วิชา BADS7601 - Artificial Intelligence

เรื่อง เทคนิคการแก้ปัญหารูบิค ขนาด 3x3 ด้วย IDA*

สมาชิก

1. วิชิต ชำนาญนาวา	6310422055	AIML
2. ภวิต บุญรัตน์	6310422069	AIML

สารบัญ

บทคัดย่อ

ความเป็นมาของ Rubik ขนาด 3x3 6 หน้าถือนับเป็นของเล่นยุค ปี 70 ที่ยังคงความมีเสน่ห์ชวนท้าทายมาถึงปัจจุบันแม้ผ่านการเวลามาเนิ่นนานราว ครึ่งศตวรรษทว่าการแก้ปัญหา Rubik ก็ยังเป็นที่แพร่หลายไปทั่วด้วยความที่ว่า ขอบข่ายรูปแบบปัญหามีประมาณ 43,252,003,274,489,856,000 (43 ล้าน ล้าน ล้าน) รูปแบบ ดังนั้นการหมุน Rubik แต่ละครั้งนั้นทำให้วิธีการแก้ปัญหาของการหมุนแต่ละครั้งแตกต่างกันไปสำหรับวิธีการการแก้ปัญหาRubik โดยการสร้างรูปแบบการค้นหาจากโครงแบบจำลองทั้งหมดทั้งในเชิงลึก (Depth First Search) หรือในเชิงกว้าง (Breadth first search)คงเป็นเรื่องที่ยากเนื่องจากรูปแบบที่เกิดขึ้นมีความมี nodesปริมาณมหาศาลซึ่งอาจเป็นข้อจำกัดในการทำงานของเครื่องคอมพิวเตอร์โดยทั่วไปในการประมวลผล จากการศึกษาวิชา BADS7601 (Artificial Intelligence) จึงมีแนวคิดเลือกใช้ IDA* ในการแก้ปัญหา

วัตถุประสงค์ :

สร้าง Program การแก้ปัญหา Rubik ขนาด 3x3

ขอบเขตการศึกษา :

ศึกษาและสร้าง โปรแกรมการแก้ปัญหาของ Rubik ขนาด 3 x3 - 6 หน้าสีด้วยหลักการ IDA*โดยได้ออกแบบฟังชันก์การคำนวนค่า Heuristics จำนวน 3 รูปแบบได้แก่

1.คำนวนจากจำนวนความแตกต่างของตำแหน่งที่อยู่ผิดตำแหน่งของทุกๆหน้า

2.คำนวนจากการใช้ Machine Learning

3.คำนวนโดยใช้ Manhattan Distance

สิ่งคาดว่าจะได้รับ

สามารถนำหลักการและแนวคิดจากวิชาBADS7601มาประยุกต์ใช้ในกรแก้ปัญหาที่มีระดับความซับซ้อนอย่างกรณีการแก้ปัญหาการหมุนของRubikในลักษณะต่างๆที่มีความแตกต่างของสถานการณ์เป็นจำนวนมหาศาล

นิยามปฏิบัติการ :

IDA*, Heuristics, Manhattan Distance, Machine Learning

ประวัติความเป็นมาของ Rubik

รูบิก (Rubik's Cube) หรือที่เรียกกันว่า ลูกรูบิก เป็นของเล่นลับสมอง ประดิษฐ์ขึ้นในปี ค.ศ. 1974 โดยแอร์เนอ รูบิก (Ernő Rubik) ซึ่งเป็นศาสตราจารย์และสถาปนิกชาวฮังการี โดยทั่วไป ตัวลูกบาศก์นั้นทำจากพลาสติกแบ่งเป็นชิ้นย่อยๆ26ชิ้นประกอบกันเป็นรูปลูกบาศก์ที่สามารถบิดหมุนไปรอบๆ ได้ ส่วนที่มองเห็นได้ของแต่ละด้าน จะประกอบด้วย 9 ส่วนย่อย ซึ่งมีสีทั้งหมด 6สีส่วนประกอบที่หมุนไปมาได้นี้ทำให้ การจัดเรียงสีของส่วนต่าง ๆ สลับกันได้หลายรูปแบบ จุดประสงค์ของเกมคือ การจัดเรียงให้แถบสีทั้ง 9 ที่อยู่ในด้านเดียวกันของลูกบาศก์ (ซึ่งมีทั้งหมด 6 ด้าน) มีสีเดียวกันลูกบาศก์ของรูบิกได้รับความนิยมสูงสุดในช่วงต้นของทศวรรษ 1980 และได้กลายเป็นหนึ่งในสัญลักษณ์ของวัฒนธรรมสมัยนิยมของยุคนั้นลูกบาศก์ของรูบิกนั้นถือได้ว่าเป็นเป็นของเล่นที่ขายได้มากที่สุดในโลก โดยมีจำนวนยอดขายรวมทั้งของแท้และของเลียนแบบมากกว่า 300,000,000 ชิ้นทั่วโลกลูกบาศก์ของรูบิกได้รับการคิดค้นขึ้นในปี ค.ศ.1974 โดยแอร์เนอรูบิกสถาปนิกชาวฮังการีผู้สนใจในเรขาคณิตและรูปทรงสามมิติ แอร์เนอได้จดสิทธิบัตร HU170062 สิ่งประดิษฐ์ในชื่อ "ลูกบาศก์มหัศจรรย์" (Magic Cube) ในปี ค.ศ.1975 ที่ประเทศฮังการี แต่ไม่ได้จดสิทธิบัตรนานาชาติ มีการผลิตชุดแรกเพื่อสำรวจตลาดในปลายปี ค.ศ. 1977 โดยจำหน่ายในร้านของเล่นในกรุงบูดาเปสต์หลังจากนั้นลูกบาศก์นี้ก็ได้รับความนิยมเพิ่มมากขึ้นทั่วทั้งประเทศฮังการีโดยการบอกเล่าปากต่อปาก วงการศึกษาในกลุ่มประเทศตะวันตกก็เริ่มให้ความสนใจ ในเดือนกันยายน ค.ศ. 1979 บริษัทไอเดียลทอยส์ (Ideal Toys)ได้ทำข้อตกลงเพื่อจัดจำหน่ายทั่วโลก มีการเปิดตัวของลูกบาศก์นี้ในระดับนานาชาติที่งานแสดงของเล่นที่กรุงลอนดอน นครนิวยอร์ก เมืองเนือร์นแบร์ก และกรุงปารีส ในช่วงต้นปี ค.ศ. 1980 บริษัทไอเดียลทอยส์เปลี่ยนชื่อของเล่นนี้เป็น "ลูกบาศก์ของรูบิก" (Rubik'sCube)และได้ส่งออกลูกบาศก์นี้จากประเทศฮังการีชุดแรก เพื่อจำหน่ายในเดือนพฤษภาคม ค.ศ. 1980ชื่อ "ลูกบาศก์ของรูบิก" เป็นเครื่องหมายการค้าของบริษัท "SevenTownsLimited"ดังนั้นบริษัทไอเดียลทอยส์จึงลังเลที่จะผลิตของเล่นนี้ในขณะนั้นปรากฏ

