# Problem Solving – State-Space Search and Control Strategies

Chapter 2 (a)

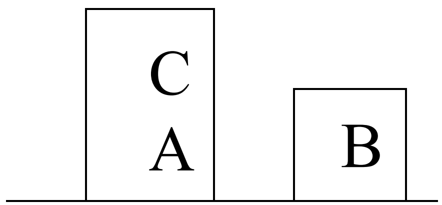
# Problem Characteristics

- Heuristic search is a very general method applicable to a large class of problem.
- In order to choose the most appropriate method (or combination of methods) for a particular problem it is necessary to analyze the problem along several key dimensions.
  - Is the problem decomposable into a set of independent smaller sub problems?

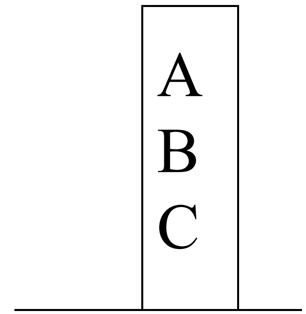
# Contd..

- Decomposable problems can be solved by the **divide-and-conquer** technique.
  - Each sub-problem is simpler to solve.
  - Each sub-problem can be handed over to a different processor. Thus can be solved in parallel processing environment.
- Is problem non decomposable? Need different strategies
  - For example, Block world problem is non decomposable.

Initial State (State0)




Goal State




Start: ON(C, A) ….

Goal: ON(B, C) $\wedge$ ON(A, B)

# Contd..

- Can solution steps be ignored or at least undone if they prove to be unwise?
    - In real life, there are three types of problems:
        - Ignorable,
        - Recoverable and
        - Irrecoverable.

# Example - Ignorable

- **(Ignorable)**: **In theorem proving -** (solution steps can be ignored)
  - Suppose we have proved some lemma in order to prove a theorem and eventually realized that lemma is no help at all, then ignore it and prove another lemma.
  - Can be solved by using simple **control strategy**?

# Example - Recoverable

- **8 puzzle game** - (solution steps can be undone)
  - Objective of 8 puzzle game is to rearrange a given initial configuration of eight numbered tiles on 3 X 3 board (one place is empty) into a given final configuration (goal state).
    - Rearrangement is done by sliding one of the tiles into empty square.
  - Steps can be undone if they are not leading to solution.
  - Solved by backtracking, so control strategy must be implemented using a **push down stack**.

# Example - Irrecoverable

- **Chess** (solution steps cannot be undone)
  - A stupid move cannot be undone.
  - Can be solved by **planning process.**

# Contd..

- What is the Role of knowledge?
  - In Chess game, knowledge is important to constrain the search
  - Newspapers scanning to decide some facts, a lot of knowledge is required even to be able to recognize a solution.
- Is the knowledge Base consistent?
  - Should not have contradiction

# Contd…

- Is a good solution Absolute or Relative ?
  - In water jug problem there are two ways to solve a problem.
    - If we follow one path successfully to the solution, there is no reason to go back and see if some other path might also lead to a solution.
    - Here a solution is **absolute**.
  - In travelling salesman problem, our goal is to find the shortest route/path. Unless all routes are known, the shortest is difficult to know.
    - This is a best-path problem whereas water jug is any-path problem.

# Contd…

- Any path problem can often be solved in reasonable amount of time using heuristics that suggest good paths to explore.
- Best path problems are in general computationally harder than any-path.

# Problem Solving

- AI programs have a clean separation of
  - computational components of data,
  - operations & control.
- Search forms the core of many intelligent processes.
- It is useful to structure AI programs in a way that facilitates describing the search process.

# Production System - PS

- PS is a formation for structuring AI programs which facilitates describing search process.
- It consists of
  - Initial or start state of the problem
  - Final or goal state of the problem
  - It consists of one or more databases containing information appropriate for the particular task.
- The information in databases may be structured
  - using knowledge representation schemes.

# Production Rules

- PS contains set of production rules,
  - each consisting of a left side that determines the applicability of the rule and
  - a right side that describes the action to be performed if the rule is applied.
  - These rules operate on the databases.
  - Application of rules change the database.
- A control strategy that specifies the order in which the rules will be applied when several rules match at once.
- One of the examples of Production Systems is an Expert System.

