CS221, Spring 2019, PS6 Schedule

Haiyuan Mei (hmei0411@stanford.edu)

Problem 0: Warmup

a. CSP for m buttons and n bulbs

1. There are m variables corresponding to each of the m buttons, denoted as $X_1, \ldots, X_m$
2. Each of the n bulbs corresponds to a constraint, denotes as $b_1, \ldots, b_n$
3. In the factor graph, each X_j will connect to bulbs in T_j. Hence there will be more than 1 variables(buttons) connected to a contraint(bulb).
4. The contraint uses XOR operation on each of the connected variables, meaning the bulb is only turned on when the button is pressed odd number of times.

For example, the following code describes a CSP with 5 buttons $X_1, \ldots, X_5$, and 3 bulbs b_1, b_2, b_3;

```
// Create your own factor graph!
// Call variable(), factor(), query() followed by an inference algorithm.
variable('X1', [0, 1])
variable('X2', [0, 1])
variable('X3', [0, 1])
variable('X4', [0, 1])
variable('X5', [0, 1])

factor('b1', 'S1 S2', function(a, b) {
return a ^ b;
})
factor('b2', 'S2 S3 S4', function(a, b, c) {
return a ^ b ^ c;
})
factor('b3', 'S3 S4 S5', function(a, b, c) {
return a ^ b ^ c;
})
sumVariableElimination()
```

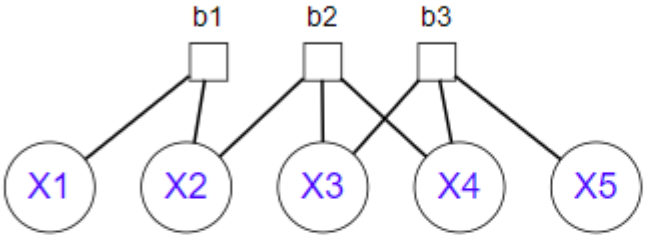
The control of each button is:

$$T_1 = \{b1\}$$
$$T_2 = \{b1, b2\}$$
$$T_3 = \{b2, b3\}$$
$$T_4 = \{b2, b3\}$$
$$T_5 = \{b3\}$$

The following shows the factor graph of the example and the result (Graph generated from by course demo scripts):

Query: $\mathbb{P}(\mathbf{X1}, \mathbf{X2}, \mathbf{X3}, \mathbf{X4}, \mathbf{X5})$

Algorithm: **variable elimination (sum)**



Algorithm done.

Final factor: final

X1	X2	X3	X4	X5	final(X1,X2,X3,X4,X5)	$\mathbb{P}$(X1,X2,X3,X4,X5)
0	1	0	0	1	1	0.25
0	1	1	1	1	1	0.25
1	0	0	1	0	1	0.25
1	0	1	0	0	1	0.25

b. Simple CSP

1. How many consistent assignments? Answer: 2 consistent assignments, they are: [1,0,1] and [0,1,0]
2. There are at most 15 calls to backtrack if no heuristics at all. With assignment order is X_1, X_3, X_2, the call stack looks like following:
   backtrack({\Phi},1,{0,1})
   ....backtrack({0},1,{0,1})
   ....backtrack({0,0},1,{0,1})
   ....backtrack({0,0,0},1,{0,1})
   ....backtrack({0,0,1},1,{0,1})
   ....backtrack({0,1},1,{0,1})
   ....backtrack({0,1,0},1,{0,1})
   ....backtrack({0,1,1},1,{0,1})
   ....backtrack({1},1,{0,1})
   ....backtrack({1,0},1,{0,1})
   ....backtrack({1,0,0},1,{0,1})
   ....backtrack({1,0,1},1,{0,1})
   ....backtrack({1,1},1,{0,1})
   ....backtrack({1,1,0},1,{0,1})
   ....backtrack({1,1,1},1,{0,1})

1. With AC-3 and assignment order is X_1, X_3, X_2 there will be only 7 backtrack calls:
   backtrack({\Phi},1,{0,1})
   ....backtrack({0},1,{0,1})
   ....backtrack({0,0},1,{0,1})
   ....backtrack({0,0,1},1,{0,1})
   ....backtrack({1},1,{0,1})
   ....backtrack({1,1},1,{0,1})
   ....backtrack({1,1,0},1,{0,1})

