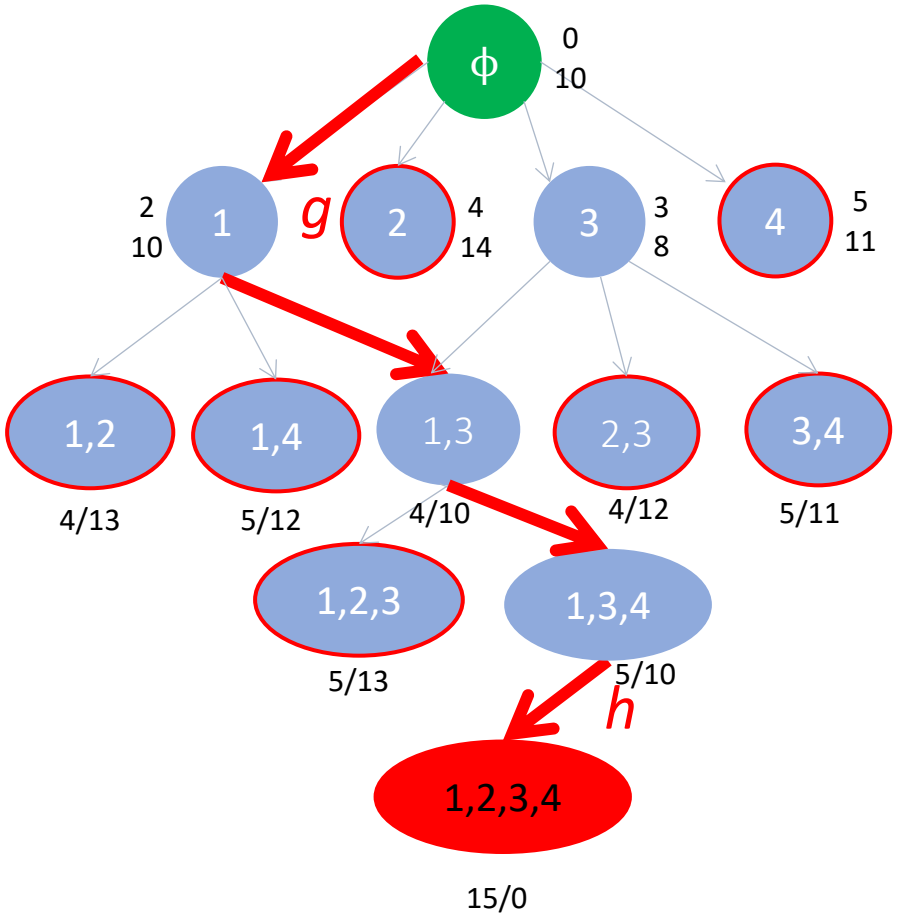
h(*U*) = *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes (In the openlist)
No outline:	closed nodes (In the closelist)

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



Expand the nodes in the order of quality: *f=g+h*

g(*U*) = *Score*(*U*)

h(*U*) = *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes (In the openlist)
No outline:	closed nodes (In the closelist)

[Yuan, Malone, Wu, IJCAI-11]

Learning Optimal Bayesian Networks Using A* Search

- Strength:
  - Can solve variables with large possible parent sets.
  - Find optimal score (best Bayesian Network)

- Weakness
  - Out of memory when deal with more variables (60) because Queue is full.

- Feature Direction to solve weakness:
  - Pruning technique to reduce paths
  - More tight heuristic bounds.