# Advantages of PS

- In addition to its usefulness as a way to describe search, the production model has other advantages as a formalism in AI.
  - It is a good way to model the strong state driven nature of intelligent action.
  - As new inputs enter the database, the behavior of the system changes.
  - New rules can easily be added to account for new situations without disturbing the rest of the system, which is quite important in real-time environment.

# Example : Water Jug Problem

- **Problem statement**:
  - Given two jugs, a 4-gallon and 3-gallon having no measuring markers on them. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into 4-gallon jug.
- **Solution**:
  - State for this problem can be described as the set of ordered pairs of integers (X, Y) such that
    - X represents the number of gallons of water in 4-gallon jug and
    - Y for 3-gallon jug.
  - Start state is (0,0)
  - Goal state is (2, N) for any value of N.

# Production Rules

- Following are the production rules for this problem.

R1: $(X, Y \mid X < 4) \quad \square \quad (4, Y)$
{Fill 4-gallon jug}
R2: $(X, Y \mid Y < 3) \quad \square \quad (X, 3)$
{Fill 3-gallon jug}
R3: $(X, Y \mid X > 0) \quad \square \quad (0, Y)$
{Empty 4-gallon jug}
R4: $(X, Y \mid Y > 0) \quad \square \quad (X, 0)$
{Empty 3-gallon jug}
R5: $(X, Y \mid X+Y >= 4 \wedge Y > 0) \quad \square (4, Y - (4 - X))$
{Pour water from 3- gallon jug into 4-gallon jug until 4-gallon jug is full}

# Contd..

R6: (X, Y | X+Y >= 3 Λ X > 0) □ (X – (3 – Y), 3)
{ Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full}

R7: (X, Y | X+Y <= 4 Λ Y > 0) □ (X+Y, 0)
{ Pour all water from 3-gallon jug into 4-gallon jug }

R8: (X, Y | X+Y <= 3 Λ X > 0) □ (0, X+Y)
{ Pour all water from 4-gallon jug into 3-gallon jug }

**Superficial Rules:** {May not be used in this problem}

R9: (X, Y | X > 0) □ (X – D, Y)
{ Pour some water D out from 4-gallon jug}

R10: (X, Y | Y > 0) □ (X, Y - D)
{ Pour some water D out from 3- gallon jug}

# Trace of steps involved in solving the water jug problem - First solution

| Number of Steps | Rules applied | 4-g jug | 3-g jug | |
|---|---|---|---|---|
| 1 | **Initial State** | 0 | 0 | |
| 2 | R2 {Fill 3-g jug} | 0 | 3 | |
| 3 | R7 {Pour all water from 3 to 4-g jug } | 3 | 0 | |
| 4 | R2 {Fill 3-g jug} | 3 | 3 | |
| 5 | R5 {Pour from 3 to 4-g jug until it is full} | 4 | 2 | |
| 6 | R3 {Empty 4-gallon jug} | 0 | 2 | |
| 7 | R7 {Pour all water from 3 to 4-g jug} | 2 | 0 | **Goal State** |

# Trace of steps involved in solving the water jug problem - Second solution

- Note that there may be more than one solutions.

| *Number of steps* | *Rules applied* | *4-g jug* | *3-g jug* |
|---|---|---|---|
| 1 | **Initial State** | 0 | 0 |
| 2 | R1 {Fill 4-gallon jug} | 4 | 0 |
| 3 | R6 {Pour from 4 to 3-g jug until it is full } | 1 | 3 |
| 4 | R4 {Empty 3-gallon jug} | 1 | 0 |
| 5 | R8 {Pour all water from 4 to 3-gallon jug} | 0 | 1 |
| 6 | R1 {Fill 4-gallon jug} | 4 | 1 |
| 7 | R6 {Pour from 4 to 3-g jug until it is full} | 2 | 3 |
| 8 | R4 {Empty 3-gallon jug} | 2 | 0 **Goal State** |

# Important Points

- For each problem
  - there is an initial description of the problem.
  - final description of the problem.
  - more than one ways of solving the problem.
  - a path between various solution paths based on some criteria of goodness or on some heuristic function is chosen.
  - there are set of rules that describe the actions called production rules.
    - Left side of the rules is current state and right side describes new state that results from applying the rule.