Problem 2: Handling n-ary factors

a. Convert to unary and binary constraints

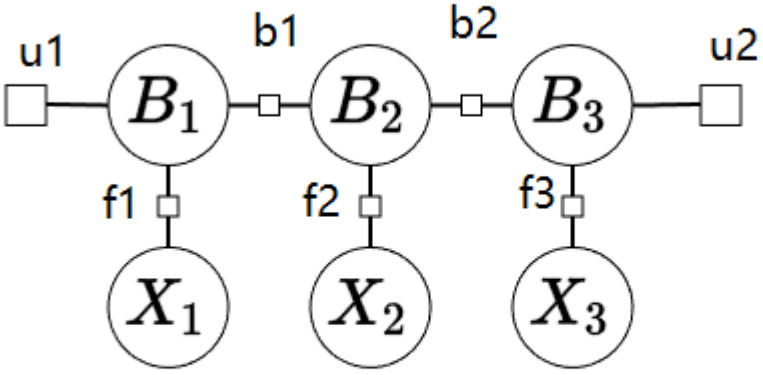
Similar to lecture notes p76, step 1 we add auxiliary variables A_0, A_1, A_2, A_3 such that:

$$[A_0 = 0]$$
$$[A_i = A_{i-1} + X_i]$$
$$[A_3 \leq K]$$

The second step is to eliminate the ternary factor $[A_i = A_{i-1} + X_i]$ by introducing auxiliary variables B, B_1, B_2, B_3 where $[B_i = (A_{i-1}, A_i)]$, we have:

$$[B_1[1] = 0]$$
$$[B_i[2] = B_i[1] + X_i]$$
$$[B_3[2] \leq K]$$
$$[B_{i-1}[2] = B_i[1]]$$

The factor graph looks like:



The above 2 unaries:

$$u1 = [B_1[1] = 0]$$
$$u2 = [B_3[2] \leq K]$$

and 4 binaries:

$$b1 = [B_1[2] = B_2[1]]$$
$$b2 = [B_2[2] = B_3[1]]$$
$$f1 = [B_1[2] = B_1[1] + X_1]$$
$$f2 = [B_2[2] = B_2[1] + X_2]$$
$$f3 = [B_3[2] = B_3[1] + X_3]$$

Problem 3: Course Scheduling

c. create you own profile.txt and run the scheduler

1. The profile file I created is:

```
# Unit limit per quarter. You can ignore this for the first
# few questions in problem 2.
minUnits 3
maxUnits 6

# These are the quarters that I need to fill. It is assumed that
# the quarters are sorted in chronological order.
register Aut2019
register Win2020

# Courses I've already taken
taken CS221
taken CS229

# Courses that I'm requesting
request CS224N in Aut2019 weight 2
request CS231A in Win2020 weight 3
request CS228 in Aut2019
request CS240 in Win2020 weight 2
```

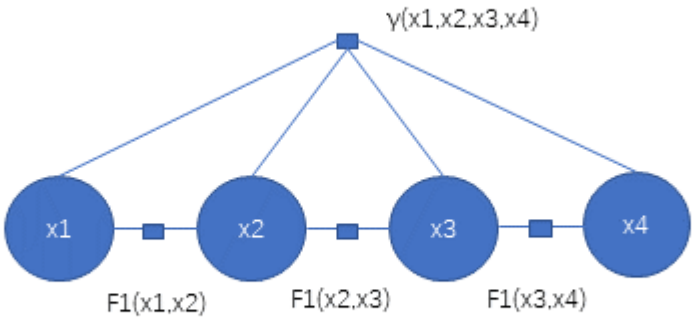
1. Here's the best schedule:

Quarter	Units	Course
Aut2019	4	CS224N
Win2020	3	CS231A
Win2020	3	CS240

Problem 4: Weighted CSPs with notable patterns

a. What would be the worst treewidth?

The worst case is when p has the maximum length n. In this case the treewidth should be n-1. γ is now the n-ary factor, the graph is shown below. Eliminating from left would result in a max arity n-1.



b. An algorithm that would fix the n-ary problem.

Instead of blindly making an n-ary CSP problem above, we could model the CSP problem in a different way. Suppose in the length of p is n_p, for each matching pattern in $x = (x_1, \ldots, x_n)$, we couls just assign each element of the matching array a unary factor with $\gamma(x_i) = \lambda^{\frac{1}{p_n}}$. Then for the whole matching pattern, there are p_n unary factors end up with $\gamma(x_i, \ldots, x_{i+n_p-1}) = \lambda$

For example, still the above X_1, X_2, X_3, X_4, suppose there is a p = {x1,x2}, matching once. The factor graph could be just as following: