Solutions homework #2

Carlos Cotrini
October 27, 2017

Probabilistic foundations of artificial intelligence
1. Bayesian Networks: d-separation

- Solutions task 1

B indep. I given F?
Principle of d-separation

- Given a set of observed variables, if there is no active trail between two variables, then they are independent.

B indep from C | A

Unclear if B indep from C | A, D
1. Bayesian Networks: d-separation

• Recap on active trails. Case 1. (Mountain)
1. Bayesian Networks: d-separation

• Recap on active trails. Case 2a. (Downhill)
1. Bayesian Networks: d-separation

- Recap on active trails. Case 2b. (Uphill)
1. Bayesian Networks: d-separation

- Recap on active trails. Case 3. (Valley)
1. Bayesian Networks: d-separation

- Recap on active trails. Case 3.
1. Bayesian Networks: d-separation

- Recap on active trails. Case 3.
1. Bayesian Networks: d-separation

- Recap on active trails. Case 3.
1. Bayesian Networks: d-separation

- Solutions task 1
1. Bayesian Networks: d-separation

- Solutions task 1
1. Bayesian Networks: d-separation

• Solutions task 1
1. Bayesian Networks: d-separation

- Solutions task 1

A indep. F?
1. Bayesian Networks: d-separation

- Solutions task 1

A indep. F?
1. Bayesian Networks: d-separation

- Solutions task 1
1. Bayesian Networks: d-separation

- Solutions task 1

A indep. G?
1. Bayesian Networks: d-separation

- Solutions task 1

A indep. G?
1. Bayesian Networks: d-separation

• Solutions task 1

A indep. G?

YES!
1. Bayesian Networks: d-separation

- Solutions task 1

B indep. I given F?
1. Bayesian Networks: d-separation

• Solutions task 1

B indep. I given F?
1. Bayesian Networks: d-separation

- Solutions task 1

B indep. I given F?

INCONCLUSIVE
1. Bayesian Networks: d-separation

• Solutions task 1

D indep. J given G, H?
1. Bayesian Networks: d-separation

• Solutions task 1

D indep. J given G, H?
1. Bayesian Networks: d-separation

- Solutions task 1

D indep. J given G, H?
1. Bayesian Networks: d-separation

- Solutions task 1

D indep. J given G, H?

YES!
1. Bayesian Networks: d-separation

- Solutions task 1

I indep. B given H?
1. Bayesian Networks: d-separation

- Solutions task 1

I indep. B given H?

YES!
1. Bayesian Networks: d-separation

• Solutions task 1

D indep. J?
1. Bayesian Networks: d-separation

• Solutions task 1

D indep. J?
1. Bayesian Networks: d-separation

- Solutions task 1
1. Bayesian Networks: d-separation

- Solutions task 1

I indep. C given H, F?
Be careful with the direction of the arrows!

B indep from C ?
Be careful with the direction of the arrows!

A indep from D ?
2. Variable elimination

• Compute $P(J = j)$

1. Joint probability implied by BN structure: $P(A, \ldots, J) = P(A) \cdot P(B|A) \cdot P(C|B) \cdot P(G) \cdot P(E|G) \cdot P(D,C,E) \cdot P(F|B,E,D) \cdot P(I|G) \cdot P(H|G,I) \cdot P(J|I)$
2. Variable elimination

- Compute \( P(J = j) \)

2. Eliminating \( A \):

\[
P(B, \ldots, J) = \sum_{a} P(a) \cdot P(B|a) \cdot P(C|B) \cdot P(G) \cdot P(E|G) \cdot P(D|C, E) \cdot P(F|B, E, D) \cdot P(I|G) \cdot P(H|G, I) \cdot P(J|I)
\]

\[
= P(C|B) \cdot P(G) \cdot P(E|G) \cdot P(D|C, E) \cdot P(F|B, E, D) \cdot P(I|G) \cdot P(H|G, I) \cdot P(J|I) \cdot \sum_{a} P(a)P(B|a)
\]

\[
= P(C|B) \cdot P(G) \cdot P(E|G) \cdot P(D|C, E) \cdot P(F|B, E, D) \cdot P(I|G) \cdot P(H|G, I) \cdot P(J|I) \cdot g_1(B)
\]
2. Variable elimination

• Compute $P(J = j)$

3. Eliminating $B$: $P(C, \ldots, J) = P(G) \cdot P(E|G) \cdot P(D|C, E) \cdot P(I|G) \cdot P(H|G, I) \cdot P(J|I) \cdot g_2(C, F, E, D)$
2. Variable elimination

- Compute $P(J = j)$

4. Eliminating $C$: $P(D, \ldots, J) = P(G) \cdot P(E \mid G) \cdot P(I \mid G) \cdot P(H \mid G, I) \cdot P(J \mid I) \cdot g_3(F, E, D)$

