Artificial Intelligence: Project 03: Golomb Ruler

We implemented three different approaches to find the optimal length Golomb Ruler for given L and M.

1. Plain Backtracking (BT)
2. BT + Forward Checking (FC)
3. BT + Constraint Propagation (CP)

We tested the results for different values of L and M and found the following relationship in context of number of nodes expanded (number of times recursion is called to try different legal values (which follow the constraints) for variables):

No. for Plain BT > No. for BT + FC > No. for BT + CP

The max value of L and M for which optimal Golomb Ruler can be calculated in few seconds is:

L	M	Output	Plain BT	BT + FC	BT + CP
55	**10**	(55, [0, 1, 6, 10, 23, 26, 34, 41, 53, 55])	**122.743s**	**109.527s**	118.545s

For L = 6 and M = 4, here are the different nodes explored or the different legal values tried:

BT: 5 nodes

{0: 0}
{0: 0, 1: 1}
{0: 0, 1: 1, 3: 3}
{0: 0, 1: 1, 4: 4}
{0: 0, 1: 1, 4: 4,6:6}

BT + FC: 4 nodes

{0: 0}
{0: 0, 1: 1}
{0: 0, 1: 1, 4: 4}
0: 0, 1: 1, 4: 4,6:6}

BT + CP: 4 nodes

{0: 0}
{0: 0, 1: 1}
{0: 0, 1: 1, 4: 4}
{0: 0, 1: 1, 4: 4,6:6}

Now we will try for L = 5 and M = 4 to get the optimal L, no. Of nodes expanded to find that "no" result exists for L = 5 and M = 4:

BT: 14 nodes

{0: 0}
{0: 0, 1: 1}
{0: 0, 1: 1, 3: 3}
{0: 0, 1: 1, 4: 4}
{0: 0, 1: 1, 5: 5}
{0: 0, 2: 2}
{0: 0, 2: 2, 3: 3}
{0: 0, 2: 2, 5: 5}
{0: 0, 3: 3}
{0: 0, 3: 3, 4: 4}
{0: 0, 3: 3, 5: 5}
{0: 0, 4: 4}
{0: 0, 4: 4, 5: 5}
{0: 0, 5: 5}

BT + FC: 11 nodes

{0: 0}
{0: 0, 1: 1}
{0: 0, 1: 1, 5: 5}
{0: 0, 2: 2}
{0: 0, 2: 2, 5: 5}
{0: 0, 3: 3}
{0: 0, 3: 3, 4: 4}
{0: 0, 3: 3, 5: 5}
{0: 0, 4: 4}
{0: 0, 4: 4, 5: 5}
{0: 0, 5: 5}

BT + CP:

{0: 0}
{0: 0, 1: 1}
{0: 0, 2: 2}
{0: 0, 3: 3}
{0: 0, 4: 4}

Here is table showing the number of nodes expanded for different values of L and M

L	M	Plain BT	BT + FP	BT + CP
34	8	1478	1170	280
44	9	19415	16116	3205
55	10	171512	148902	25500

As we can see that the number of nodes expanded decrease significantly for Backtracking as we use Forward checking or Constant Propagation. Number of nodes for CP are very less as compared to both BT and BT + FP.

Here are the stats for time taken for different approaches for different L and M:

Note: The time is the total time taken in finding the optimal length for given L and M I.e. for looking for different values of L until we find the optimal Length for given M

L	M	Output	Plain BT	BT + FP	BT + CP
10	**0**	(-1, [])	**0.0s**	**0.0s**	**0.0s**
3	**3**	(3, [0, 1, 3])	**0.0s**	**0.0s**	**0.0s**
11	**5**	(11, [0, 1, 4, 9, 11])	**0.001s**	**0.001s**	**0.002s**
17	**6**	(17, [0, 1, 4, 10, 12, 17])	**0.01s**	**0.01s**	**0.013s**
25	**7**	(25, [0, 1, 4, 10, 18, 23, 25])	**0.105s**	**0.113s**	**0.149s**
34	**8**	(34, [0, 1, 4, 9, 15, 22, 32, 34])	**1.048s**	**1.164s**	**1.626s**
44	**9**	(44, [0, 1, 5, 12, 25, 27, 35, 41, 44])	**11.123s**	**11.093s**	**14.403s**
55	**10**	(55, [0, 1, 6, 10, 23, 26, 34, 41, 53, 55])	**122.358s**	**101.645s**	118.545s

As we saw in previous table that the number of nodes expanded are very less for (BT + FP) and (BT + CP). **As the value of L increases, the total time in finding optimal L for (BT + FP) and (BT + CP) decreases as compared to the time needed for plain BT**.

The time taken by (BT + CP) is greater than time needed for (BT + FP) because of the time needed to update the domains multiple times for each variable every time value is updated. But number of nodes expanded are always lesser for (BT + CP) as compared to number of nodes expanded for Plain BT and (BT + FP)

For L = 45 and M = 8, Here is search flow of the different values of L to get the optimal value for L:

L = 45, Output = 45 [0, 1, 3, 7, 12, 20, 30, 45]

L = 44, Output = 44 [0, 1, 3, 7, 12, 20, 30, 44]

L = 43, Output = 43 [0, 1, 3, 7, 16, 21, 33, 43]

L = 42, Output = 42 [0, 1, 3, 7, 15, 24, 37, 42]

L = 41, Output = 41 [0, 1, 3, 7, 15, 20, 31, 41]

L = 40, Output = 40 [0, 1, 3, 7, 15, 24, 35, 40]

L = 39, Output = 39 [0, 1, 3, 8, 14, 18, 30, 39]

L = 38, Output = 38 [0, 1, 3, 8, 17, 28, 32, 38]

L = 37, Output = 37 [0, 1, 3, 13, 21, 28, 32, 37]

L = 36, Output = 36 [0, 1, 3, 13, 21, 27, 32, 36]

L = 35, Output = 35 [0, 1, 8, 20, 22, 25, 31, 35]

L = 34, Output = 34 [0, 1, 4, 9, 15, 22, 32, 34]

L = 33, Output = -1,[]

So optimal length is 33.