Tutorial
MDP Tool

MDP declaration language -

The tool allows creation of Markov Decision Processes (MDP) and facilitates in generating an optimal control policy for the MDP via decomposition and distributed optimization [1]. To use the tool, a text file has to be created that details the states and transitions of an MDP. To define an MDP the following keywords are used -

- **states -** States keyword is used to declare the states present in the MDP graph. The tool parses the source file to locate the keyword, the line containing the keyword should specify the names of all the states in the MDP within curly braces and separated by a comma. For example -

  states {s0,s1}

- **initial -** Initial keyword is used to declare the probability distribution of the initial states. Each state can be assigned an initial probability such that the total of all the probabilities is less than or equal to 1. User can choose to declare the initial probabilities of only some of the states of the MDP. In such a case the undeclared initial probabilities are assigned 0 by default. The tool parses the MDP source file to locate the initial keyword. Once located, each new line is treated as a distinct initial probability till the **end** keyword is detected to mark the end of initial probability distribution. Each initial probability is declared with the following format -

  {state, probability}

  For Example -

  initial
  {s0, 0.75}
  {s1, 0.2}
  {s3, 0.05}
  end

- **transitions -** Transitions keyword is used to declare the transitions present in the MDP graph. Each transition has 4 parameters associated with it, namely - source state, destination state, transition probability and action. Each transition must contain all 4 parameters to be accepted by the tool. Further, the parameters must be valid, in the sense that the transition probability should be between 0 and 1, the source and destination states must be defined before the transitions by using the states keyword. The tool parses the MDP source file to locate the transitions keyword. Once located, each new line is treated as a new transition till the **end** keyword is detected to mark the end of transitions. Each transition is declared with the following format -

  {source state, action, probability, destination state}

For example -

transitions
{s0,a,1,s1}
{s1,b,1,s0}
end

- **rewards -** Rewards keyword is used to declare the reward function for each state and action pair in the MDP. The user needs to provide a decimal based reward associated with each action that the agent can take at every state. The tool parses the MDP source file to locate the rewards keyword. Once located, each new line is treated as a new reward declaration till the **end** keyword is detected to mark the end of rewards. If the user does not specify a reward for a state action pair, then the reward is defined at 0 by default. Each transition is declared with the following format -

  {state, action, reward value}

  For example -

  rewards
  {s0,a,0}
  {s0,b,0}
  {s1,a,-0.7}
  {s1,b,0}
  {s2,b,-0.5}
  end

- **regions -** Regions keyword is used to declare the MDP decomposition in one of two ways. The user could either provide a manual decomposition to the tool or request the tool to generate a decomposition based on a DFS based decomposition algorithm explained later in the tutorial. The tool uses the algorithm to generate a decomposition that ensures, relatively small subproblem to be solved in distributed optimization (See [1] for the definition of size of subproblems). To request the tool for a decomposition, the user needs to provide the number of regions expected from the decomposition using the regions keyword with the following format -

  regions = N (where N is the number of regions requested)

- **regions -** The tool also allows the user to provide their own decomposition using the keyword “regions”. **To provide a manual decomposition, the user need not give the number of expected regions as in the case above.** The user should use the regions keyword and then declare a new region in each new line and use the **end** keyword to end the region declarations. Each region must contain a name which has the format of **ri** where **r** is tool identifier for regions and **i** is a number to declare multiple regions starting with r1. Further, each region is declared as a set of states with the following format -

ri={comma separated state names}

For example -

regions
r1={s0,s1}
r2={s2}
end

- **end -** End keyword is used to denote the end of regions and transitions declarations and explained in the above sections.

Notes -
- The MDP declaration language is not case sensitive.
- Comments can be given given by “**//”**. Comments **should not** be given within the region and transitions declarations.