5. Eliminating $D$: $P(E, \ldots, J) = P(G) \cdot P(E \mid G) \cdot P(I \mid G) \cdot P(H \mid G, I) \cdot P(J \mid I) \cdot g_4(F, E)$
2. Variable elimination

- Compute $P(J = j)$

4. Eliminating $C$: $P(D, \ldots, J) = P(G) \cdot P(E|G) \cdot P(I|G) \cdot P(H|G, I) \cdot P(J|I) \cdot g_3(F, E, D)$

5. Eliminating $D$: $P(E, \ldots, J) = P(G) \cdot P(E|G) \cdot P(I|G) \cdot P(H|G, I) \cdot P(J|I) \cdot g_4(F, E)$
2. Variable elimination

- Compute $P(J = j)$

6. Eliminating $E$: $P(F, \ldots, J) = P(G) \cdot P(I|G) \cdot P(H|G, I) \cdot P(J|I) \cdot g_5(F, G)$

7. Eliminating $F$: $P(G, \ldots, J) = P(G) \cdot P(I|G) \cdot P(H|G, I) \cdot P(J|I) \cdot g_6(G)$

8. Eliminating $G$: $P(H, I, J) = P(J|I) \cdot g_7(I, H)$
2. Variable elimination

• Compute $P(J = j)$

9. Eliminating $H$: $P(I, J) = P(J|I) \cdot g_8(I)$

10. Eliminating $I$: $P(J) = g_9(J)$
3. An algorithm for d-separation

• Given a Bayesian Network, a variable X, and a set of variables $E$ with observed values $e$, compute all variables $Z$ that are indep. from X given $E$. 
3. Algorithm for d-separation
3. Algorithm for d-separation

D-reachable: if it can be reached with an active trail from X
3. Algorithm for d-separation
3. Algorithm for d-separation
3. Algorithm for d-separation

Key insight: Subtrails of active trails are also active!
3. Algorithm for d-separation

Key insight: Subtrails of active trails are also active!

Compute d-reachable variables with a depth-first (or breadth-first) search.
3. Algorithm for d-separation

toVisit = [X]

d-reachable = {}
3. Algorithm for d-separation

toVisit = []
d-reachable = \{X\}
3. Algorithm for d-separation

toVisit = [A]
d-reachable = {X}
3. Algorithm for d-separation

toVisit = \{S, A\}
d-reachable = \{X\}
3. Algorithm for d-separation

toVisit = [I, S, A]
d-reachable = \{X\}
3. Algorithm for d-separation

toVisit = [S, A]
d-reachable = {X}
3. Algorithm for d-separation

toVisit = [S, A]
d-reachable = \{X\}
3. Algorithm for d-separation

toVisit = [A]
d-reachable = \{X, S\}
3. Algorithm for d-separation

toVisit = [W, A]
d-reachable = {X, S}
3. Algorithm for d-separation

toVisit = [B, W, A]
d-reachable = {X, S}
3. Algorithm for d-separation

toVisit = [U, B, W, A]
d-reachable = \{X, S\}
3. Algorithm for d-separation

toVisit = [W, B, A]
d-reachable = {X, S, U}
3. Algorithm for d-separation

toVisit = [B, A]
d-reachable = {X, S, U}
3. Algorithm for d-separation

toVisit = [B, A]
d-reachable = {X, S, U}
3. Algorithm for d-separation

toVisit = [A]
d-reachable = {X, S, U}
3. Algorithm for d-separation

toVisit = [R, A]
d-reachable = {X, S, U}
3. Algorithm for d-separation

toVisit = [A]
d-reachable = {X, S, U, R}
3. Algorithm for d-separation

toVisit = []
d-reachable = \{X, S, U, R, A\}
3. Algorithm for d-separation

toVisit = []
d-reachable = {X, S, U, R, A}
3. Algorithm for d-separation

toVisit = [K]
d-reachable = {X, S, U, R, A}
3. Algorithm for d-separation

toVisit = []
d-reachable = \{X, S, U, R, A, K\}
3. Algorithm for d-separation

```python
def get_reachable(X, E):
    toVisit = [X], visited = {}
    while toVisit != []:
        V = toVisit.pop()
        "visit V"
        add V to visited
        for Y an unvisited neighbor of V:
            if .... then push Y
```

Loop’s invariant: Z is in toVisit iff there is an active trail from X to Z
3. Algorithm for \( d \)-separation

```python
def get_reachable(X, E):
    toVisit = [X], visited = {}, reachable = {}
    while toVisit != []:
        V = toVisit.pop()
        if V is not obs then add V to reachable
        add V to visited
        for Y an unvisited neighbor of V:
            if .... then push Y
```

Loop’s invariant: \( Z \) is in toVisit iff there is an active trail from \( X \) to \( Z \)
What neighbors to append?

