Example:

Sequence:
CUCGACCCCCUUGACGUCGCAAGUGUUGGGG

Ground truth:

bb_x	bb_y	siz_x	siz_y	bb_type
0	29	6	6	stem
9	22	4	4	stem
5	24	5	3	internal_loop
12	19	8	8	hairpin_loop

Proposals (at 0.1 threshold):
6 stems, 18 iloops, 3 hloops

Step 1: Filter out invalid bounding boxes

For iloop, we need at least one compatible stem on each side, i.e. iloop.top_right == some_stem1.bottom_left and iloop.bottom_left == some_stem2.top_right. Iloop not satisfying this is discarded.

For hloop, we need at least one compatible stem on the outer side, i.e. hloop.top_right == some_stem.bottom left. Also hloop needs to be on the diagonal of the matrix. Hloop not satisfying the above is discarded.

Result: after filtering, we're left with much fewer bounding boxes:

Stem

bb_x	bb_y	siz_x	siz_y	prob	bl_x	bl_y
0	29	6	6	[0.41025621353158326, 0.44290844275857927, 0.2...	5	24
5	30	4	4	[0.20544070627534108]	8	27
6	30	4	4	[0.4038486250672587, 0.27864132850155837, 0.27...	9	27
9	22	4	4	[0.6770828362027589, 0.573858923373826, 0.8764...	12	19
12	26	4	4	[0.09296749466427809]	15	23
13	23	2	2	[0.4102197996540831, 0.32933917103708865, 0.41...	14	22

iloop

bb_x	bb_y	siz_x	siz_y	prob	bl_x	bl_y	num_compatible_stem_outer	num_compatible_stem_inner
5.0	24.0	5.0	3.0	[0.22653995386454878, 0.4762805845032449, 0.40...	9.0	22.0	1.0	1.0
8.0	27.0	2.0	6.0	[0.21078391904750435]	9.0	22.0	1.0	1.0
8.0	27.0	5.0	2.0	[0.05582155712225877]	12.0	26.0	1.0	1.0
9.0	27.0	5.0	5.0	[0.10652713805518142, 0.0927118477820101]	13.0	23.0	1.0	1.0

hloop

bb_x	bb_y	siz_x	siz_y	prob	bl_x	bl_y	num_compatible_stem_outer
12	19	8.0	8.0	[0.5592991751604471, 0.6358038212924311, 0.709...	19.0	12.0	1.0
14	22	9.0	9.0	[0.5634663251435189, 0.2603944252215587, 0.666...	22.0	14.0	1.0

Step 2: Extend each bounding box by one step, towards the outer direction

For each bounding box, we find all compatible “next” bounding boxes where bounding_box.top_right == next.bottom_left. Following local constraints need to be satisfied:

hloop.next = stem
iloop.next = stem
stem.next = iloop or None

Result: each bounding box is associated with a list (can be empty) of other bounding boxes as candidate “next”:

```
{'iloop_0': iloop iloop_0 top right (5.0, 24.0), bottom left (9.0, 22.0). Next: ['stem_0'],
 'iloop_1': iloop iloop_1 top right (8.0, 27.0), bottom left (9.0, 22.0). Next: ['stem_1'],
 'iloop_2': iloop iloop_2 top right (8.0, 27.0), bottom left (12.0, 26.0). Next: ['stem_1'],
 'iloop_3': iloop iloop_3 top right (9.0, 27.0), bottom left (13.0, 23.0). Next: ['stem_2'],
 'stem_0': stem stem_0 top right (0, 29), bottom left (5, 24). Next: N/A,
 'stem_1': stem stem_1 top right (5, 30), bottom left (8, 27). Next: N/A,
 'stem_2': stem stem_2 top right (6, 30), bottom left (9, 27). Next: N/A,
 'stem_3': stem stem_3 top right (9, 22), bottom left (12, 19). Next: ['iloop_0', 'iloop_1'],
 'stem_4': stem stem_4 top right (12, 26), bottom left (15, 23). Next: ['iloop_2'],
 'stem_5': stem stem_5 top right (13, 23), bottom left (14, 22). Next: ['iloop_3'],
 'hloop_0': hloop hloop_0 top right (12, 19), bottom left (19.0, 12.0). Next: ['stem_3'],
 'hloop_1': hloop hloop_1 top right (14, 22), bottom left (22.0, 14.0). Next: ['stem_5']}
```

Step 3: Assemble a full “chain”

Since all the one-step parent-child relationship is given from step 2, in this step, we enumerate all possible short and long chains subject to constraint:

- start/root can be hloop or stem
- end can only be stem

Note that a particular chain can be a subset of others, since we're enumerating.

Also note that this can be implemented efficiently, since if all possible chains have been constructed for bounding box A, and we're trying to grow a new chain whose last element is A, then we can stitch the already known chains (starting from A) to the current one without exploration. (to be implemented)

Result:

```
[FullChain ['stem_0'] Completed,
 FullChain ['stem_1'] Completed,
 FullChain ['stem_2'] Completed,
 FullChain ['stem_3'] Completed,
 FullChain ['stem_3', 'iloop_0', 'stem_0'] Completed,
 FullChain ['stem_3', 'iloop_1', 'stem_1'] Completed,
 FullChain ['stem_4'] Completed,
 FullChain ['stem_4', 'iloop_2', 'stem_1'] Completed,
 FullChain ['stem_5'] Completed,
 FullChain ['stem_5', 'iloop_3', 'stem_2'] Completed,
 FullChain ['hloop_0', 'stem_3'] Completed,
 FullChain ['hloop_0', 'stem_3', 'iloop_0', 'stem_0'] Completed,
 FullChain ['hloop_0', 'stem_3', 'iloop_1', 'stem_1'] Completed,
 FullChain ['hloop_1', 'stem_5'] Completed,
 FullChain ['hloop_1', 'stem_5', 'iloop_3', 'stem_2'] Completed]
```

Note that in the above, we already recovered the ground truth solution:
hloop_0 top right (12, 19), bottom left (19.0, 12.0),
stem_3 top right (9, 22), bottom left (12, 19),
iloop_0 top right (5.0, 24.0), bottom left (9.0, 22.0),
stem_0 top right (0, 29), bottom left (5, 24))
(due to the fact that this is a short sequence with only one "chain")

Step 4: Global assembly and filter w.r.t. global constraints

Given the short and long chains from step 3, in this step we create all valid global assemblies. Although in theory the number of combinations is 2^n where n is the number of chains, in practise it's far less, due to these constraints and considerations:

- Chain A and chain B are not compatible if there exists one stem in A that has base pair conflict with one stem in B. Since we can pre-compute the pairwise bp pair compatibility between all stems, this is easy to check.

- Validate that the following conversion bounding_box -> base pair matrix -> bounding box results in the same set of bounding boxes. This is a easy way to check for the following constraints:
    - For a particular chain, if the innermost bounding box is a stem, then it's implied it connects to a multi-loop but NOT a hairpin loop, since hloop is not in the chain. This means there exists some base pairing between i and j, where (i,j) is the closing base of the stem. We can check this is the case and discard those not satisfying this.
    - Also, the 'empty space' between two inner stem bounding box implies multi-loop, we also need to check no base pairing exist between the two inner corners.

Stem bp conflict:

	stem_0	stem_1	stem_2	stem_3	stem_4	stem_5
stem_0	True	True	True	False	True	False
stem_1	True	True	True	False	False	False
stem_2	True	True	True	True	False	False
stem_3	False	False	True	True	True	True
stem_4	True	False	False	True	True	True
stem_5	False	False	False	True	True	True

True: conflict (thus all diagonals are True since we can't have the same stem bounding box twice)

After filtering according to above criteria:

```
: print(len(grow_global.global_structs), len(valid_global_structs))
```

```
26 6
```

The first one is 'no structure', the rest, with probability from stage 1 model added:

	bb_x	bb_y	siz_x	siz_y	bb_type	n_proposal	prob_median	n_proposal_norm
0	12	19	8	8	hloop	19	0.510153	0.296875
1	9	22	4	4	stem	16	0.559821	1.000000

	bb_x	bb_y	siz_x	siz_y	bb_type	n_proposal	prob_median	n_proposal_norm
0	12	19	8	8	hloop	19	0.510153	0.296875
1	9	22	4	4	stem	16	0.559821	1.000000
2	5	24	5	3	iloop	5	0.408967	0.333333
3	0	29	6	6	stem	36	0.394395	1.000000

	bb_x	bb_y	siz_x	siz_y	bb_type	n_proposal	prob_median	n_proposal_norm
0	12	19	8	8	hloop	19	0.510153	0.296875
1	9	22	4	4	stem	16	0.559821	1.000000
2	8	27	2	6	iloop	1	0.210784	0.083333
3	5	30	4	4	stem	1	0.205441	0.062500

	bb_x	bb_y	siz_x	siz_y	bb_type	n_proposal	prob_median	n_proposal_norm
0	14	22	9	9	hloop	21	0.630828	0.259259
1	13	23	2	2	stem	4	0.413329	1.000000

	bb_x	bb_y	siz_x	siz_y	bb_type	n_proposal	prob_median	n_proposal_norm
0	14	22	9	9	hloop	21	0.630828	0.259259
1	13	23	2	2	stem	4	0.413329	1.000000
2	9	27	5	5	iloop	2	0.099619	0.080000
3	6	30	4	4	stem	14	0.275844	0.875000

N_proposal is the number of pixels proposing this exact bounding box, prob_median is the median of all joint probabilities (one per pixel), n_proposal_norm is n_proposal normalized by the bounding box size.

Compare the above with the target (ground truth):

	bb_x	bb_y	siz_x	siz_y	bb_type
0	0	29	6	6	stem
1	9	22	4	4	stem
2	5	24	5	3	iloop
3	12	19	8	8	hloop

It’s the second one above. Note that all proposed bounding boxes have high probabilities and n_propoasl_norm, so for this particular example a greedy algorithm could have resulted in this global structure.

Potential problem:
The number of valid global structures might become unmanageable with the increase in sequence length, for example, with len=56:

```
row = df.iloc[15]
row
```

```
ensemble_diversity                                                  4.96
free_energy                                                        -10.1
len                                                                   56
mfe_frequency                                                   0.371533
one_idx              ([2, 3, 4, 5, 9, 10, 11, 24, 25, 26, 30, 31, 3...
seq                  AGCGACUACAGCAUACCCUUAAAAGCUUUUGUCGAGUGGCUUGCCA...
bounding_boxes       [((2, 30), (4, 4), stem), ((9, 24), (3, 3), st...
bb_stem              [{'bb_x': 0, 'bb_y': 40, 'siz_x': 3, 'siz_y': ...
bb_iloop             [{'bb_x': 2, 'bb_y': 36, 'siz_x': 3, 'siz_y': ...
bb_hloop             [{'bb_x': 2, 'bb_y': 10, 'siz_x': 9, 'siz_y': ...
Name: 16, dtype: object
```

We have:

```
print(len(grow_global.global_structs), len(valid_global_structs))
```

```
635 182
```

This makes the last step (evaluating/ranking) very costly.

We can do the following pre-processing:

Bounding box prediction from stage 1 is kept if one of the following conditions is satisfied:

- More than 3 pixels predict the bounding box (TODO use percentage)
- Max predicted probability >= 0.5

Step 5: Ranking of global structures (model? Heuristic using stage 1 probability?)

Example
df = pd.read_pickle('data/rand_s1_bb_0p1_global_structs_60.pkl.gz')

```
row = df.iloc[4000]
seq = row['seq']
print(seq)
print(len(seq))
print(row['gt_found'])
print(row['n_bb_found'], row['n_bb_target'])
```

```
UCUACGGCUAACAUACCAGAACAUUAAUGUCUGAC
35
True
4 4
```

```
df_target
```