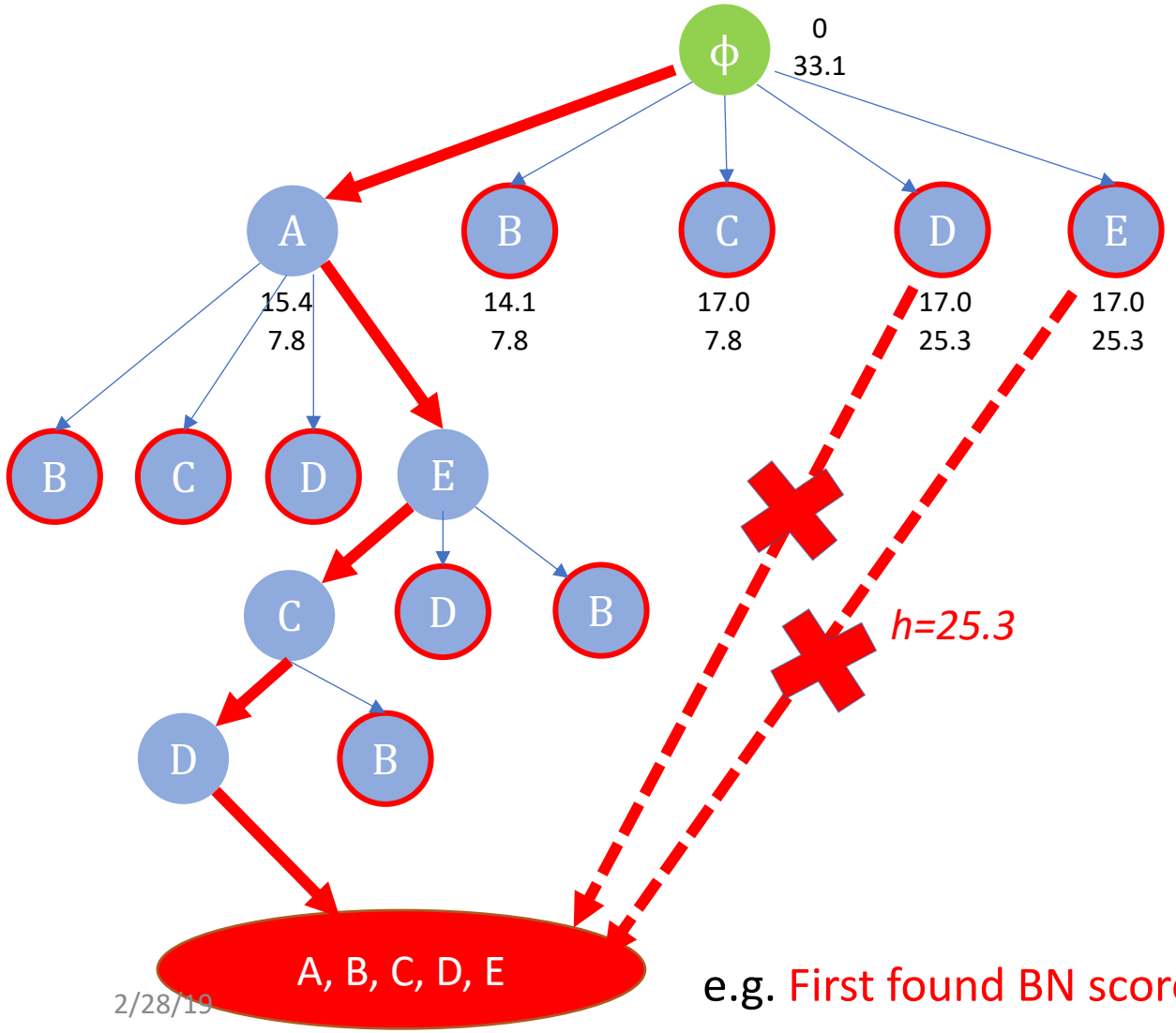
DFBnB Alogrithm

- In Peter and Hella-Franziska paper, they present a constraint-based depth-first BnB approach.

- The lower bound is based on the lower bound proposed by Fan and Yuan.

- The initial upper bound is based on the local search algorithm propsed by Teyssier and Koller. As better solutions are found during the search the upper bound is updated.

DFBnB using heuristic lower bound



For D, E the estimated score:
$g+h$= 17+25.3 = 42.3

In the future searching, the D,E paths' scores are definitely no smaller than 42.3, also no smaller than first found BN score.

