

八上 逻辑作业

1. COSTLESSTHAN (OMELETTES, PANCAKES)
2. ^(3x) [COST (AT (MEAL(x) the RED LION), THREE POUNDS)]
3. ~~(4x)~~ [COST (AT (MEAL(x), the RED LION), THREE POUNDS)]

1.

日期: 猴子香蕉问题. (红色为求解过程)

初始: AT(money, a)	目标: AT(money, c)
AT(box, b)	AT(box, c)
\neg ONBOX	ONBOX
\neg HB	HB.

定义: EMPTY(x) x的手中是空的

AT(t, y) t在y处

CLEAR(y) y上是空的

BOX(n) 箱子n \in box

BANANA(v) 香蕉v \in banana

个体域: x \in money

t \in box, banana

y \in a, b, c, ceiling

需要定义的操作:

WALK(m, n): 猴子从m走到n处, m, n \in a, b, c

CARRY(s, r): 猴子从r处拿到s, r \in b, ceiling,

s \in box, banana

2.

12. 开始:
CLIMB(m,b): 猴子在 b 处爬上 u

AT(money, a)

AT(money, b)

开始: EMPTY(money) WALK(a,b) EMPTY(money)

AT(box, b)

AT(box, b)

AT(banana, ceiling)

AT(banana, ceiling)

→ ONBOX

→ ONBOX

→ HB.

→ HB.

CARRY(b, box) → AT(money, b) WALK(b,c) → AT(money, c)

AT(banana, ceiling)

EMPTY(money)

→ ONBOX

AT(box, c)

HB (hold box)

AT(banana, ceiling)

→ ONBOX

CLIMB(box, c) → AT(money, box)

→ HB.

EMPTY(money) → CARRY → AT(money, c)

AT(box, c)

↙ (banana, ceiling)

AT(box, c)

AT(banana, ceiling)

ONBOX

ONBOX

HB.

→ HB

定义:

$\text{Like}(x, y)$ x 喜欢 y 课程

$\text{Easy}(x)$ x 是简单的课程

$\text{Eng}(x)$ x 是工程类课程

$\text{Phy}(x)$ x 是物理类课程

故条件为:

1) $\text{Easy}(x) \rightarrow \text{Like}(\text{Li}, x)$

2) $\neg \text{Easy}(x) \rightarrow \neg \text{Like}(\text{Li}, x)$

3) $\text{Eng}(x) \rightarrow \neg \text{Easy}(x)$

4) $\text{Phy}(x) \rightarrow \neg \text{Easy}(x)$

5) $\neg \text{Like}(\text{Li}, x) \rightarrow \text{Like}(\text{Wu}, x)$

6) $\text{Phy}(\text{Phy200})$

7) $\text{Eng}(\text{Eng300})$

求: ① $\text{Like}(\text{Li}, x)$ ② $\text{Like}(\text{Wu}, \text{Eng300})$

化简为子句集

C1: $\neg \text{Easy}(x) \vee \text{Like}(\text{Li}, x)$

C2: $\text{Easy}(x) \vee \neg \text{Like}(\text{Li}, x)$

C3: $\neg \text{Eng}(x) \vee \neg \text{Easy}(x)$

C4: $\neg \text{Phy}(x) \vee \neg \text{Easy}(x)$

C5: $\text{Like}(\text{Li}, x) \vee \text{Like}(\text{Wu}, x)$

C6: $\text{Phy}(\text{Phy200})$

C7: $\text{Eng}(\text{Eng300})$

T1: $\neg \text{Like}(\text{Li}, x) \vee \text{ANSWER}(x)$

T2: $\neg \text{Like}(\text{Wu}, \text{Eng300})$

① 求解 $T_1: \neg \text{Like}(Li, v) \vee \text{Like}(Li, v)$ (v 为变量)
 $\neg \text{Like}(Li, v) \vee \text{Like}(Li, v) \rightarrow \text{Easy}(x) \vee \neg \text{Like}(Li, x)$
 $\{v/x\}$
 $\neg \text{Easy}(x) \vee \text{Like}(Li, v) \quad \neg \text{Phy}(x) \vee \text{Easy}(x)$
 $\{v/x\}$
 $\neg \text{Phy}(v) \vee \text{Like}(Li, v) \quad \text{Phy}(Phy200)$
 $\text{Phy200}/v$
 $\text{Like}(Li, \text{Phy200})$

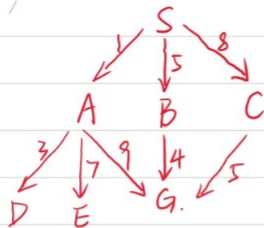
由结论可知, 小李喜欢 Phy200 这门课.