- Summary: In order to provide a formal description of a problem, it is necessary to do the following things:
  - Define a state space that contains all the possible configurations of the relevant objects.
  - Specify one or more states within that space that describe possible situations from which the problem solving process may start. These states are called **initial states.**
  - Specify one or more states that would be acceptable as solutions to the problem called **goal states.**
  - Specify a set of rules that describe the actions. Order of application of the rules is called control strategy.
  - Control strategy should cause motion towards a solution.

# Control Strategies

- Control Strategy decides which rule to apply next during the process of searching for a solution to a problem.
- Requirements for a good Control Strategy
  - **It should cause motion**

    In water jug problem, if we apply a simple control strategy of starting each time from the top of rule list and choose the first applicable one, then we will never move towards solution.

  - **It should explore the solution space in a systematic manner**

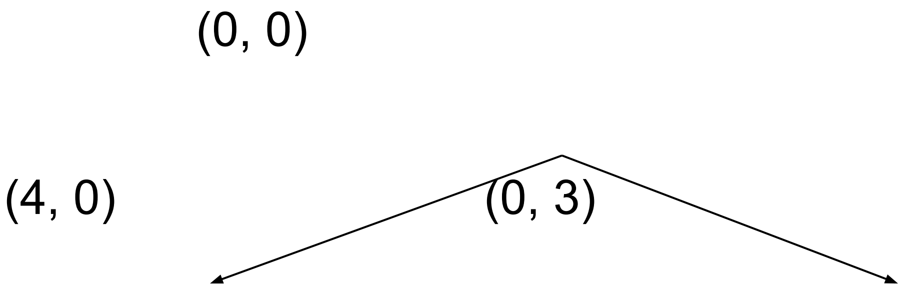
    If we choose another control strategy, say, choose a rule randomly from the applicable rules then definitely it causes motion and eventually will lead to a solution. But one may arrive to same state several times. This is because control strategy is not systematic.

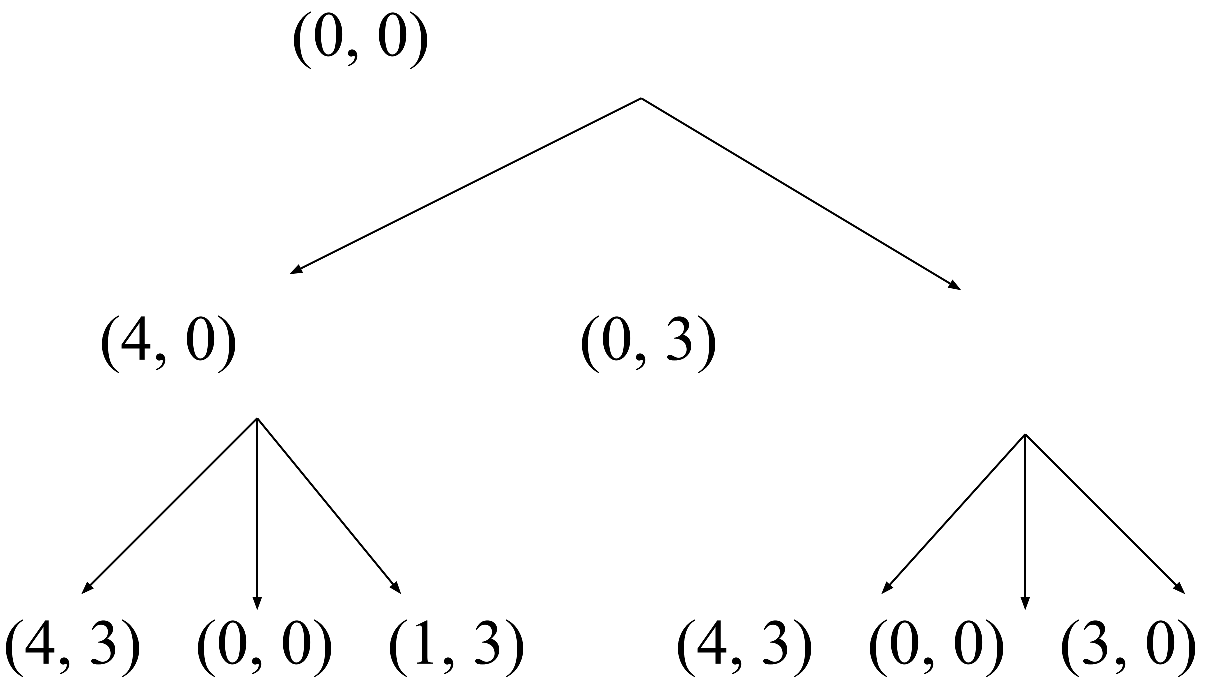
# Systematic Control Strategies (Blind searches)

- Blind searches are exhaustive in nature.
- These are uninformed searches.
- If the problem is simple then
  - any control strategy that causes motion and is systematic will lead to an answer.
- But in order to solve some real world problems,
  - we must use a control strategy that is efficient.
- Let us discuss these strategies using water jug problem.
- These may be applied to any search problem.