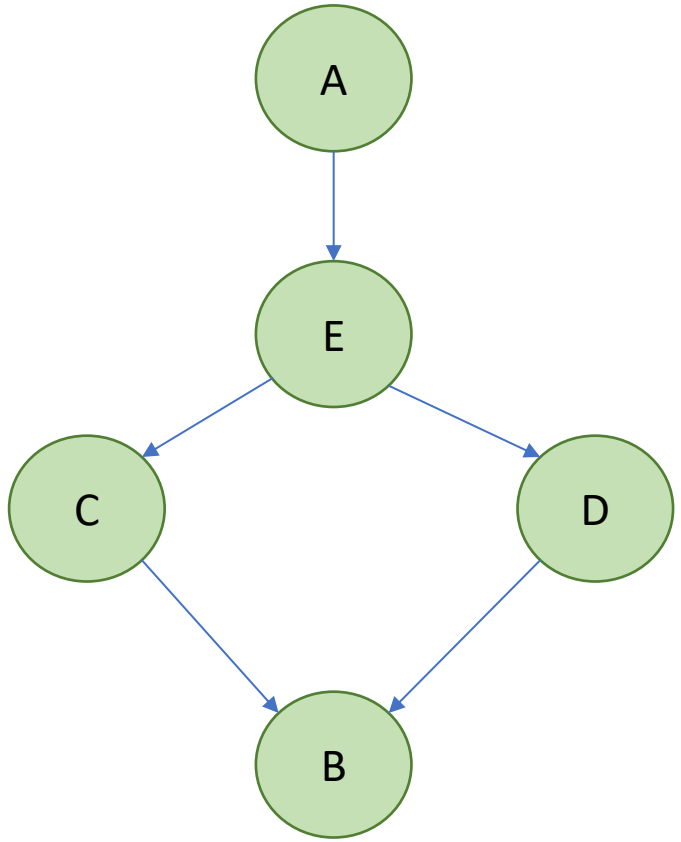
The paths can be pruned.

e.g. First found BN score = 38.9

Constraint Programming Approach

- Symmetry-breaking constraints
- Dominance constraints
- Acyclic constraint

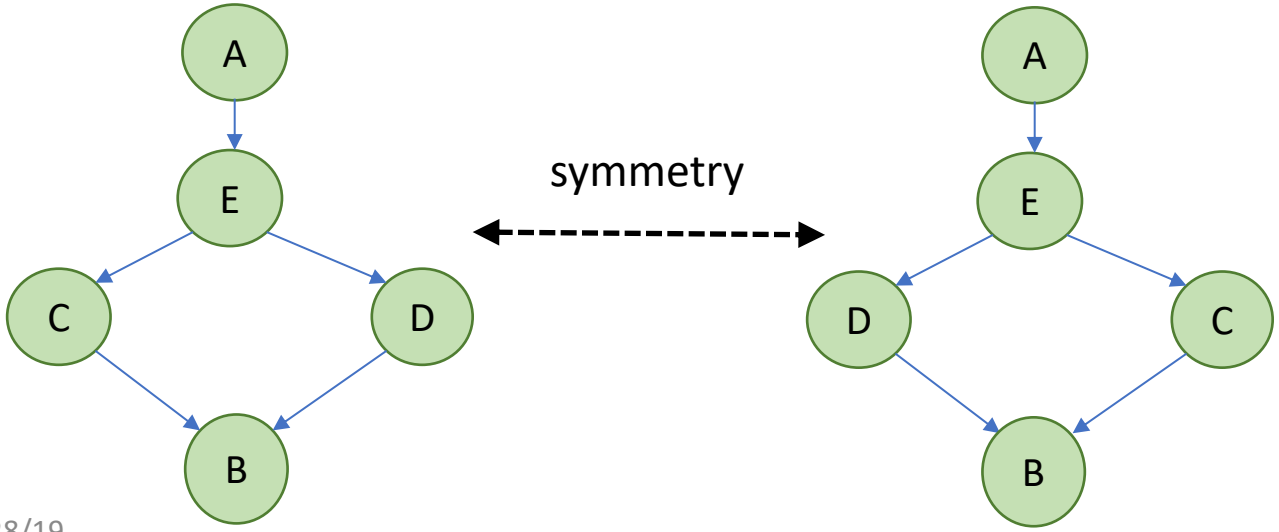
A	{D} 9.6	{C} 9.9	{E} 10.0	{} 15.4	
B	{C,D} 12.1	{C} 12.2	{E} 12.3	{} 14.1	
C	{E} 3.6	{D} 5.2	{A,B} 10.9	{A} 11.4	{} 17.0
D	{E} 3.6	{C} 5.2	{A,B} 10.9	{A} 11.4	{} 17.0
E	{D} 3.7	{A} 4.2	{A,B} 11.2	{C} 11.6	{} 17.0