② 求证 $T_2: \neg \text{Like}(Wu, \text{Eng300})$
 $\neg \text{Like}(Wu, \text{Eng300}) \quad \text{Like}(Li, x) \vee \text{Like}(Wu, x)$
 $\{ \text{Eng300}/x \}$
 $\text{Like}(Li, \text{Eng300}) \quad \text{Easy}(x) \vee \neg \text{Like}(Li, x)$
 $\{ \text{Eng300}/x \}$
 $\text{Easy}(\text{Eng300}) \quad \neg \text{Eng}(x) \vee \neg \text{Easy}(x)$
 $\{ \text{Eng300}/x \}$
 $\neg \text{Eng}(\text{Eng300}) \quad \text{Eng}(\text{Eng300})$
 NIL

得证 小吴喜欢 Eng300 这门课.

日期: /

题目:



① DFS

Expand. node	Open list
	{S}
S not goal	{A, B, C}
A not goal	{D, E, G, B, C}
D not goal	{E, G, B, C}
E not goal	{G, B, C}
G goal	{B, C} no expand

最终路径: $S \rightarrow A \rightarrow G$.

② BFS

Expand. node	Open list
	{S}
S not goal	{A, B, C}

A not goal	{B, C, D, E, G}
B not goal	{C, D, E, G, G}
C not goal	{D, E, G, G, G}
D not goal	{E, G, G, G}
E not goal	{G, G, G}
G goal	{G, G} no expand

最终路径: $S \rightarrow A \rightarrow G$.

③ UCS

Expand node	Open list
	{S:0}
S not goal	{A:1, B:1, C:8}
A not goal	{D:4, B:5, E:8, C:8, G:10}
D not goal	{B:3, E:8, C:8, G:10}
B not goal	{E:8, C:8, G:10, G:9}
E not goal	{C:8, G:9, G:10}
C not goal	{G:9, G:10, G:13}
G goal	{G:10, G:13} no expand

最终路径: $S \rightarrow B \rightarrow G$.

④ IDS

depth: 1

Expand node	Frontier list
	{S}
S not goal	{A, B, C}
A not goal	{B, C}
B not goal	{C}
C not goal	{ } no expand - FAIL

depth: 2

Expand node	Frontier list
	{S}
S not goal	{A, B, C}
A not goal	{B, C}
B not goal	{C}
C not goal	{ }

S	no test	{A, B, C}
A	no test	{D, E, G, B, C}
D	not goal	{E, G, B, C}
E	not goal	{G, B, C}
G	goal	{B, C} no expand.

最终路径: $S \rightarrow A \rightarrow G$.