# Breadth First Search – BFS : Water Jug Problem

- BFS
  - Construct a tree with the initial state of the problem as its root.
  - Generate all the offspring of the root by applying each of the applicable rules to the initial state.
  - For each leaf node, generate all its successors by applying all the rules that are appropriate.
  - Repeat this process till we find a solution, if it exists.




(0, 0)
(4, 0)
(0, 3)
(4, 3)
(0, 0)
(1, 3)
(4, 3)
(0, 0)
(3, 0)

# Depth First Search - DFS

- Here we pursue a single branch of the tree until it yields a solution or some pre-specified depth has reached.
- If solution is not found then
  - go back to immediate previous node and
  - explore other branches in DF fashion.
- Let us see the tree formation for water jug problem using DFS

(0, 0) **Start State**

(4, 0)

(4, 3) (0, 0) (1, 3)

(0, 3) (4, 0)

× (4, 3) × (0, 0) (3, 0)

(3, 3) (0, 0) (0, 3)

× (3, 0) × (0, 3) (4, 2)

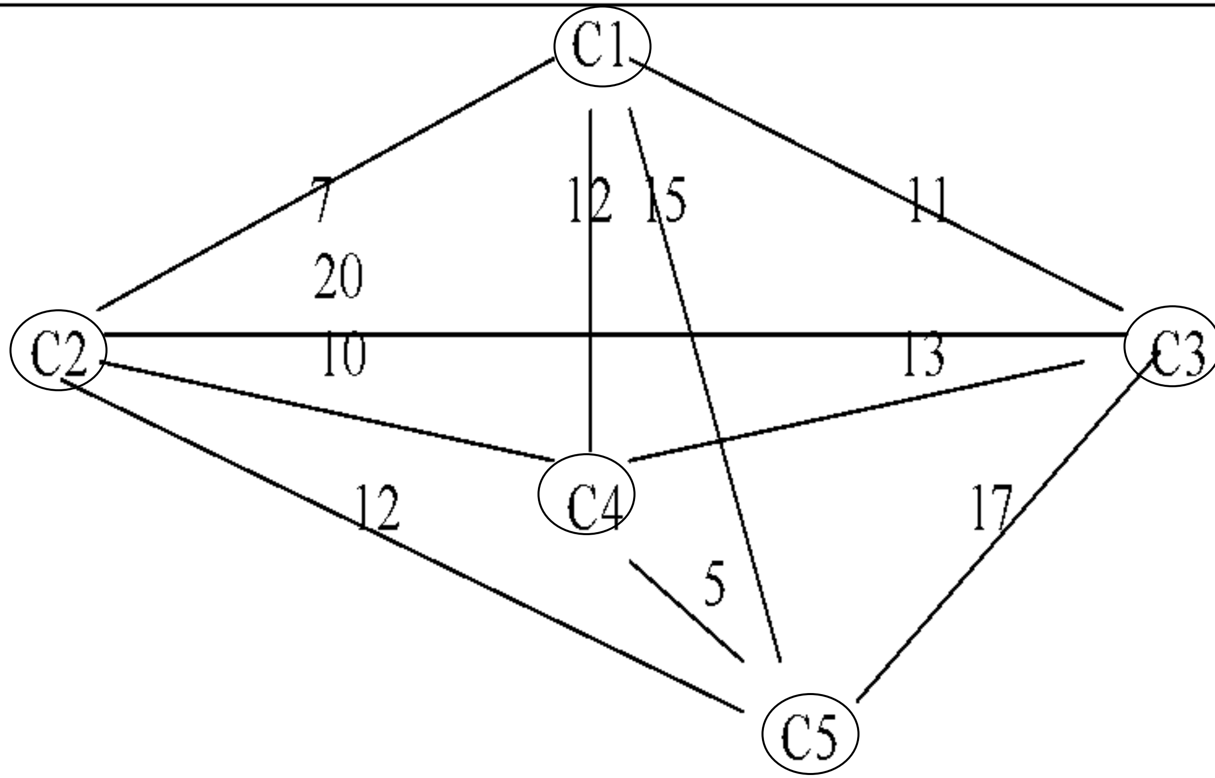
(0, 2) (3, 3) (4, 0)

× (4, 2) × (0, 0) (2, 0) **Goal state**

# Traveling Salesman Problem

- Consider 5 cities.
  - A salesman is supposed to visit each of 5 cities.
  - All cities are pair wise connected by roads.
  - There is one start city.
  - The problem is to find the shortest route for the salesman who has to
    - visit each city only once and
    - returns to back to start city.

# Traveling Salesman Problem (Example) – Start city is C1




D(C1,C2) = 7; D(C1,C3) = 11; D(C1,C4) = 12; D(C1,C5) = 15; D(C2,C3) =20;

D(C2,C4) = 10; D(C2,C5) =12; D(C3,C4) =13; D(C3,C5) = 17; D(C4,C5) =5;

# Contd..

- A simple motion causing and systematic control structure could, in principle solve this problem.
- Explore the search tree of all possible paths and return the shortest path.
- This will require 4! paths to be examined.
- If number of cities grow, say 25 cities, then the time required to wait a salesman to get the information about the shortest path is of 0(24!) which is not a practical situation.

# Contd..

- This phenomenon is called **combinatorial explosion**.
- We can improve the above strategy as follows:
  - Begin generating complete paths, keeping track of the shortest path found so far.
  - Give up exploring any path as soon as its partial length becomes greater than the shortest path found so far.
  - This algorithm is efficient than the first one, still requires exponential time $\propto$ some number raised to N (number of cities).

| **Paths explored. Assume C1 to be the start city** | | **Distance** |
|---|---|---|
| **1.** C1 → C2 → C3 → C4 → C5 → C1<br>7 20 13 5 15<br>27 40 45 60 | current best path | 60 √ × |
| **2.** C1 → C2 → C3 → C5 → C4 → C1<br>7 20 17 5 12<br>27 44 49 61 | | 61 × |
| **3.** C1 → C2 → C4 → C3 → C5 → C1<br>7 10 13 17 15<br>17 40 57 72 | | 72 × |
| **4.** C1 → C2 → C4 → C5 → C3 → C1<br>7 10 5 17 11<br>17 22 39 50 | current best path, cross path at S.No 1. | 50 √ × |
| **5.** C1 → C2 → C5 → C3 → C4 → C1<br>7 12 17 13 12<br>19 36 49 61 | | 61 × |
| **6.** C1 → C2 → C5 → C4 → C3 → C1<br>7 12 5 13 11<br>19 24 37 48 | current best path, cross path at S.No. 4. | 48 √ |
| **7.** C1 → C3 → C2 → C4 → C5 → C1<br>11 20 10 5 15<br>31 41 46 63 | | 63 × |

| **Paths explored. Assume C1 to be the start city** | **Distance** |
|---|---|
| **8.** C1 → C3 → C2 → C5 → C4 (not to be expanded further)<br>11 20 12 5<br>37 49 54 | 54 × |
| **9.** C1 → C3 → C4 → C2 → C5 → C1<br>11 13 10 12 15<br>24 34 46 61 | 61 × |
| **10.** C1 → C3 → C4 → C5 → C2 → C1 same as current best path at S. No. 6.<br>11 13 5 12 7<br>24 29 41 48 | 48 √ |
| **11.** C1 → C3 → C5 → C2 (not to be expanded further)<br>11 17 12<br>38 50 | 50 × |
| **12.** C1 → C3 → C5 → C4 → C2 (not to be expanded further)<br>11 17 5 10<br>38 43 53 | 53 × |
| **13.** C1 → C4 → C2 → C3 → C5 (not to be expanded further)<br>12 10 20 17<br>22 42 55 | 59 × |
| Continue like this | |

# Missionaries and Cannibals

- *Problem Statement:* Three missionaries and three cannibals want to cross a river. There is a boat on their side of the river that can be used by either one or two persons.
- Constraint:
  - At any point in time, cannibals should not be more than missionaries.
    - If the cannibals ever outnumber the missionaries (on either bank) then the missionaries will be eaten.