Score: 38.9

Symmetry-breaking

A	**{D} 9.6**	**{C} 9.9**	**{E} 10.0**	**{} 15.4**	
B	**{C,D} 12.1**	**{C} 12.2**	**{E} 12.3**	**{} 14.1**	
C	**{E} 3.6**	**{D} 5.2**	**{A,B} 10.9**	**{A} 11.4**	**{} 17.0**
D	**{E} 3.6**	**{C} 5.2**	**{A,B} 10.9**	**{A} 11.4**	**{} 17.0**
E	**{D} 3.7**	**{A} 4.2**	**{A,B} 11.2**	**{C} 11.6**	**{} 17.0**



PA(C) = {E,D, AB, A, ϕ}

[C/D]PA(C) = {E,~~D~~, AB, A, ϕ}

C

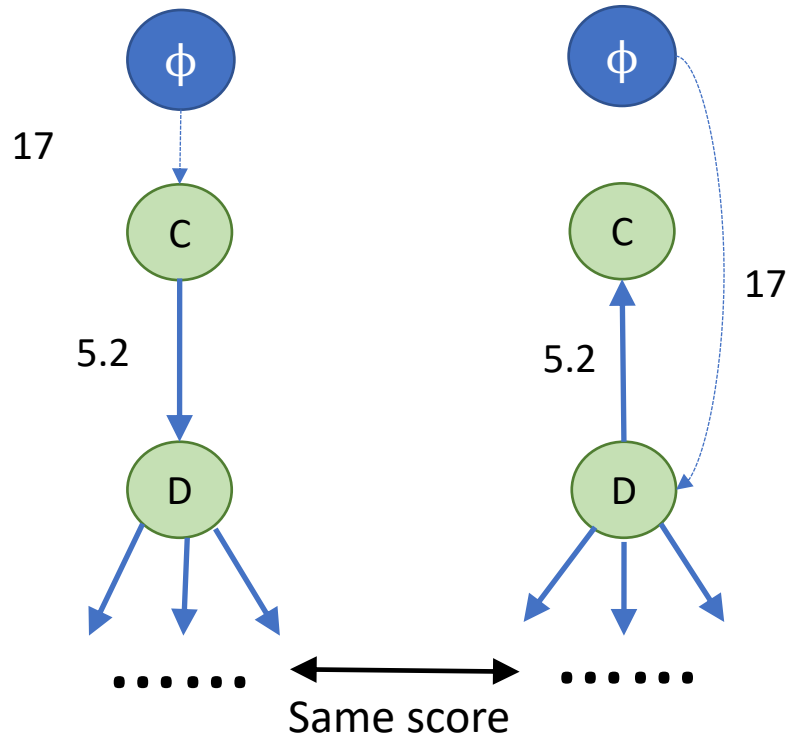
={E,C, AB, A, ϕ}
=PA(D)

We can see a symmetry here, then later we just need expand one pattern pruning the symmetric pattern.

Using the lexicographic order to keep only one BN.

Symmetry-breaking

- I-equivalent: Two DAGs encode the same set of conditional independence assumptions.