• Let V be the node currently visited and Y be an unvisited neighbor.
• By our invariant, there is an active trail $t$ from $X$ to $V$.
• What we need to decide if $t.append(Y)$ active...
What neighbors to append?

- Let $V$ be the node currently visited and $Y$ be an unvisited neighbor.
- By our invariant, there is an active trail $t$ from $X$ to $V$.
- What we need to decide if $t$.append($Y$) active...
  - $V -$ --> $Y$ or $V$ <--$Y$?
What neighbors to append?

- Let V be the node currently visited and Y be an unvisited neighbor.
- By our invariant, there is an active trail $t$ from X to V.
- What we need to decide if $t$.append(Y) active...
  - $V \rightarrow Y$ or $V \leftarrow Y$?

  - Is V or any of its descendants observed?
What neighbors to append?

- Let V be the node currently visited and Y be an unvisited neighbor.
- By our invariant, there is an active trail \( t \) from X to V.
- What we need to decide if \( t.append(Y) \) active...
  - \( V \rightarrow Y \) or \( V \leftarrow Y \)?
  - Is \( V \) or any of its descendants observed?
  - \( W \rightarrow V \) or \( W \leftarrow V \)? (\( W \) is \( V \)'s predecessor in \( t \))
What neighbors to append?

- Let $V$ be the node currently visited and $Y$ be an unvisited neighbor.
- By our invariant, there is an active trail $t$ from $X$ to $V$.
- What we need to decide if $t$.append($Y$) active...
  - $V \rightarrow Y$ or $V \leftarrow Y$?
    
    **Easy to check**
    
    - Is $V$ or any of its descendants observed?
    
    - $W \rightarrow V$ or $W \leftarrow V$? ($W$ is $V$’s predecessor in $t$)
What neighbors to append?

- Let V be the node currently visited and Y be an unvisited neighbor.
- By our invariant, there is an active trail $t$ from X to V.
- What we need to decide if $t$.append(Y) active...
  - V - - > Y or V <- - Y?

  Easy to check
  - Is V or any of its descendants observed?

  Compute in advance the ancestors of all observed variables
  - W - - > V or W <- - V? (W is V’s predecessor in $t$)
What neighbors to append?

- Let V be the node currently visited and Y be an unvisited neighbor.
- By our invariant, there is an active trail t from X to V.
- What we need to decide if t.append(Y) active...
  - V - -> Y or V <- - Y?

  **Easy to check**
  - Is V or any of its descendants observed?

  **Compute in advance the ancestors of all observed variables**
  - W - -> V or W <- - V? (W is V’s predecessor in t)

  **Keep track how you reached V**
3. Algorithm for d-separation

```python
def get_reachable(X, E):
    toVisit = [X], visited = {}, reachable = {}

    while toVisit != []:
        V = toVisit.pop()
        if V is not obs then add V to reachable
        add V to visited

        for Y an unvisited neighbor of V:
            if …. then push Y
```

Loop’s invariant: Z is in toVisit iff there is an active trail from X to Z
3. Algorithm for d-separation

```python
def get_reachable(X, E):
    toVisit = [X], visited = {}, reachable = {}
    ancestors_E = computeAncestors(E)
    while toVisit != []:
        V = toVisit.pop()
        if V is not obs then add V to reachable
        add V to visited
        for Y an unvisited neighbor of V:
            if .... then push Y

Loop's invariant: Z is in toVisit iff there is an active trail from X to Z
```
3. Algorithm for d-separation

```python
def get_reachable(X, E):
    toVisit = [ (<--.,X) ], visited = {}, reachable = {}
    ancestors_E = computeAncestors(E)
    while toVisit != []:
        (dir, V) = toVisit.pop()
        if V is not obs then add V to reachable
        add V to visited
        for Y an unvisited neighbor of V:
            if .... then push Y
    Loop’s invariant: Z is in toVisit iff there is an active trail from X to Z
```
for Y an unvisited neighbor of V:

- If dir == <-- :

- If dir == --> :
for Y an unvisited neighbor of V:

• If dir == <-- :
  
• If dir == --> :
  – If V --> Y:
    •
  – If V <-- Y:
    •
for Y an unvisited neighbor of V:

• If dir == <-- :
  - If V is not observed, then push (dir’, Y)

• If dir == --> :
  - If V --> Y:
    • If V is not observed, then push (-->, Y)
  - If V <-- Y:
    • If V is in ancestors_E, then push (<--, Y)