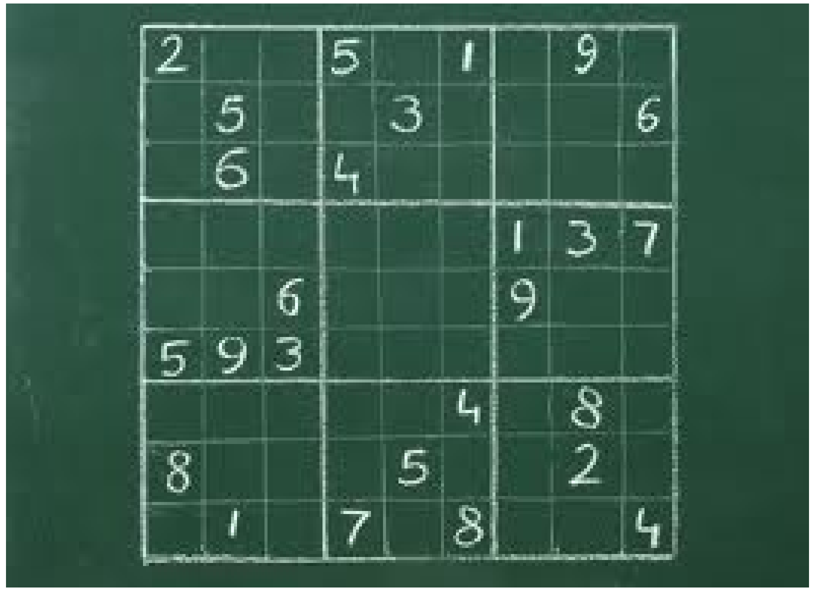
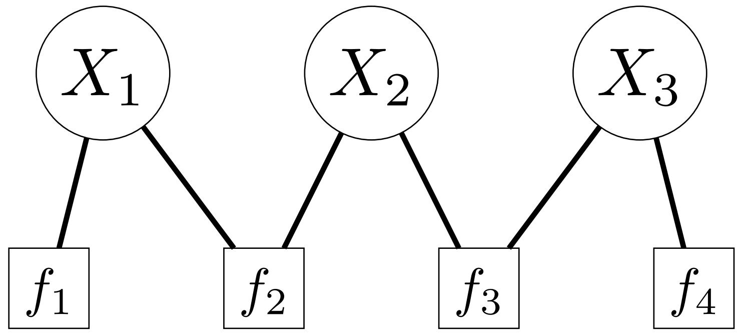
CSPs: local search



- In this module, I will talk about local search, a strategy for approximately computing the maximum weight assignment in a CSP.

Review: CSPs



Definition: factor graph

Variables:

$X = (X_1, \dots, X_n)$, where $X_i \in \text{Domain}_i$

Factors:

$f_1, \dots, f_m$, with each $f_j(X) \geq 0$

Definition: assignment weight

Each **assignment** $x = (x_1, \dots, x_n)$ has a **weight**:

$$\text{Weight}(x) = \prod_{j=1}^{m} f_j(x)$$

Objective:

$$\arg\max_x \text{Weight}(x)$$

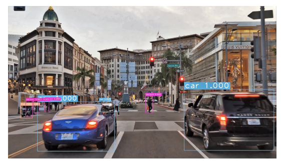
- Recall that a constraint satisfaction problem is defined by a factor graph, where we have a set of variables and a set of factors. Each assignment of values to variables has a weight, and the objective is to find the assignment with the maximum weight.

Search strategies

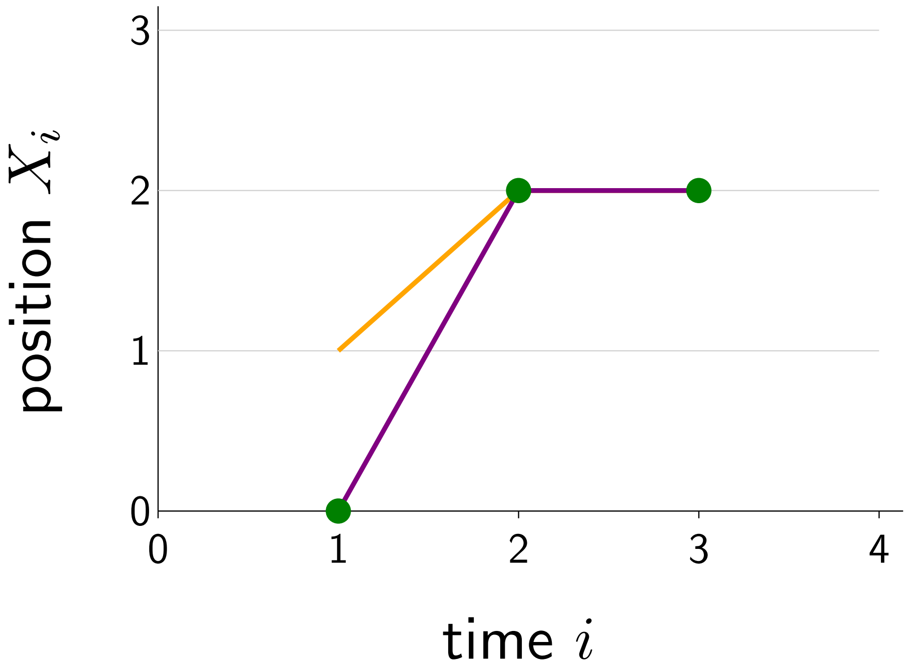
Backtracking/beam search: extend partial assignments