An edge x -> y in a Bayesian Network is a covered edge if $PA(y) = PA(x) \cup \{x\}$

e.g. we already confirm a local network with A -> E

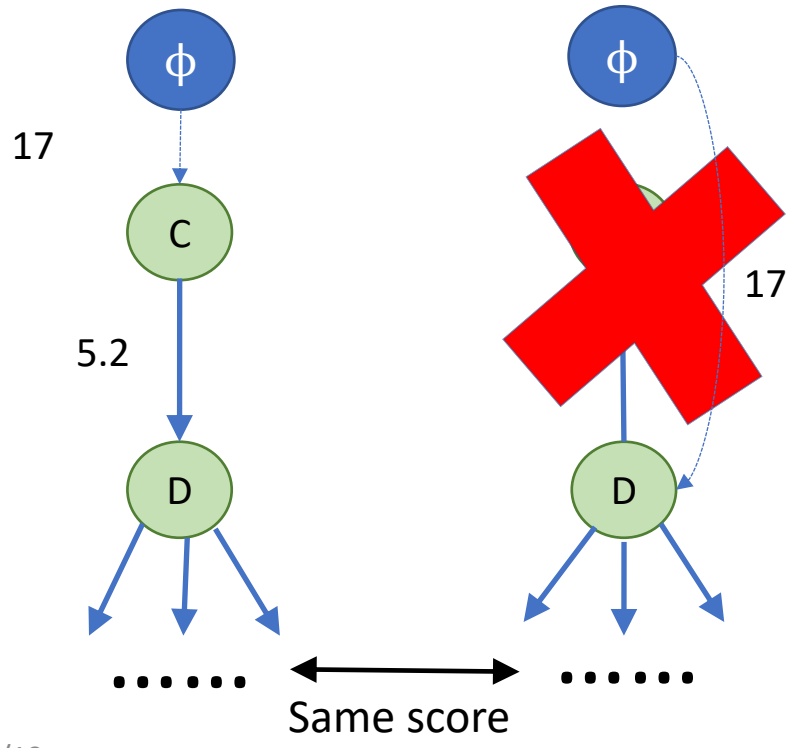
$PA(C) = \{\}$

$PA(D) = \{\} \cup \{C\}$

The Bayesian network with the edge reversed to be D -> C is an I-equivalent Bayesian network.

Symmetry-breaking

- I-equivalent: Two DAGs encode the same set of conditional independence assumptions.



An edge x -> y in a Bayesian Network is a covered edge if $PA(y) = PA(x) \cup \{x\}$

e.g. we already confirm a local network with A -> E

$PA(C) = \{\}$

$PA(D) = \{\} \cup \{C\}$

The Bayesian network with the edge reversed to be D -> C is an I-equivalent Bayesian network.

Machine learning of Bayesian networks using constraint programming

- Strength:
  - An improvement on BnB Depth-first search, saving time and space
  - Find optimal score (best Bayesian Network)

- Weakness
  - Have the same scalability problem with A* search. Out of time when the variable size is large.
  - It is a little bit tricky, because it randomly order the variables in the beginning.