	bb_x	bb_y	siz_x	siz_y	bb_type
0	16	32	4	4	stem
1	21	28	2	2	stem
2	19	29	3	2	iloop
3	22	27	6	6	hloop

```
def ad_hoc_score(df_pred):
    df_stem = df_pred[df_pred['bb_type'] == 'stem']
    df_stem = dgp.add_column(df_stem, 'score', ['siz_x', 'prob_median', 'n_proposal_norm'],
                             lambda a, b, c: a * b * c)
    return df_stem['score'].sum()
```

```
   bb_x  bb_y  siz_x  siz_y bb_type  n_proposal  prob_median  n_proposal_norm
0     6    15     10     10   hloop          43     0.473339             0.43
1     5    16      2      2    stem           4     0.661425             1.00
1.3228499085845131

   bb_x  bb_y  siz_x  siz_y bb_type  n_proposal  prob_median  n_proposal_norm
0     6    15     10     10   hloop          43     0.473339         0.430000
1     5    16      2      2    stem           4     0.661425         1.000000
2    22    27      6      6   hloop          15     0.847050         0.416667
3    20    29      3      3    stem           3     0.094281         0.333333
1.4171308491078685

   bb_x  bb_y  siz_x  siz_y bb_type  n_proposal  prob_median  n_proposal_norm
0     6    15     10     10   hloop          43     0.473339         0.430000
1     5    16      2      2    stem           4     0.661425         1.000000
2    22    27      6      6   hloop          15     0.847050         0.416667
3    21    28      2      2    stem           4     0.894705         1.000000
3.1122603890155247

   bb_x  bb_y  siz_x  siz_y bb_type  n_proposal  prob_median  n_proposal_norm
0     6    15     10     10   hloop          43     0.473339            0.430
1     5    16      2      2    stem           4     0.661425            1.000
2     2    17      4      2   iloop           3     0.128789            0.375
3     0    19      3      3    stem           9     0.155059            1.000
1.7880272317812598

   bb_x  bb_y  siz_x  siz_y bb_type  n_proposal  prob_median  n_proposal_norm
0     6    15     10     10   hloop          43     0.473339         0.430000
1     5    16      2      2    stem           4     0.661425         1.000000
2     2    17      4      2   iloop           3     0.128789         0.375000
3     0    19      3      3    stem           9     0.155059         1.000000
4    22    27      6      6   hloop          15     0.847050         0.416667
5    20    29      3      3    stem           3     0.094281         0.333333
1.8823081723046151
```

```
   bb_x  bb_y  siz_x  siz_y bb_type  n_proposal  prob_median  n_proposal_norm
0     6    15     10     10   hloop          43     0.473339         0.430000
1     5    16      2      2    stem           4     0.661425         1.000000
2     2    17      4      2   iloop           3     0.128789         0.375000
3     0    19      3      3    stem           9     0.155059         1.000000
4    22    27      6      6   hloop          15     0.847050         0.416667
5    21    28      2      2    stem           4     0.894705         1.000000
3.577437712212271

   bb_x  bb_y  siz_x  siz_y bb_type  n_proposal  prob_median  n_proposal_norm
0    19    29     11     11   hloop           8     0.053018         0.066116
1    16    32      4      4    stem          16     0.851025         1.000000
3.4040981343619583

   bb_x  bb_y  siz_x  siz_y bb_type  n_proposal  prob_median  n_proposal_norm
0    22    27      6      6   hloop          15     0.847050         0.416667
1    20    29      3      3    stem           3     0.094281         0.333333
0.09428094052335524

   bb_x  bb_y  siz_x  siz_y bb_type  n_proposal  prob_median  n_proposal_norm
0    22    27      6      6   hloop          15     0.847050         0.416667
1    21    28      2      2    stem           4     0.894705         1.000000
1.7894104804310114

   bb_x  bb_y  siz_x  siz_y bb_type  n_proposal  prob_median  n_proposal_norm
0    22    27      6      6   hloop          15     0.847050         0.416667
1    21    28      2      2    stem           4     0.894705         1.000000
2    19    29      3      2   iloop           3     0.656677         0.500000
3    16    32      4      4    stem          16     0.851025         1.000000
5.19350861479297
Best score!
Ground truth!
```

Statistics: (short sequence to speed up analysis)
Number of examples with len <= 60: 45885
Number of examples with 100% bounding box sensitivity (stage 1): 29025
Number of examples where best score structure is identical to the ground truth: 22799

	sensitivity	ppv
count	45015.000000	45015.000000
mean	0.764079	0.746885
std	0.340698	0.336887
min	0.000000	0.000000
25%	0.562500	0.545455
50%	1.000000	1.000000
75%	1.000000	1.000000
max	1.000000	1.000000

TODOs

Heuristics: More structure is better -> if global struct A is subset of B, discard A

For predicted best global structure not equal to ground truth, calculate FE and compare with ground truth

Debug stage 1 iloop data

Pseudo knot?

RNA-RNA interaction? Run stage 1 model three times, A-A, B-B & A-B, 2nd stage will have different constraints

Long sequence?

Greedy approach of assembly? Start with high prob bounding boxes, terminate after explored say 100 global structures?