# Contd…

- PS for this problem can be described as the set of ordered pairs of left and right bank of the river as (L, R) where each bank is represented as a list [nM, mC, B]
  - n is the number of missionaries M, m is the number of cannibals C, and B represents boat.
- Start state: ( [3M, 3C, 1B], [0M, 0C, 0B] ),
  - 1B means that boat is present and 0B means it is not there on the bank of river.
- Goal state: ( [0M, 0C, 0B], [3M, 3C, 1B] )

# Contd…

- Any state: $([n_1 M,\ m_1 C,\ 1B],\ [n_2\ M,\ m_2\ C,\ 0B])$, with constraints/conditions as $n_1\ (\neq 0) \geq m_1$; $n_2\ (\neq 0) \geq m_2$; $n_1 + n_2 = 3$, $m_1 + m_2 = 3$
  - By no means, this representation is unique.
  - In fact one may have number of different representations for the same problem.
  - The table on the next slide consists of production rules based on the chosen representation.

| Set of Production Rules<br>Applied keeping constrains in mind | | | |
|---|---|---|---|
| **RN** | **Left side of rule** | $\rightarrow$ | **Right side of rule** |
| *Rules for boat going from left bank to right bank of the river* | | | |
| L1 | $([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$ | $\rightarrow$ | $([(n_1-2)M, m_1C, 0B], [(n_2+2)M, m_2C, 1B])$ |
| L2 | $([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$ | $\rightarrow$ | $([(n_1-1)M,(m_1-1)C,0B],[(n_2+1)M,(m_2+1)C, 1B])$ |
| L3 | $([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$ | $\rightarrow$ | $([n_1M, (m_1-2)C, 0B], [n_2M, (m_2+2)C, 1B])$ |
| L4 | $([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$ | $\rightarrow$ | $([(n_1-1)M, m_1C,0B],[(n_2+1)M, m_2C, 1B])$ |
| L5 | $([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$ | $\rightarrow$ | $([n_1M, (m_1-1)C, 0B], [n_2M, (m_2+1)C, 1B])$ |
| *Rules for boat coming from right bank to left bank of the river* | | | |
| R1 | $([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$ | $\rightarrow$ | $([(n_1+2)M, m_1C, 1B], [(n_2-2)M, m_2C, 0B])$ |
| R2 | $([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$ | $\rightarrow$ | $([(n_1+1)M,(m_1+1)C,1B],[(n_2-1)M,(m_2-1)C, 0B])$ |
| R3 | $([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$ | $\rightarrow$ | $([n_1M, (m_1+2)C, 1B], [n_2M, (m_2-2)C, 0B])$ |
| R4 | $([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$ | $\rightarrow$ | $([(n_1+1)M, m_1C,1B],[(n_2-1)M, m_2C, 0B])$ |
| R5 | $([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$ | $\rightarrow$ | $([n_1M, (m_1+1)C, 1B], [n_2M, (m_2-1)C, 0B])$ |

# One of the possible paths

| | | |
|---|---|---|
| **Start →** | **([3M, 3C, 1B], [0M, 0C, 0B]]** | |
| L2: | ([2M, 2C, 0B], [1M, 1C, 1B]) | **1M,1C →** |
| R4: | ([3M, 2C, 1B], [0M, 1C, 0B]) | **1M ←** |
| L3: | ([3M, 0C, 0B], [0M, 3C, 1B]) | **2C →** |
| R4: | ([3M, 1C, 1B], [0M, 2C, 0B]) | **1C ←** |
| L1: | ([1M, 1C, 0B], [2M, 2C, 1B]) | **2M →** |
| R2: | ([2M, 2C, 1B], [1M, 1C, 0B]) | **1M,1C ←** |
| L1: | ([0M, 2C, 0B], [3M, 1C, 1B]) | **2M →** |
| R5: | ([0M, 3C, 1B], [3M, 0C, 0B]) | **1C ←** |
| L3: | ([0M, 1C, 0B], [3M, 2C, 1B]) | **2C →** |
| R5: | ([0M, 2C, 1B], [3M, 1C, 0B]) | **1C ←** |
| L3: | **([0M, 0C, 0B], [3M, 3C, 1B])** | **2C → Goal state** |