A Comparison

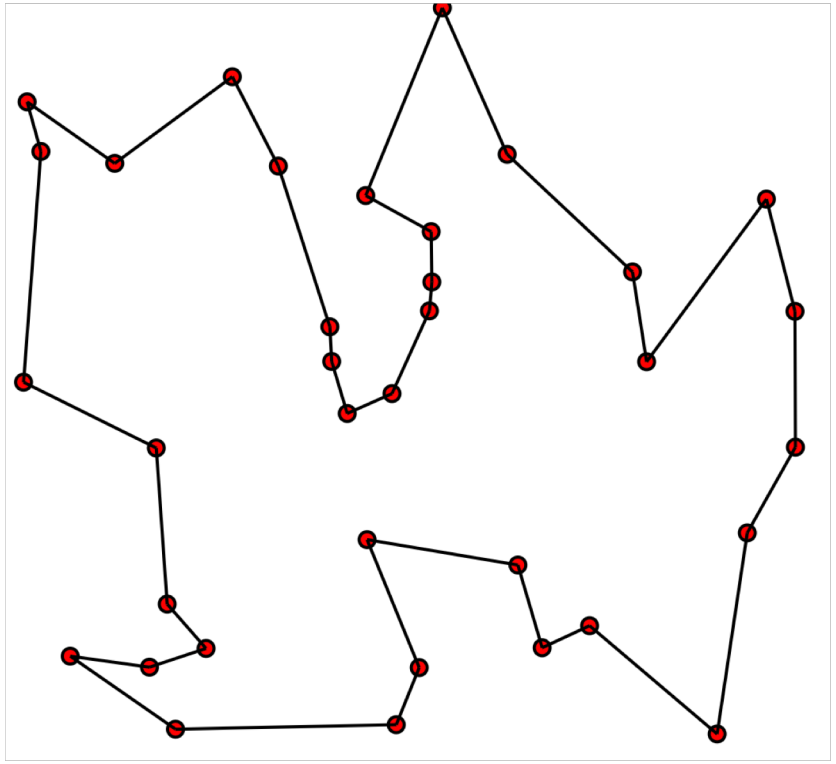
			BDeu				BIC			
Benchmark	n	N	d	GOBN. v1.4.1	A* v2015	CPBayes v1.0	d	GOBN. v1.4.1	A* v2015	CPBayes v1.0
shuttle	10	58,000	812	58.5	0.0	0.0	264	2.8	0.1	0.0
adult	15	32,561	768	1.4	0.1	0.0	547	0.7	0.1	0.0
letter	17	20,000	18,841	5,060.8	1.3	1.4	4,443	72.5	0.6	0.2
voting	17	435	1,940	16.8	0.3	0.1	1,848	11.6	0.4	0.1
zoo	17	101	2,855	177.7	0.5	0.2	554	0.9	0.4	0.1
tumour	18	339	274	1.5	0.9	0.2	219	0.4	0.9	0.2
lympho	19	148	345	1.7	2.1	0.5	143	0.5	1.0	0.2
vehicle	19	846	3,121	90.4	2.4	0.7	763	4.4	2.1	0.5
hepatitis	20	155	501	2.1	4.9	1.1	266	1.7	4.8	1.0
segment	20	2,310	6,491	2,486.5	3.3	1.3	1,053	13.2	2.4	0.5
mushroom	23	8,124	438,185	OT	255.5	561.8	13,025	82,736.2	34.4	7.7
autos	26	159	25,238	OT	918.3	464.2	2,391	108.0	316.3	50.8
insurance	27	1,000	792	2.8	583.9	107.0	506	2.1	824.3	103.7
horse colic	28	300	490	2.7	15.0	3.4	490	3.2	6.8	1.2
steel	28	1,941	113,118	OT	902.9	21,547.0	93,026	OT	550.8	4,447.6
flag	29	194	1,324	28.0	49.4	39.9	741	7.7	12.1	2.6
wdbc	31	569	13,473	2,055.6	OM	11,031.6	14,613	1,773.7	1,330.8	1,460.5
water	32	1,000					159	0.3	1.6	0.6
mildew	35	1,000	166	0.3	7.6	1.5	126	0.2	3.6	0.6
soybean	36	266					5,926	789.5	1,114.1	147.8
alarm	37	1,000					672	1.8	43.2	8.4
bands	39	277					892	15.2	4.5	2.0
spectf	45	267					610	8.4	401.7	11.2
sponge	45	76					618	4.1	793.5	13.2
barley	48	1,000					244	0.4	1.5	3.4
hailfinder	56	100					167	0.1	9.9	1.5
hailfinder	56	500					418	0.5	OM	9.3
lung cancer	57	32					292	2.0	OM	10.5
carpo	60	100					423	1.6	OM	253.6
carpo	60	500					847	6.9	OM	OT

Combinatorial Optimization Problems

- Combinatorial Optimization has numerous applications:
  - Transportation, communications and scheduling
  - Genetics, etc…
- Many are NP-hard problems on graph
  - Brute-force solution: O($n!$)
  - DP algorithms: O($n^2 2^n$)
- Traveling Salesman Problem(TSP)
- Knapsack Problem
- Minimum Vertex Cover
- Bayesian Network Learning

TSP

- Given a graph, search the space of permutations to find an optimal sequence of nodes with minimal total edge weights.