ของลอกเลียนแบบออกจำหน่าย ในปี ค.ศ. 1984 บริษัทไอเดียลทอยส์แพ้คดีละเมิดสิทธิบัตรหมายเลข US3655201 ซึ่งฟ้องร้องโดยแลร์รี นิโคลส์ (Larry Nichols) ชาวญี่ปุ่นชื่อ อิชิงิ เทรูโตชิ (Terutoshi Ishigi) ได้ทำการจดสิทธิบัตรของเล่นที่มีลักษณะเกือบจะเหมือนกันกับลูกบาศก์ของรูบิก หมายเลข JP55—8192 ที่ประเทศญี่ปุ่น ในช่วงเวลาระหว่างที่สิทธิบัตรที่รูบิกขอนั้นกำลังอยู่ระหว่างดำเนินการ นายอิชิกิจึงได้รับการยอมรับโดยทั่วไปว่าเป็นการค้นพบซ้ำกัน

ลูกบาศก์ของรูบิกมีขนาดมาตรฐานโดยประมาณ 2 1/8 นิ้ว (5.4 เซนติเมตร) กว้าง ยาว และสูงลูกบาศก์ประกอบด้วยลูกบาศก์ขนาดย่อม26ชิ้น"ชิ้นกลางหน้า"ของแต่ละด้านจะเป็นชิ้นที่มีสีหน้าเดียวและเชื่อมต่อกับกลไกการหมุนที่แกนกลางซึ่งชิ้นกลางหน้าที่ยึดติดกับแกนกลางนี้จะเป็นโครงสร้างที่ขัดส่วนที่เหลือไว้ด้วยกันและหมุนไปมาได้ดังนั้นทั้งหมดจะมี27ชิ้นส่วนแกนกลางสำหรับหมุน 1 ชิ้นชิ้นกลางหน้า6ชิ้นและชิ้นอื่นๆอีก20ชิ้นซึ่งสามารถประกอบเข้ากับชิ้นกลางหน้าที่ยึดติดกับแกนหมุนได้พอดีโดยจะมีส่วนที่ออกแบบให้ยึดขัดกันไม่ให้หลุดออกจากกัน แต่หมุนไปมาได้ การแยกชิ้นส่วนของลูกบาศก์ก็ไม่ได้ยากอะไรเพียงแต่งัดชิ้นขอบให้หลุดออกมาส่วนที่เหลือก็จะหลุดออกจากกันเองการแก้ปัญหาลูกบาศก์ของรูบิกโดยวิธีการแยกส่วนประกอบเป็นวิธีที่ง่ายแต่ขาดความท้าทายนอกเหนือจากชิ้นกลางหน้าแล้ว จะมีลูกบาศก์ขนาดย่อมอีก 20 ชิ้น มี 12 ชิ้นเป็น "ชิ้นขอบ" ซึ่งมีสี 2 ด้าน และ 8 ชิ้นเป็น "ชิ้นมุม" ซึ่งมีสี 3 ด้าน

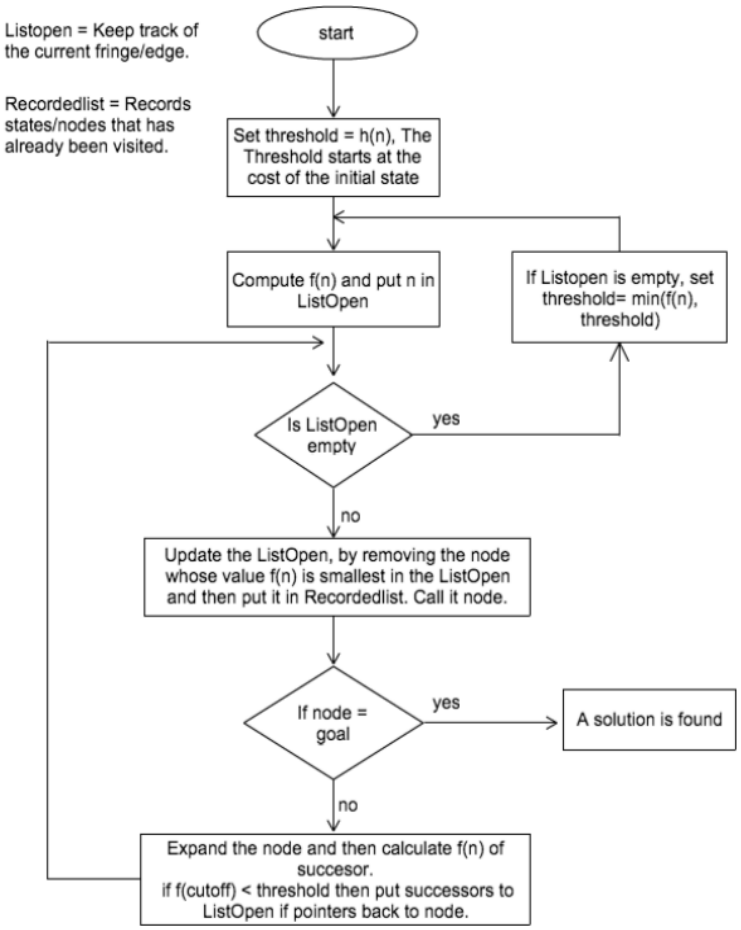
รูปแบบของปัญหาที่เกิดขึ้นสำหรับ Rubik 3 x3

ลูกบาศก์ของรูบิกมีจำนวนรูปแบบการเรียงสับเปลี่ยนที่แตกต่างกันทั้งหมด $8x3^7x\ 12!x\ 2^{10}$ รูปแบบ ซึ่งเมื่อคำนวณแล้วจะมีค่าเท่ากับ 43,252,003,274,489,856,000 รูปแบบ ($\sim 4.33 \times 10^{19}$) หรือประมาณ 43 ล้าน ล้าน ล้าน (quintillion) รูปแบบ
ถึงแม้จะมีรูปแบบการจัดเรียงเป็นจำนวนมาก แต่ทุกรูปแบบสามารถแก้ได้ภายในการบิด 20 ครั้งหรือน้อยกว่า (Richard E. Korf , 1997)

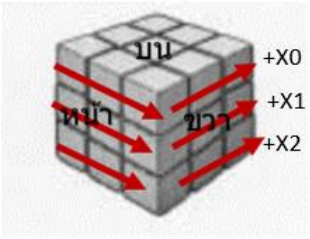
หลักการทำงานของ IDA*

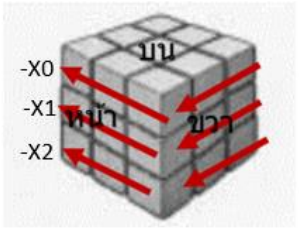
1. procedure IDAStar* ()
2. cutoff ⟵ f(s) = h(n)
3. while goal node is not found, or no new nodes exist do
4. DFS search to explore nodes with f-values within cutoff
5. if goal not found then
6. extend cutoff to next unexpanded value if there exists on

Flow Chart แสดงกระบวนการทำงานของโปรแกรม

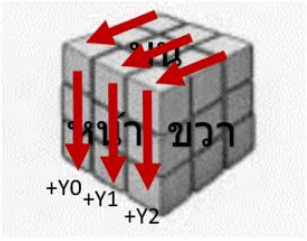


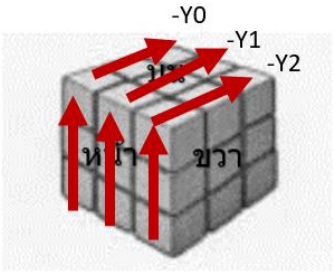
จำนวน Branching Factor



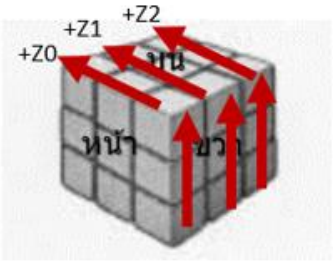


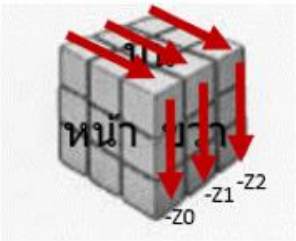
ทิศทาง (+X0,+X1+X2 ,-X0,X1,X2)





ทิศทาง (+Y0,+Y1+Y2 ,-Y0,Y1,Y2)





ทิศทาง (+Z0,+Z1+Z2 ,-Z0,Z1,Z2)

จำนวน Branching Factor มีค่าเท่ากับรูปแบบในการหมุน ในแต่ละครั้งจำนวน **18 รูปแบบคือ**

+X0,-X0,+X1,-X1,+X2,-X2,+X3,-X3,+Y0,-Y0,+Y1,-Y1,+Y2,-Y2,+Z0,-Z0,+Z1,-Z1,+Z2,-Z2

คำนวน bigO

Branching Factor จำนวน 18 ก้าน ในการ Expand node แต่ละตัว

Distance	Count of Positions
0	1
1	18
2	243
3	3,240
4	43,239
5	574,908
6	7,618,438
7	100,803,036
8	1,332,343,288
9	17,596,479,795
10	232,248,063,316
11	3,063,288,809,012
12	40,374,425,656,248
13	531,653,418,284,628
14	6,989,320,578,825,358
15	91,365,146,187,124,313
16	about 1,100,000,000,000,000,000
17	about 12,000,000,000,000,000,000
18	about 29,000,000,000,000,000,000
19	about 1,500,000,000,000,000,000
20	about 490,000,000

ตาราง : รูปแบบที่เกิดขึ้นจากการหมุนรูบิกจากจุดเริ่มต้น (ที่มา Cube20.org)

เราจะได้ค่าทดสอบที่มีค่าความระดับง่ายที่สุดจนถึงยากที่สุดทั้งนี้ระดับความง่ายของรูบิคจะเริ่มต้นที่มีการเปลี่ยนทิศทางแค่ครั้งเดียว ซึ่งมีด้วยกันทั้ง สิ้น 18 รูปแบบและถ้ามีการเปลี่ยนทิศทางเพิ่มขึ้นอีก ระดับความยากและรูปแบบของรูบิกจะเพิ่มขึ้นดังตาราง

จากการคำนวนจำนวน node ทั้งหมดคร่าวๆ พบว่ามีขนาดที่เยอะมากๆจึงเลือกใช้ IDA* ในการแก้ปัญหาเนื่องจากใช้ memory น้อย สามารถแก้ปัญหาที่มีขนาดใหญ่ได้ดี

การดำเนินการ

1.เขียนโปรแกรม Rubik แล้วตรวจสอบว่าการหมุนทุกๆแบบเป็นไปด้วยความถูกต้อง โดยกำหนดการหมุนไว้จำนวน 18 แกนตามที่กล่าวมาแล้วข้างต้น

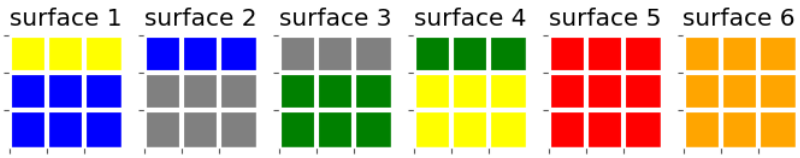
2.เขียนโปรแกรมการแก้ปัญหา Rubik ตามหลักการของ IDA*

ซึ่งจากการเขียนโปรแกรมโดยใช้หลัก IDA* ในครั้งแรกจะพบว่า Heristic Function ค่อนข้างมีความสำคัญในการแก้ปัญหา rubik ว่าต้อง Expand node มากน้อยแค่ไหน ถ้า Heristic Function มีประสิทธิภาพดี จะ Expand node น้อยทำให้ใช้เวลาไปให้ Goal เร็วขึ้น โดยครั้งแรกสุดใช้ Different Position Surface,Regression model, Manhuston distance ตามลำดับ

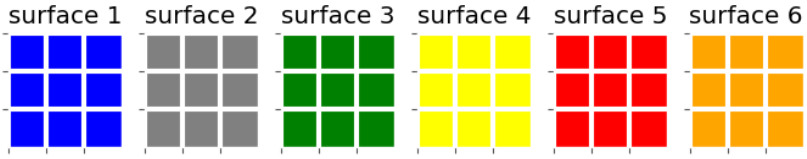
Heristic Function

1.Different Position Surface คำนวน ค่า heristic จากตำแหน่งของสีของชิ้นส่วน rubik ที่อยู่ผิดตำแหน่ง ตัวอย่าง

State ปัจจุบัน



State Goal



จากภาพมีจำนวนชิ้นส่วนของ rubik อยู่ผิดตำแหน่งอยู่ 12 ที่ ค่า heristic จะได้ 12

ถ้า state นั้นอยู่ที่ goal คือไม่มีชิ้นส่วนของ rubik ที่อยู่ผิดตำแหน่งเลย ค่า heristic จะได้ 0

2.Regression model คำนวนค่า heristic จาก model Machine Learning ในการ predict ค่า heristic จาก state ของ rubik ปัจจุบัน โดยมีขั้นตอนการทำดังนี้

2.1 เริ่มจากการสร้าง trainning data จากการคำนวนการหมุนแต่ละครั้ง ถ้าหมุน 1 ครั้งจาก goal จะเก็ยค่า Y = 1 ถ้าหมุน 2 ครั้งจาก goal จะเก็บค่า Y = 2 โดยได้ทำการ random สร้าง trainng data จำนวน 100,000 ชุด

	5.0	2.0	0.0	0.0.1	0.0.2	4.0	5.0.1	0.0.3	0.0.4	4.0.1	...	1.0.3	1.0.4	1.0.5	3.0.7	5.0.6	1.0.6	4.0.5	2.0.6	4.0.6	31
0	5.0	1.0	1.0	4.0	5.0	5.0	3.0	0.0	3.0	5.0	...	0.0	4.0	0.0	2.0	1.0	3.0	1.0	3.0	5.0	16
1	5.0	5.0	0.0	3.0	5.0	0.0	5.0	4.0	3.0	4.0	...	3.0	0.0	4.0	2.0	0.0	0.0	1.0	3.0	4.0	17
2	1.0	1.0	1.0	1.0	1.0	5.0	1.0	1.0	1.0	2.0	...	5.0	5.0	5.0	5.0	5.0	5.0	0.0	3.0	0.0	4
3	3.0	5.0	3.0	4.0	1.0	0.0	0.0	1.0	3.0	5.0	...	1.0	2.0	2.0	0.0	5.0	3.0	5.0	0.0	2.0	19
4	5.0	5.0	3.0	4.0	1.0	4.0	2.0	0.0	3.0	5.0	...	1.0	3.0	2.0	5.0	5.0	5.0	0.0	4.0	4.0	18
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99994	4.0	3.0	4.0	2.0	5.0	2.0	1.0	0.0	5.0	2.0	...	4.0	3.0	3.0	0.0	3.0	0.0	3.0	3.0	5.0	47
99995	0.0	1.0	0.0	5.0	3.0	0.0	3.0	4.0	1.0	3.0	...	4.0	2.0	5.0	3.0	5.0	1.0	4.0	0.0	3.0	47
99996	5.0	2.0	5.0	4.0	1.0	1.0	3.0	3.0	4.0	1.0	...	5.0	5.0	1.0	4.0	5.0	2.0	1.0	3.0	0.0	23
99997	0.0	5.0	1.0	3.0	0.0	5.0	4.0	4.0	1.0	2.0	...	3.0	2.0	0.0	4.0	3.0	1.0	3.0	2.0	1.0	45
99998	5.0	1.0	3.0	5.0	5.0	5.0	1.0	0.0	4.0	5.0	...	4.0	3.0	3.0	0.0	2.0	4.0	4.0	3.0	5.0	43

99999 rows × 55 columns

2.2 ใช้ Randomforest Regression ในการ Train model

```
In [16]:
1 #train model
2 from sklearn.ensemble import RandomForestRegressor
3 # X, Y = make_regression(n_features=4, n_informative=2,random_state=0, shuffle=False)
4 regr = RandomForestRegressor(max_depth=20, random_state=3)
5 regr.fit(X, Y)

Out[16]: RandomForestRegressor(max_depth=20, random_state=3)
```

2.3 ทดสอบและได้ model มาพร้อมที่จะใช้งานเก็บ model ไว้ในไฟล์

RandomForestRegressor18_model.sav

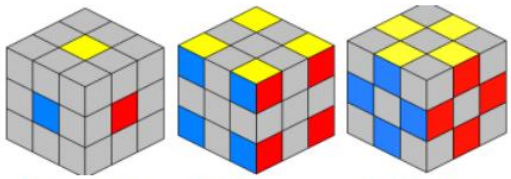
3.Manhuston distance

h(n) = max{(hconers(n), he1,he2(n)}

The hcorners(n) : calculates the minimum number of moves to fix all the corners in the correct position.

he1(n) : calculates the minimum number of moves to fix half of the edge cubes

he2(n) : fixes the rest of the edges.



(a) Center pieces (b) Corner pieces (c) Edges pieces

การใช้งานโปรแกรม Ai+Rubik

1.ติดตั้ง library ที่เกี่ยวข้อง

- tkinter
- numpy
- matplotlib
- random
- copy
- json
- time
- csv
- pickle

2.Run Program

2.1 **Full Program with UI** >>>> python Ai+Rubik.py

2.2 **Test program** >>>> python test_rubik_ida.py

Enter Solver method 1.Diiferent positioin 2.Machine Learning 3.Manhuston

Distance >>> 2

Enter move Rubik ex. -x1+y0 >>> -x1+y0

3.องค์ประกอบต่างๆของ Program ประกอบไปด้วย

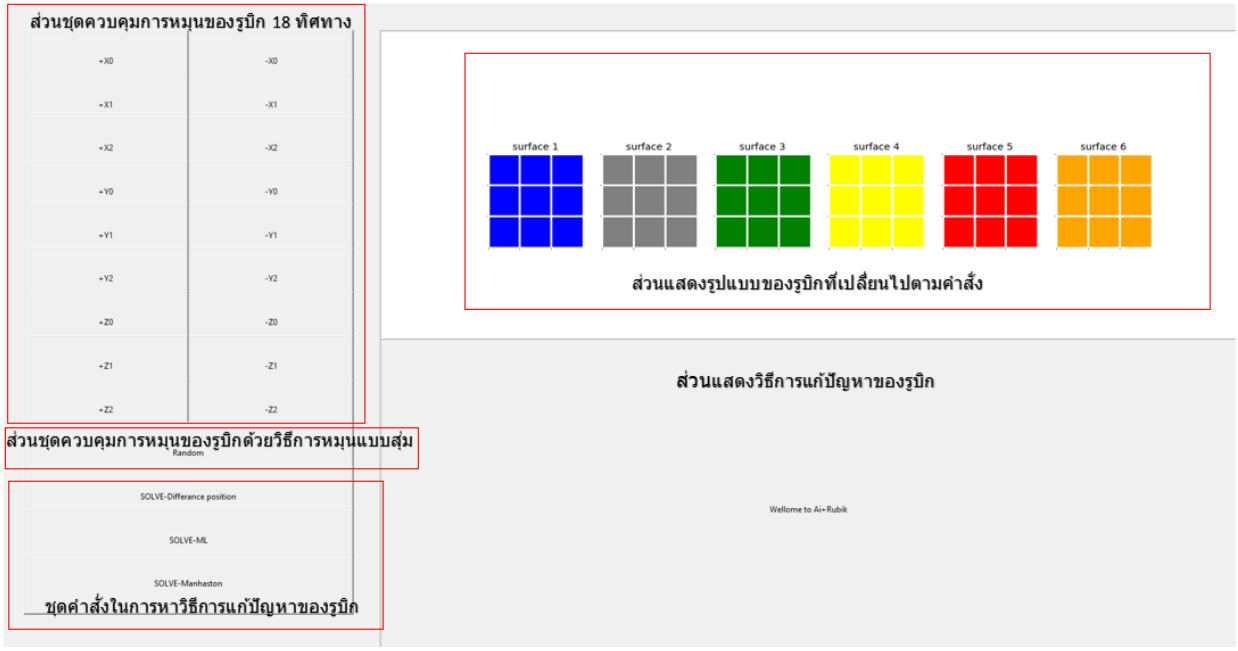
3.1 ส่วนชุดควบคุมการหมุนของรูบิก 18 ทิศทาง

3.2 ส่วนชุดควบคุมการหมุนของรูบิกด้วยวิธีการสุ่ม(ปุ่ม random)

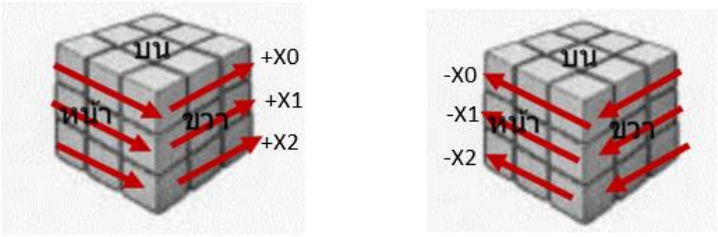
3.3 ชุดคำสั่งในการหาวิธีการแก้ปัญหาของรูบิก(ปุ่ม solve)

3.4 ส่วนแสดง state รูบิค ปัจจุบัน

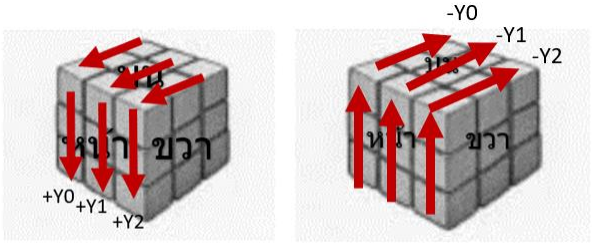
3.5 ส่วนที่แสดงวิธีการแก้ปัญหาของรูบิค



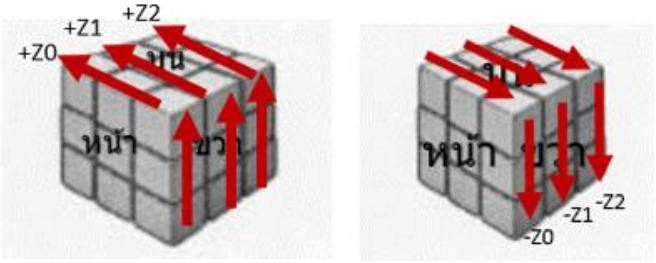
3.1 ส่วนชุดควบคุมการหมุนของรูบิก 18 ทิศทาง



ทิศทาง (+X0,+X1+X2 ,-X0,X1,X2)



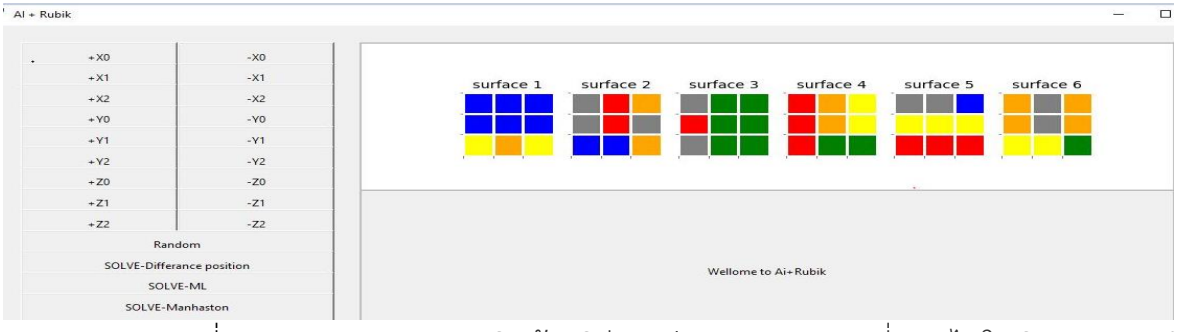
ทิศทาง (+Y0,+Y1+Y2 ,-Y0,Y1,Y2)



ทิศทาง (+Z0,+Z1+Z2 ,-Z0,Z1,Z2)

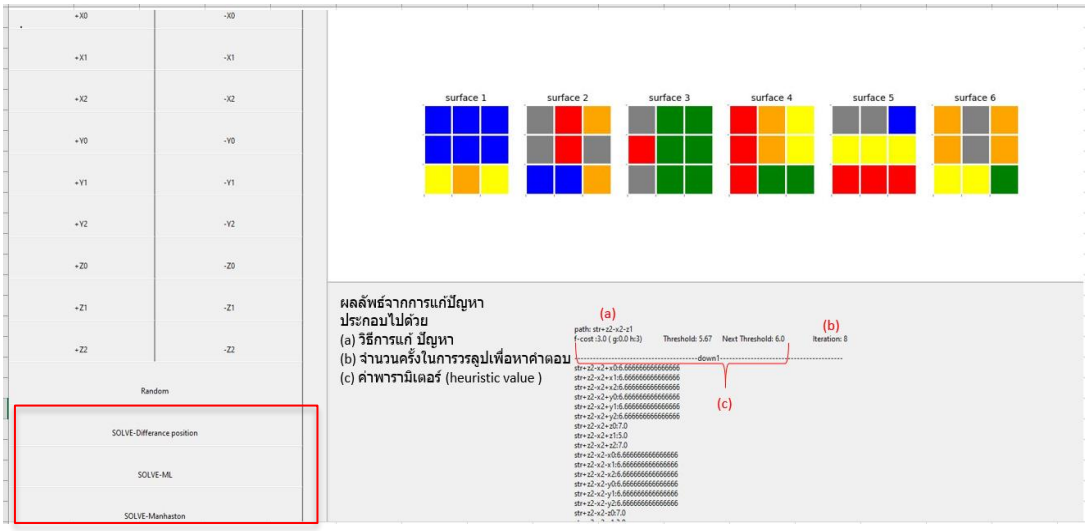
ปุ่ม +X0,-X0,+X1,-X1,+X2,-X2,+X3,-X3,+Y0,-Y0,+Y1,-Y1,+Y2,-Y2,+Z0,-Z0,+Z1,-Z1,+Z2,-Z2 ใช้ในการหมุน rubik ในตำแหน่งต่างๆ

3.2 ส่วนชุดควบคุมการหมุนของรูบิกด้วยวิธีการสุ่ม(ปุ่ม random)



รูปถาพแสดงผลที่ควบคุมการหมุนของรูบิกด้วยวิธีการสุ่ม จะแสดงภาพที่หมุนไปในทิศทางและจำนวนครั้งที่สุ่มขึ้นมา

3.3 ชุดคำสั่งในการหาวิธีการแก้ปัญหาของรูบิก

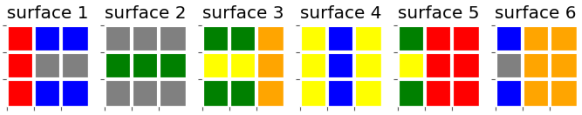


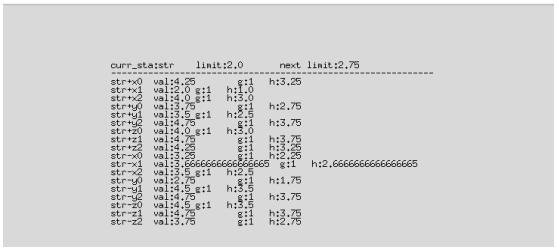
ปุ่ม Solve ใช้ในการแก้ rubik โดยในโปรแกรมนี้จะใช้เทคนิค IDA*

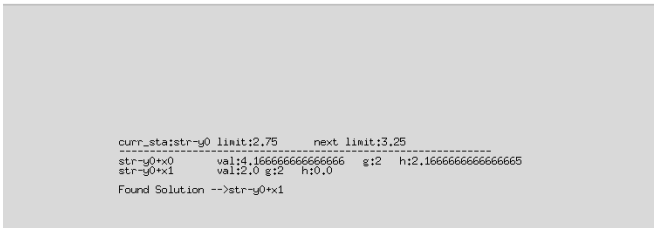
ในการแก้ปัญหาโดยได้ทดลองเขียน function heristic ที่ต่างกันจำนวน 3 วิธี ในการแก้ปัญหา ดังนี้

1. Different Position Surface คำนวน ค่า heristic จากตำแหน่งของสีของชิ้นส่วน rubik ที่อยู่ผิดตำแหน่ง
2. Regression model คำนวนค่า heristic จาก model Machine Learning ในการ predict ค่า heristic จาก state ของ rubik ปัจจุบัน
3. Manhuston distance

4.ค่าต่างๆ ในโปรแกรม







a. บอก State ปัจจุบันว่าของการ Search

b. บอกค่าจากการ expand ของแต่ละ Node ประกอบด้วยค่า g+h

c. บอกค่าของ Thersold(Limit) ของรอบ IDA* ปัจจุบัน

d. บอกค่าของ Thersold (Limit) ของรอบที่จะทำถัดไป

e. บอกค่า min ของ nodes ที่ทำการ expand

f. solution ในการหมุน

Graph แสดงการทำงาน

ตัวอย่างการทดสอบ

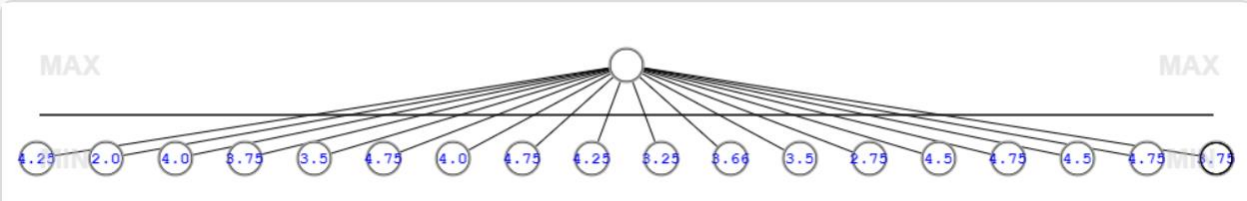
RUN test_rubik_ida.py

Enter Solver method 1.Diiferent positioin 2.Machine Learning 3.Manhuston Distance