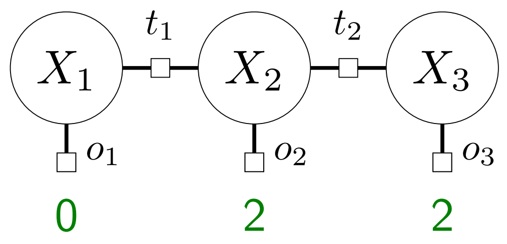
Local search: modify complete assignments

- So far, we've seen both backtracking and beam search. These search algorithms build up a partial assignment incrementally, and are structured around an ordering of the variables (even if it's dynamically chosen). With backtracking search, we can't just go back and change the value of a variable much higher in the tree due to new information; we have to wait until the backtracking takes us back up, in which case we lose all the information about the more recent variables. With beam search, we can't even go back at all.
- **Local search** (i.e., hill climbing) provides us with additional flexibility. Instead of building up partial assignments, we work with a complete assignment and make repairs by changing one variable at a time.



Example: object tracking





x_1	$o_1(x_1)$
0	2
1	1
2	0

x_2	$o_2(x_2)$
0	0
1	1
2	2

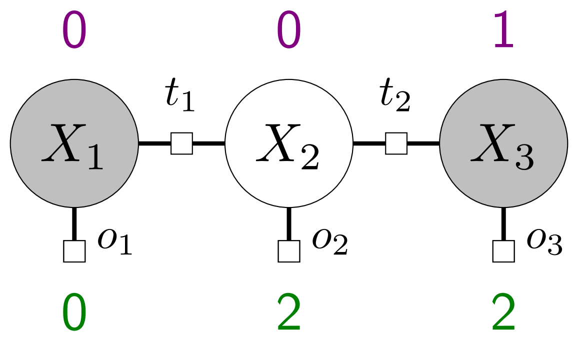
x_3	$o_3(x_3)$
0	0
1	1
2	2

$\|x_i - x_{i+1}\|$	$t_i(x_i, x_{i+1})$
0	2
1	1
2	0

[demo]

- Recall the object tracking example in which we observe noisy sensor readings 0, 2, 2.
- We have observation factors o_i that encourage the position X_i and the corresponding sensor reading to be nearby.
- We also have transition factors t_i that encourage the positions X_i and X_{i+1} to be nearby.

One small step



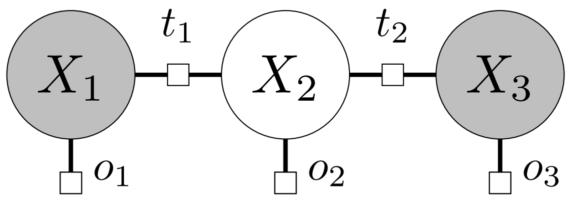
Old assignment: $(0,0,1)$; how to improve?

(x_1, v, x_3)	weight
$(0,0,1)$	$2 \cdot 2 \cdot 0 \cdot 1 \cdot 1 = 0$
$(0,1,1)$	$2 \cdot 1 \cdot 1 \cdot 2 \cdot 1 = 4$
$(0,2,1)$	$2 \cdot 0 \cdot 2 \cdot 1 \cdot 1 = 0$

New assignment: $(0,1,1)$

- Suppose we have a complete assignment $(0, 0, 1)$, perhaps randomly generated. This complete assignment has weight 0.
- Can we make a local change to the assignment to improve the weight? Let's just try setting x_2 to a new value v.
- For each possible value v, we compute the weight of the resulting assignment from setting $x_2 : v$.
- We then just take the v that produces the maximum weight.
- This results in a new assignment $(0, 1, 1)$ with a higher weight (4 rather than 0).
- This is one step of ICM, and one can now take another variable and try to change its value to improve the weight of the complete assignment.

Exploiting locality



Weight of new assignment (x_1, v, x_3):

$$o_1(x_1)t_1(x_1, v)o_2(v)t_2(v, x_3)o_3(x_3)$$

Key idea: locality

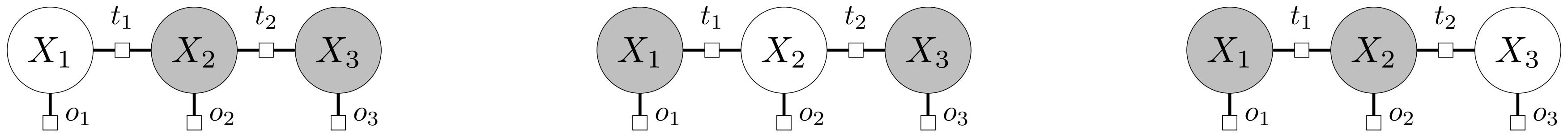
When evaluating possible re-assignments to X_i, only need to consider the factors that depend on X_i.