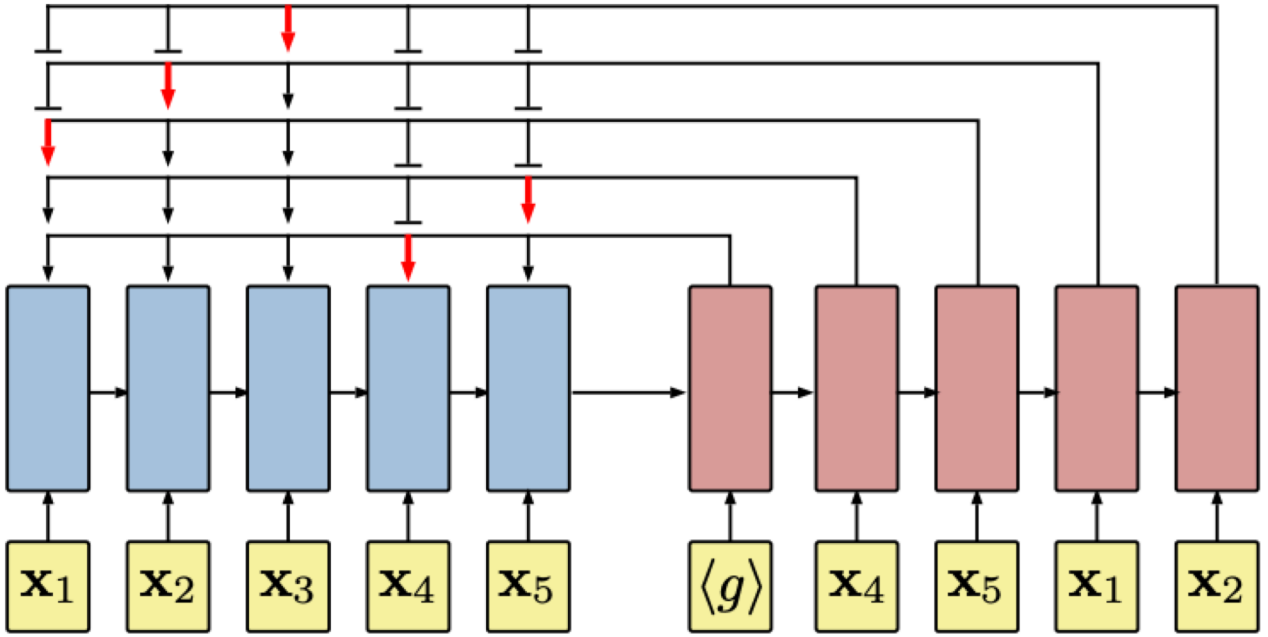
- Focus on the 2D Euclidean TSP.
- Input graph sequence:

$$s = \{x_i\}_{i=1}^{n}$$

- Permutation: π
- Length of a tour:

$$L(\pi \mid s) = \left\|\mathbf{x}_{\pi(n)} - \mathbf{x}_{\pi(1)}\right\|_2 + \sum_{i=1}^{n-1} \left\|\mathbf{x}_{\pi(i)} - \mathbf{x}_{\pi(i+1)}\right\|_2$$

Pointer Network for TSP



- By *'NEURAL COMBINATORIAL OPTIMIZATION WITH REINFORCEMENT LEARNING'*
- Applied on TSP and Knapsack

- Input: 2-d coordinate for each variable
- Output: The ordering or input variables

Supervised learning?

- Parameterize probability distribution over solutions with a NN theta:
  - $p_\theta(solution|problem)$
- Train a model to maximize the likelihood of the correct solutions:
  - $p_\theta(best\ solution|problem)$
- Enough training examples?
- Have access to optimal labels?

Reinforcement Learning

- Parameterize probability distribution over solutions with a NN theta:
  - $p_\theta(solution|problem)$
- Train NN theta with RL to minimize expected cost of solutions:
  - $E_{sol\sim p_\theta(.|problem)}[cost(sol|problem)]$
- It's easy to measure cost of any proposed solutions.
- Don't need to train with only optimal tour, but any tours.

Neural Combinatorial Optimization With Reinforcement Learning

- Strength:
  - Can solve large dataset
  - Good for unsupervised data using reinforcement learning
- Weakness:
  - No standard to check the result is good or not.
- Feature Direction:
  - Apply to Bayesian Networks.

Thanks!