>>> 2

Enter move Rubik ex. -x1+y0 >>> -x1+y0

Step1



---

curr_sta: str limit: 2.0 next limit: 2.75

str+x0 4.25 g: 1 h: 3.25

str+x1 2.0 g: 1 h: 1.0

str+x2 4.0 g: 1 h: 3.0

str+y0 3.75 g: 1 h: 2.75

str+y1 3.5 g: 1 h: 2.5

str+y2 4.75 g: 1 h: 3.75

str+z0 4.0 g: 1 h: 3.0

str+z1 4.75 g: 1 h: 3.75

str+z2 4.25 g: 1 h: 3.25

str-x0 3.25 g: 1 h: 2.25

str-x1 3.6666666666666665 g: 1 h: 2.6666666666666665

str-x2 3.5 g: 1 h: 2.5

str-y0 2.75 g: 1 h: 1.75

str-y1 4.5 g: 1 h: 3.5

str-y2 4.75 g: 1 h: 3.75

str-z0 4.5 g: 1 h: 3.5

str-z1 4.75 g: 1 h: 3.75

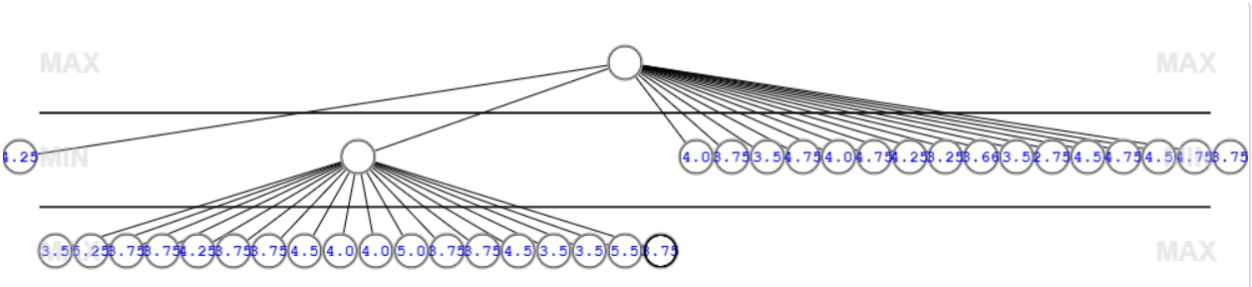
str-z2 3.75 g: 1 h: 2.75

next_node: str+x1

---

node str+x1=2.0 <= limit=2.0 ดังนั้นรอบต่อไปจะดำเนินการ expand node str+x1

Step2



---

curr_sta: str+x1 limit: 2.0 next limit: 2.75

str+x1+x0 3.5 g: 2 h: 1.5

str+x1+x1 5.25 g: 2 h: 3.25

str+x1+x2 3.75 g: 2 h: 1.75

str+x1+y0 3.75 g: 2 h: 1.75

str+x1+y1 4.25 g: 2 h: 2.25

str+x1+y2 3.75 g: 2 h: 1.75

str+x1+z0 3.75 g: 2 h: 1.75

str+x1+z1 4.5 g: 2 h: 2.5

str+x1+z2 4.0 g: 2 h: 2.0

str+x1-x0 4.0 g: 2 h: 2.0

str+x1-x1 5.0 g: 2 h: 3.0

str+x1-x2 3.75 g: 2 h: 1.75

str+x1-y0 3.75 g: 2 h: 1.75

str+x1-y1 4.5 g: 2 h: 2.5

str+x1-y2 3.5 g: 2 h: 1.5

str+x1-z0 3.5 g: 2 h: 1.5

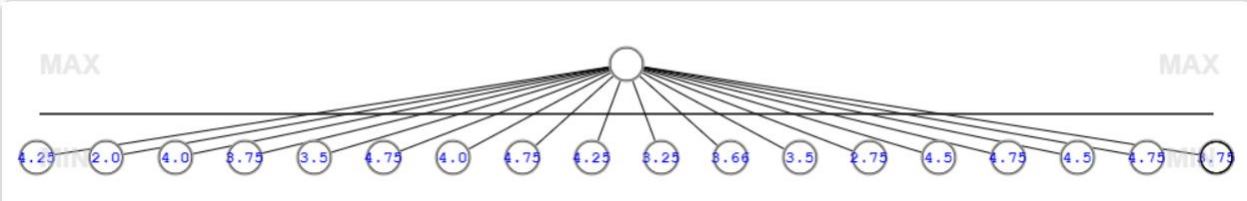
str+x1-z1 5.5 g: 2 h: 3.5

str+x1-z2 3.75 g: 2 h: 1.75

next_node: str

ไม่มี node ไหนที่ <= limit ดังนั้นจะดำเนินการกลับไป state ก่อนหน้า

Step3



---

curr_sta: str limit: 2.0 next limit: 2.75

str+x0 4.25 g: 1 h: 3.25

str+x1 2.0 g: 1 h: 1.0 **เคย expand แล้ว

str+x2 4.0 g: 1 h: 3.0

str+y0 3.75 g: 1 h: 2.75

str+y1 3.5 g: 1 h: 2.5

str+y2 4.75 g: 1 h: 3.75

str+z0 4.0 g: 1 h: 3.0

str+z1 4.75 g: 1 h: 3.75

str+z2 4.25 g: 1 h: 3.25

str-x0 3.25 g: 1 h: 2.25

str-x1 3.6666666666666665 g: 1 h: 2.6666666666666665

str-x2 3.5 g: 1 h: 2.5

str-y0 2.75 g: 1 h: 1.75

str-y1 4.5 g: 1 h: 3.5

str-y2 4.75 g: 1 h: 3.75

str-z0 4.5 g: 1 h: 3.5

str-z1 4.75 g: 1 h: 3.75

str-z2 3.75 g: 1 h: 2.75

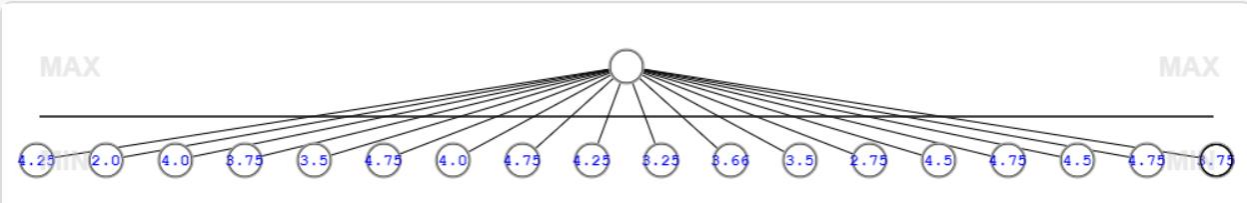
next_node:

next limit

ไม่มี node ใดที่ยังไม่เคย expand มีค่า <= limit และไม่มี node ที่สามารถ expand ได้ต่อไป

ดังนั้นจะดำเนินการ กำหนดใช้ limit ตัวถัดไป

Step4



---

curr_sta: str limit: 2.75 next limit: 9999

str+x0 4.25 g: 1 h: 3.25

str+x1 2.0 g: 1 h: 1.0

str+x2 4.0 g: 1 h: 3.0

str+y0 3.75 g: 1 h: 2.75

str+y1 3.5 g: 1 h: 2.5

str+y2 4.75 g: 1 h: 3.75

str+z0 4.0 g: 1 h: 3.0

str+z1 4.75 g: 1 h: 3.75

str+z2 4.25 g: 1 h: 3.25

str-x0 3.25 g: 1 h: 2.25

str-x1 3.6666666666666665 g: 1 h: 2.6666666666666665

str-x2 3.5 g: 1 h: 2.5

str-y0 2.75 g: 1 h: 1.75

str-y1 4.5 g: 1 h: 3.5

str-y2 4.75 g: 1 h: 3.75

str-z0 4.5 g: 1 h: 3.5

str-z1 4.75 g: 1 h: 3.75

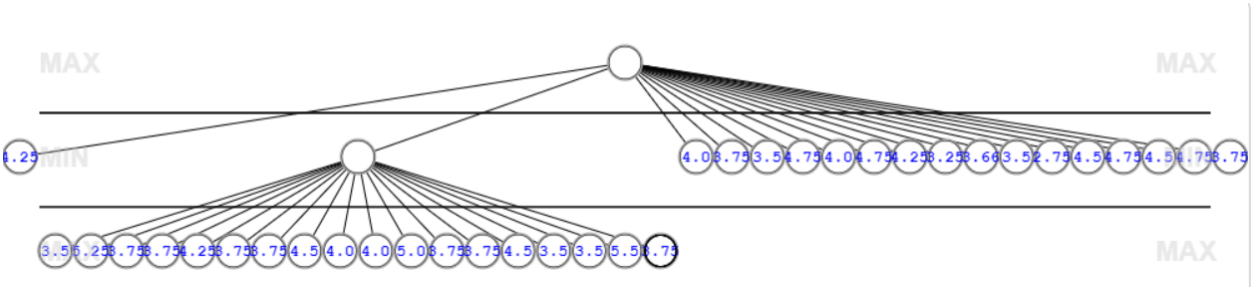
str-z2 3.75 g: 1 h: 2.75

next_node: str+x1

---

node str+x1=2.0 <= limit=2.0 ดังนั้นรอบต่อไปจะดำเนินการ expand node str+x1

Step5



---

curr_sta: str+x1 limit: 2.75 next limit: 3.25

str+x1+x0 3.5 g: 2 h: 1.5

str+x1+x1 5.25 g: 2 h: 3.25

str+x1+x2 3.75 g: 2 h: 1.75

str+x1+y0 3.75 g: 2 h: 1.75

str+x1+y1 4.25 g: 2 h: 2.25

str+x1+y2 3.75 g: 2 h: 1.75

str+x1+z0 3.75 g: 2 h: 1.75

str+x1+z1 4.5 g: 2 h: 2.5

str+x1+z2 4.0 g: 2 h: 2.0

str+x1-x0 4.0 g: 2 h: 2.0

str+x1-x1 5.0 g: 2 h: 3.0

str+x1-x2 3.75 g: 2 h: 1.75

str+x1-y0 3.75 g: 2 h: 1.75

str+x1-y1 4.5 g: 2 h: 2.5

str+x1-y2 3.5 g: 2 h: 1.5

str+x1-z0 3.5 g: 2 h: 1.5

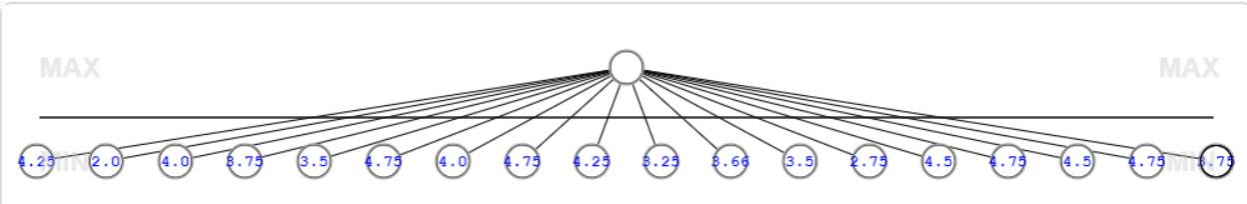
str+x1-z1 5.5 g: 2 h: 3.5

str+x1-z2 3.75 g: 2 h: 1.75

next_node: str

ไม่มี node ไหนที่ <= limit ดังนั้นจะดำเนินการกลับไป state ก่อนหน้า

Step6



---

curr_sta: str limit: 2.75 next limit: 3.25

str+x0 4.25 g: 1 h: 3.25

str+x1 2.0 g: 1 h: 1.0 **เคย expand แล้ว

str+x2 4.0 g: 1 h: 3.0

str+y0 3.75 g: 1 h: 2.75

str+y1 3.5 g: 1 h: 2.5

str+y2 4.75 g: 1 h: 3.75

str+z0 4.0 g: 1 h: 3.0

str+z1 4.75 g: 1 h: 3.75

str+z2 4.25 g: 1 h: 3.25

str-x0 3.25 g: 1 h: 2.25

str-x1 3.6666666666666665 g: 1 h: 2.6666666666666665

str-x2 3.5 g: 1 h: 2.5

str-y0 2.75 g: 1 h: 1.75

str-y1 4.5 g: 1 h: 3.5

str-y2 4.75 g: 1 h: 3.75

str-z0 4.5 g: 1 h: 3.5

str-z1 4.75 g: 1 h: 3.75

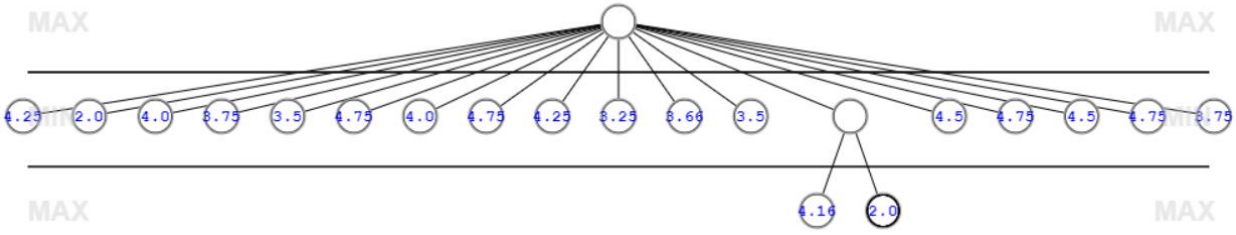
str-z2 3.75 g: 1 h: 2.75

next_node: str-y0

---

node str-y0=2.75 <= limit=2.75 ดังนั้นรอบต่อไปจะดำเนินการ expand node str-y0

Step7



---

curr_sta: str-y0 limit: 2.75 next limit: 3.25

str-y0+x0 4.166666666666666 g: 2 h: 2.1666666666666665

str-y0+x1 2.0 g: 2 h: 0.0

found solution str-y0+x1

next_node: str

---

node str-y0+x1 2.0 g: 2 h: 0.0 มีค่า h = 0.0 แสดงว่า เจอ GOAL!!

เพราะฉะนั้นการแก้รู้บิค คือ str-y0+x1

ประสิทธิภาพโปรแกรม :

พบว่าการคำนวนจากจำนวนความแตกต่างของตำแหน่งที่อยู่ผิดตำแหน่งของทุกๆหน้าจะใช้ได้ดีถ้าระดับความลึกไม่เกิน 5 ครั้ง ขณะที่ค่า **heuristic** ที่ได้จากคำนวนจากการใช้ Machine Learning นั้นสามารถได้ดีกว่าที่ระดับความลึกไม่เกิน 10 ชั้น

นิยามปฏิบัติการ :

IDA*, Heuristics, Manhattan Distance, Machine Learning

Manhattan Distance: หลักการวัดความแตกต่างระหว่างจุด 2 จุดตามหลักโดยไม่คำนึงถึงทิศทาง (เครื่องหมาย **Absolute)** เช่นระยะห่างระหว่างจุด **a (x1, y1)** และ จุด **b (x2, y2)** บนระนาบมีค่าเท่ากับ **|x1 - x2| + |y1 - y2|**

Machine Learning :เป็นการประยุกต์ใช้สถิติขั้นสูงเพื่อเรียนรู้ให้สามารถระบุรูปแบบข้อมูลและคาดการณ์จากรูปแบบเหล่านั้นได้อัลกอริทึมของ **Machine Learning** คือ การให้ ‘ชุดการสอน **(teaching set)**’ ของข้อมูล แล้วใช้ข้อมูลดังกล่าวเพื่อแสวงหาคำตอบ ตัวอย่างเช่น เราอาจเตรียมชุดการสอนเกี่ยวกับรูปภาพ ภาพบางส่วนคือ “นี่คือแมว” ภาพบางส่วน “นี่ไม่ใช่แมว” จากนั้นเราสามารถแสดงภาพชุดใหม่ ๆ สอบถามคอมพิวเตอร์ แล้วให้ **computer** ตอบคำถามเหล่านั้นยกแยะได้ว่ารูปใดเป็นรูปแมว

Heuristics: เป็นเทคนิคการค้นหาแบบมีข้อมูล (**informed)** การค้นหาจะนำข้อมูลมาประกอบเพื่อช่วยเพิ่มประสิทธิภาพ ฟังก์ชันพื้นฐานที่นำมาใช้ประกอบกับการค้นหาแบบ **Heuristic** มี 2 ชนิด ฟังก์ชัน **Evaluation (Evaluation function f(n))** ทำหน้าที่ประมาณค่าใช้จ่ายทั้งหมดบนเส้นทางจากโหนด **n** ไปยังโหนดเป้ าหมาย ฟังก์ชัน **Heuristic (Heuristic function h(n))** ทำหน้าที่บอกปริมาณทรัพยากรที่ใช้ไปตั้งแต่ตำแหน่ง **n** จนถึงเป้าหมาย

IDA* ((iterative deepening A*) การค้นหาเชิงลึกแบบวนเพิ่มความลึกด้วยเทคนิค **A*** เป็นขั้นตอนวิธีสำหรับการค้นปริภูมิสถานะ (**state space search)** ที่อาศัยการค้นหาเชิงลึกแบบวนเพิ่มความลึกไปเรื่อยๆตามหลักการ **iterative deepening**แต่จะใช้เทคนิค **A*** มาคอยประเมินผลเพื่อพิจารณาการเพิ่มระดับความลึกของการค้นหาในแต่ละรอบจนไปถึงเป้าหมายที่ต้องการ

บรรณานุกรม

1.A study of how to solve the Rubik's cube using two popular approaches: the Thistlewaite's algorithm and the IDA* algorithm (Harpreet Happy Kaur ,2015)

2.A*+IDA*: A Simple Hybrid Search Algorithm (Zhaoxing Bu and Richard E. Korf , IJCAI, 2019)

3. Predicting the performance of IDA* using conditional distributions (UZahavi,A Felner, Journal of ArtificialIntelligence,2010)