MDP Tool Usage -

- The tool can be used on any system that has a Java Virtual Machine installed.
- To use the tool, user needs to write an MDP source file as described in the above section and provide that on the console, while executing the program.
- To provide the MDP source file to the tool, user needs to input the filename, along with the extension.
- If the file is in a different directory as the tool source code, then the user needs to provide the absolute path to the file along with the name and extension of the file.
- An example execution is given below -

MDP Source File -

```
states {s0, s1, s2}

initial
{s0, 1.0}
end

transitions
{s0, a, 0.7, s1}
{s0, a, 0.3, s0}
{s0, b, 1, s0}
{s1, a, 0.3, s1}
{s1, a, 0.7, s2}
{s1, b, 1, s0}
{s2, b, 0.5, s2}
{s2, b, 0.5, s1}
end

rewards
{s0,a,0}
{s0,b,0}
{s1,a,-0.7}
{s1,b,0}
{s2,b,-0.5}
end

regions=2
r1 = {s0,s1}
r2 = {s2}
end
```

Command -

java -jar MDP_Decomposer.java -Xms8G -Xmx8G -XX:+UseConcMarkSweepGC

Output -

Please enter the MDP source file name or enter Grid(n,r) to run on a gridworld problem : testMDP2.txt
Original K0 size = 2
Final K0 size = 2

Created LP -
Xvector - XVector.txt
A, B, C Vectors - A_B_C.mat
Time taken for LP creation : 76 mSec
--

The formulated LP problem is of the following form -

$$\min_{x \in \mathbb{R}_+^m} \sum_{j=0}^{N} c_j^T x_j, \quad \text{subject to } Ax = b,$$

where $c_j^T x_j = \sum_{s \in K_j} \sum_{a \in A(s)} -R(s,a)x(s,a)$,

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} & \dots & A_{0N} \\ A_{10} & A_{11} & & & \\ A_{20} & & A_{22} & & 0 \\ \vdots & 0 & & \ddots & \\ A_{N0} & & & & A_{NN} \end{bmatrix}, \quad b = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix},$$

Here the matrix A is generated from the supplied MDP source file based on the following equations -

$$\sum_{a \in A(s)} x(s,a) = u_0(s) + \gamma \cdot \sum_{i=0}^{N} \sum_{s' \in K_i} \sum_{a' \in A(s')} x(s',a') \cdot P(s',a')(s),$$

More details can be found in the original paper by J. Fu, S. Han, U. Topcu. The generated output contains 2 files i.e. XVector.txt and A_B_C.mat.

- XVector.txt - This file contains the state action pairs in each region, in the order in which they are placed in the A, B and C vectors.
- A_B_C.mat - This file contains the formulated LP matrices. It contains 3 matrices. Matrix A is generated as explained above and is broken down into multiple matrices while storing. Each matrix is marked by an index Aij, where i and j are indices to denote the kernels. It is stored in sparse representation in which each matrix Aij is broken down into 5 vectors i.e. Aijcol, Aiji, Aijj, Aijrow and Aijv. Where Aijcol and Aijrow store the row and column length of the matrix Aij. Aiji and Aijj each represent the location of each entry in Aijv within the Aij matrix. The matrices B and C are stored as vectors Bi and Ci where i is the kernel indices.

Generated A_B_C Matrix -

Name	Size	Bytes	Class	Attributes
A00col	1x1	8	double	[3]
A00i	1x5	40	double	[0,0,1,1,1]
A00j	1x5	40	double	[0,1,0,1,2]
A00row	1x1	8	double	[2]
A00v	1x5	40	double	[0.55,-0.63,-0.45,0.73,1]
A01col	1x1	8	double	[2]
A01i	1x1	8	double	[1]
A01j	1x1	8	double	[0]
A01row	1x1	8	double	[2]
A01v	1x1	8	double	[-0.63]
A10col	1x1	8	double	[3]
A10i	1x1	8	double	[0]
A10j	1x1	8	double	[2]
A10row	1x1	8	double	[1]
A10v	1x1	8	double	[-0.90]
A11col	1x1	8	double	[2]
A11i	1x2	16	double	[0,0]
A11j	1x2	16	double	[0,1]
A11row	1x1	8	double	[1]
A11v	1x2	16	double	[0.73,0.1]
B0	1x2	16	double	[0,0]
B1	1x1	16	double	[1]
C0	1x3	8	double	[-0.5,-0.7,0]
C1	1x2	16	double	[0,0]

Generated X Vector -

```
X Vector -
x0
(s2,b) (s1,a) (s1,b)
x1
(s0,a) (s0,b)
```

MDP Decomposition Algorithm -

The MDP decomposition algorithm used in the tool involves two different algorithms. The first step of the algorithm is to create a **base decomposition** of the MDP graph (as explained below). In the next step the base decomposition is improved by moving states between regions to find an optimal decomposition which minimizes the inter regional transitions.

1. **Base Decomposition -** To create a base decomposition the tool performs a recursive depth first search from the initial state of the MDP graph. Each state explored is placed in the the first region **r1**, till the number of states in that region exceeds the count of (no. of states/no. of regions). Then a new region is created and the following states are placed in that region. This process continues till all the states have been placed in a region. The algorithm is described in the following pseudo-code (region is an identifier to be used in a map of regions to states) -

```
static void DFSDecomposition(String initialState, MDP mdp, int regionLevel)
{
        if (size(region(regionLevel) > stateCount/regionCount)
        {
                regionLevel++;
                createRegion(regionLevel);
        }
        region(regionLevel).add(initialState);
        seenStates.add(initialState);
        next = getNextStates(initialState);
        for(State s in next)
        {
                if(s not in seenStates)
                {
                        DFSDecomposition(s,mdp,region);
                }
        }
}
```

2. **Improving Decomposition -** The base decomposition assigns states to regions based on their position in the graph. This can cause a high number of inter regional transition, which lead to a high number of states in the K0 kernel. The improvement algorithm aims to minimize the size of kernel K0 such that the size of K0 is approximately the size of Ki for $0 < i <= N$. To reduce the size of K0, the improvement algorithm, checks each state in the MDP graph. For each state a map is created which counts the number of transitions to and from a region associated with that state. The resultant map contains key value pairs that associate a region and the number of transition to/from that region connected to that state. This is used to find the region that has the maximum number of transitions to/from that region connected to that state. If the state does not belong to that region then the state's region is changed to that region. This results in reducing the number of inter regional transitions. This is performed for all the states in the MDP graph. The algorithm can be bootstrapping in nature and thus it should be done iteratively till the algorithm converges. This occurs when either the resultant decomposition stops changing or it

starts oscillating between a fixed number of decompositions. In such a case the algorithm should be terminated. The algorithm is described in the pseudo-code -

```
public static Map<String, LinkedHashSet<String>> improveDecomposition(MDP mdp)
{
	regionMap = new Map <string, Map<String,Integer>>();
	for(every state s in MDP)
	{
		regionMap.put(s.Label, new Map<String,Integer>());
		for(every transition t in state s)
		{
			String region=t.getToState().getRegionLabel();
			if(regionMap.get(s.Label).contains(region))
			{
				int count = regionMap.get(s.Label).get(region);
				regionMap.get(s.Label).put(region, ++count);
			}
			else
			{
				regionMap.get(s.Label).put(region, 1);
			}
			String toStateLabel=t.getToState().Label;
			if(!regionMap.contains(toState))
			{
				regionMap.put(toStateLabel, new
				Map<String,Integer>());
			}
			if(regionMap.get(toStateLabel).contains(region))
			{
				int count = regionMap.get(toStateLabel).get(region);
				regionMap.get(toStateLabel).put(region, ++count);
			}
			else
			{
				regionMap.get(toStateLabel).put(region, 1);
			}
		}

	}

	for(every state s in regionMap)
	{
		maxRegion=regionMap.get(s.Label).getMaxKey();
		if(s.getRegion()!=maxRegion)
		{
			s.setRegion(maxRegion);
		}
	}
}
```

MDP Decomposition Algorithm Run Time -

Let n - number of states in the MDP and m - number of transition in the MDP

1. **Base Decomposition -**

   T=O(n+m)
   The above time complexity is based on the time complexity of DFS.