- There is one optimization we can make. If we write down the weight of a new assignment $x \cup \{X_2 : v\}$, we will notice that all the factors return the same value as before except the ones that depend on X_2.
- Therefore, we only need to compute the product of these relevant factors and take the maximum weight. Because we only need to look at the factors that touch the variable we're modifying, this can be a big saving if the total number of factors is much larger.

Iterated conditional modes (ICM)

Algorithm: iterated conditional modes (ICM)

```
Initialize $x$ to a random complete assignment
Loop through $i = 1, \dots, n$ until convergence:
    Compute weight of $x_v = x \cup \{X_i : v\}$ for each $v$
    $x \leftarrow x_v$ with highest weight
```

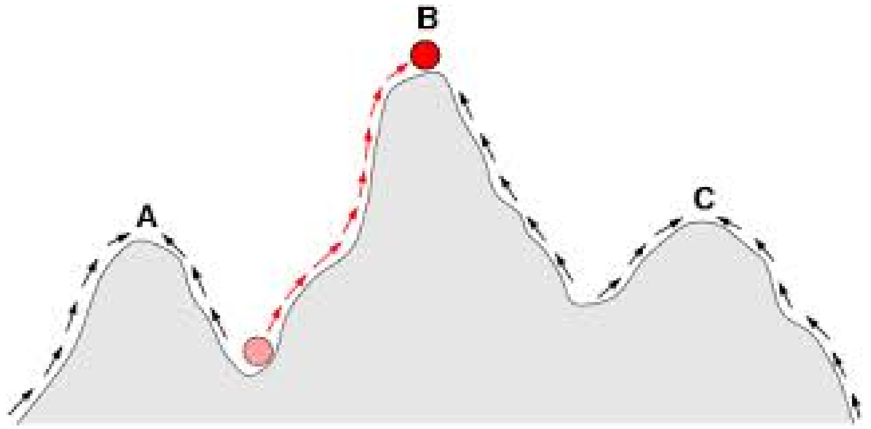


[demo: `iteratedConditionalModes()`]

- Now we can state our first algorithm, ICM. The idea is simple: we start with a random complete assignment. We repeatedly loop through all the variables X_i.
- On variable X_i, we consider all possible ways of re-assigning it $X_i : v$ for $v \in \text{Domain}_i$, and choose the new assignment that has the highest weight.
- Graphically, we represent each step of the algorithm by having shaded nodes for the variables which are fixed and unshaded for the single variable which is being re-assigned.
- Note that in the demo, ICM gets stuck in a local optimum with weight 4 rather than the global optimal weight of 8.

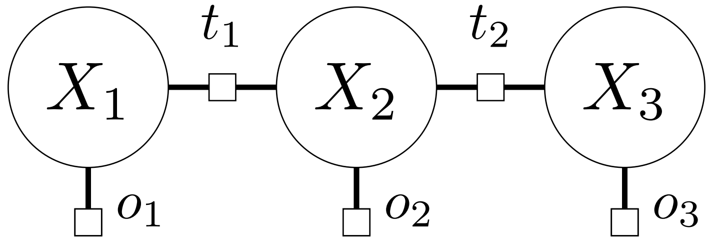
Convergence properties

- Weight(x) increases or stays the same each iteration
- Converges in a finite number of iterations
- Can get stuck in **local optima**
- Not guaranteed to find optimal assignment!



- Note that each step of ICM cannot decrease the weight because we can always stick with the old assignment.
- ICM terminates when we stop increasing the weight, which will happen eventually since there are a finite number of assignments and therefore possible weights we can increase to.
- However, ICM can get stuck in local optima, where there is a assignment with larger weight elsewhere, but no one-variable change increases the weight.
- Connection: this hill-climbing is called coordinate-wise ascent. We already saw an instance of coordinate-wise ascent in the K-means algorithm which would alternate between fixing the centroids and optimizing the object with respect to the cluster assignments, and fixing the cluster assignments and optimizing the centroids. Recall that K-means also suffered from local optima issues.
- There are two ways to mitigate local optima. One is to change multiple variables at once. Another is to inject randomness, which we'll see later with Gibbs sampling.

Summary



Algorithm	Strategy	Optimality	Time complexity
Backtracking search	extend partial assignments	exact	exponential
Beam search	extend partial assignments	approximate	linear
Local search (ICM)	modify complete assignments	approximate	linear

- This concludes our presentation of a local search algorithm, Iterated Conditional Modes (ICM).
- Let us summarize all the search algorithms for finding maximum weight assignment CSPs that we have encountered.
- Backtracking search starts with an empty assignment and incrementally build up partial assignments. It produces exact (optimal) solutions and requires exponential time (although heuristics such as dynamic ordering and AC-3 help).
- Beam search also extends partial assignments. It takes linear time in the number of variables, but yields approximate solutions.
- In this module, we've considered an alternative strategy, local search, which works directly with complete assignments and tries to improve them one variable at a time. If we always choose the value that maximizes the weight, we get ICM, which has the same characteristics as beam search: approximate but fast.