2. **Improving Decomposition -**

   T=O(n*m+n)=O(nm)

To create a sparse matrix-

number of states = n
number of regions = r
number of kernels = r+1 -> approx to r
Max number of states/kernel = n/r
actions per state = worst case of total number of actions = a

T = O(r * r * n/r * n/r * a) = O(n * n * a)

Has been improved to -

T= O((3r+1) * (n/r * n/r *a)) = O(n * n * a / r)

Experimentation Results -

Note - All the experiments were done using the following flags ***- -Xms8G -Xmx8G -XX:+UseConcMarkSweepGC*** *to improve performance.*

1. Grid World (4x4, 2 regions) -

Please enter the MDP source file name or enter Grid(n,r) to run on a gridworld problem :
Grid(4,2)
-------GRID WORLD--------
Size - 16
End State -16
Heaven State - 12
Hell State - 4
Blocked States -
3 8 14

Original K0 size = 8
Final K0 size = 8

Created LP -
Grid World Source MDP - testMDP4.txt
Xvector - XVector.txt
A, B, C Vectors - A_B_C.mat
Time taken for LP creation : 203 mSec
---

2. Grid World (10x10, 3 regions) -

Please enter the MDP source file name or enter Grid(n,r) to run on a gridworld problem :
Grid(10,3)
-------GRID WORLD--------
Size - 100
End State -100
Heaven State - 90
Hell State - 10
Blocked States -
81 3 37 85 25 76 60 63 47

Original K0 size = 57
Final K0 size = 57

Created LP -
Grid World Source MDP - testMDP4.txt
Xvector - XVector.txt
A, B, C Vectors - A_B_C.mat
Time taken for LP creation : 477 mSec
---

3. Grid World (20x20, 10 regions) -

Please enter the MDP source file name or enter Grid(n,r) to run on a gridworld problem :
Grid(20,10)
-------GRID WORLD--------
Size - 400
End State -400
Heaven State - 380
Hell State - 20
Blocked States -

33	162	131	99	100	231	136	264	360	74	333	113	338
	212	57	122	159	95	191						

Original K0 size = 327
Final K0 size = 253

Created LP -
Grid World Source MDP - testMDP4.txt
Xvector - XVector.txt
A, B, C Vectors - A_B_C.mat
Time taken for LP creation : 2369 mSec
----

4. Grid World (4x4, 2 regions) -

Please enter the MDP source file name or enter Grid(n,r) to run on a gridworld problem :
Grid(30,15)
-------GRID WORLD--------
Size - 900
End State -900
Heaven State - 870
Hell State - 30
Blocked States -

64	322	644	69	263	647	77	781	209	82	662	599	153
	283	160	36	361	493	301	750	814	434	757	54	888
	376	57	570	251								

Original K0 size = 746
Final K0 size = 375

Created LP -
Grid World Source MDP - testMDP4.txt
Xvector - XVector.txt
A, B, C Vectors - A_B_C.mat
Time taken for LP creation : 6420 mSec
----

5. Grid World (40x40, 20 regions) -

Please enter the MDP source file name or enter Grid(n,r) to run on a gridworld problem :
Grid(40,20)
-------GRID WORLD--------
Size - 1600
End State -1600
Heaven State - 1560
Hell State - 40
Blocked States -
128 384 320 901 839 1100 846 271 786 915 84 20 22
1369 604 925 991 799 352 1184 1058 931 995 549 42
940 878 368 1137 177 1269 309 1591 315 60 636
1085 1598 639

Original K0 size = 1334
Final K0 size = 493

Created LP -
Grid World Source MDP - testMDP4.txt
Xvector - XVector.txt
A, B, C Vectors - A_B_C.mat
Time taken for LP creation : 17927 mSec
----

6. Grid World (50x50, 25 regions) -

Please enter the MDP source file name or enter Grid(n,r) to run on a gridworld problem :
Grid(50,25)
-------GRID WORLD--------
Size - 2500
End State -2500
Heaven State - 2450
Hell State - 50
Blocked States -
2434 258 1027 518 902 1800 1416 1545 912 1298 533 2458
1948 1313 2471 2473 1449 1963 1839 1592 440 2104 1722 1979 828
2109 1602 1859 1222 1735 71 2248 2125 1361 1883 732 739
2147 486 1000 2408 2281 1641 1259 1910 124 764 1021 638

Original K0 size = 2104
Final K0 size = 789

Created LP -
Grid World Source MDP - testMDP4.txt
Xvector - XVector.txt
A, B, C Vectors - A_B_C.mat
Time taken for LP creation : 45743 mSec

----

7. Grid World (60x60, 30 regions) -

Please enter the MDP source file name or enter Grid(n,r) to run on a gridworld problem :
Grid(60,30)
-------GRID WORLD--------
Size - 3600
End State -3600
Heaven State - 3540
Hell State - 60
Blocked States -
3330 2309 136 1547 268 527 148 1812 3348 789 2971 3355 284
542 2591 1056 2211 3493 2343 1960 1449 1707 1717 53 440
1976 1592 317 1472 194 2498 2115 2119 1353 1993 3534 846
974 1873 337 2655 480 3168 2146 2787 2916 2661 1126
2534 2279 2668 1900 1006 2419 2037 3445 2553 2170 2301

Original K0 size = 2738
Final K0 size = 905

Created LP -
Grid World Source MDP - testMDP4.txt
Xvector - XVector.txt
A, B, C Vectors - A_B_C.mat
Time taken for LP creation : 63746 mSec
----

8. Grid World (70x70, 35 regions) -

Please enter the MDP source file name or enter Grid(n,r) to run on a gridworld problem :
Grid(70,35)
-------GRID WORLD--------
Size - 4900
End State -4900
Heaven State - 4830
Hell State - 70
Blocked States -
2176 1797 1926 3462 390 4871 2952 779 2699 2189 3728 1940
2710 2966 4248 2329 1690 4507 2075 1565 3743 4002 1828 2084
3240 3372 1965 4785 1586 4019 1973 4154 4285 3773 2749 1982 446
707 3140 4554 1866 3275 3147 4427 3276 2509 4049 4442 989
2781 3678 1503 3169 3681 4324 4452 3045 4457 1514 875
3566 3054 2802 4339 4087 2170 2427 1916 4605

Original K0 size = 4060
Final K0 size = 1237

Created LP -
Grid World Source MDP - testMDP4.txt
Xvector - XVector.txt
A, B, C Vectors - A_B_C.mat
Time taken for LP creation : 143180 mSec
---

9. Grid World (100x100, 50 regions) -

Please enter the MDP source file name or enter Grid(n,r) to run on a gridworld problem :
Grid(100,50)
-------GRID WORLD--------
Size - 10000
End State -10000
Heaven State - 9900
Hell State - 100
Blocked States -

4096	7937	6658	6157	7438	5140	4375	5912	2841	3610	3614	4897	
3361	809	2092	5422	7984	6960	819	824	2365	2624	8769	1347	
4421	3915	334	1618	4692	4184	3166	3937	1125	2408	6250	5740	
4206	2159	4719	1907	5747	9587	6517	3448	9338	8826	4228	5510	
5256	1417	9866	651	7820	2446	9615	9617	1686	6040	5528	2713	
6041	6059	1708	4014	1199	8368	7346	7095	4288	9664	9668	3271	199
	7367	6602	970	2763	9420	4045	2511	8143	6355	980	8916	
5844	7639	9178	4319	6882	6639	1264	4336	4594	2803	7156	1524	
5367	9207	9210										

Original K0 size = 8458
Final K0 size = 1761

Created LP -
Grid World Source MDP - testMDP4.txt
Xvector - XVector.txt
A, B, C Vectors - A_B_C.mat
Time taken for LP creation : 512969 mSec