中南大學
CENTRAL SOUTH UNIVERSITY

《人工智能》实验报告

学生姓名	张文睿
学 号	8207211004
专业班级	计科 2101
指导教师	黄芳
学 院	计算机学院
完成时间	2023.5.4

计算机学院

2023 年 5 月

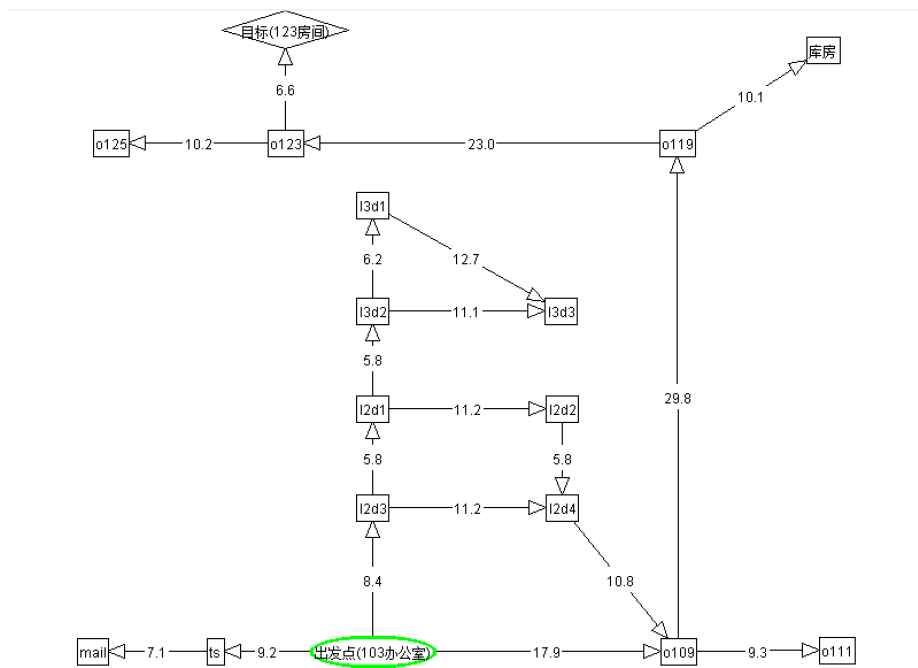
实验一 搜索策略

一、实验目的

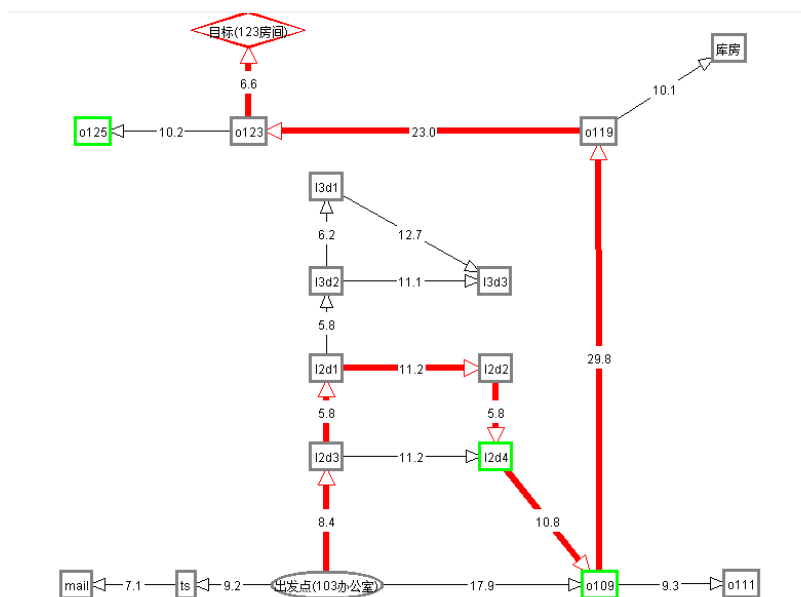
熟悉和掌握各种启发式搜索策略的思想，掌握 A*算法的定义、估价函数和算法过程，理解求解流程和搜索顺序。

二、算法比较

图的搜索



1) DFS



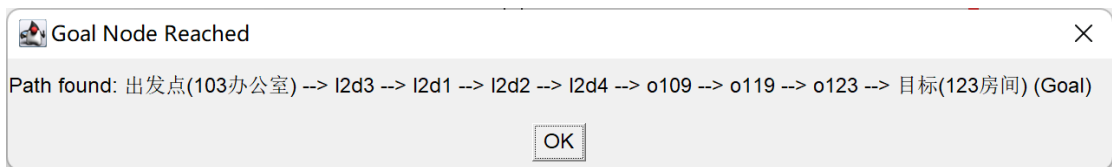
表格如下：

Open 表	Close 表	估价函数
{出发点}	{ }	无
{ts,o109,12d3}	{出发点}	
{mail,o109,12d3}	{出发点,ts}	
{ o109,12d3}	{出发点,ts,mail}	
{o109,12d1,12d4}	{出发点,ts,mail ,12d3}	
{ o109,12d4,13d2,12d2}	{出发点,ts,mail ,12d3,12d1}	
{ o109,12d4,12d2,13d1,13d3}	{出发点,ts,mail ,12d3,12d1, 13d2}	
{ o109,12d4,12d2,13d3}	{出发点,ts,mail ,12d3,12d1, 13d2,13d1}	
{ o109,12d4,12d2}	{出发点,ts,mail ,12d3,12d1, 13d2,13d1,13d3}	
{ o109,12d4,12d2}	{出发点,ts,mail ,12d3,12d1, 13d2,13d1,13d3}	
{12d4,o109}	{出发点,ts,mail ,12d3,12d1, 13d2,13d1,13d3}	
{o109}	{出发点,ts,mail ,12d3,12d1,13d2,13d1,13d3,12d2,12d4}	
{ o111,o119}	{出发点,ts,mail ,12d3,12d1, 13d2,13d1,13d3,12d2,o109,12d4}	
{o119}	{出发点,ts,mail ,12d3,12d1,13d2,13d1,13d3,12d2,12d4}	
{o123,库房}	{出发点,ts,mail ,12d3,12d4,12d1,13d2,13d1,13d3,12d2.o119}	
{ o123}	{出发点,ts,mail ,12d3,12d4,12d1,13d2,13d1,13d3,12d2.o119,库房}	
{o125,123 房间}	{出发点,ts,mail ,12d3,12d4,12d1,13d2,13d1,13d3,12d2.o119,库房,o123}	
GOAL		

搜索路径：

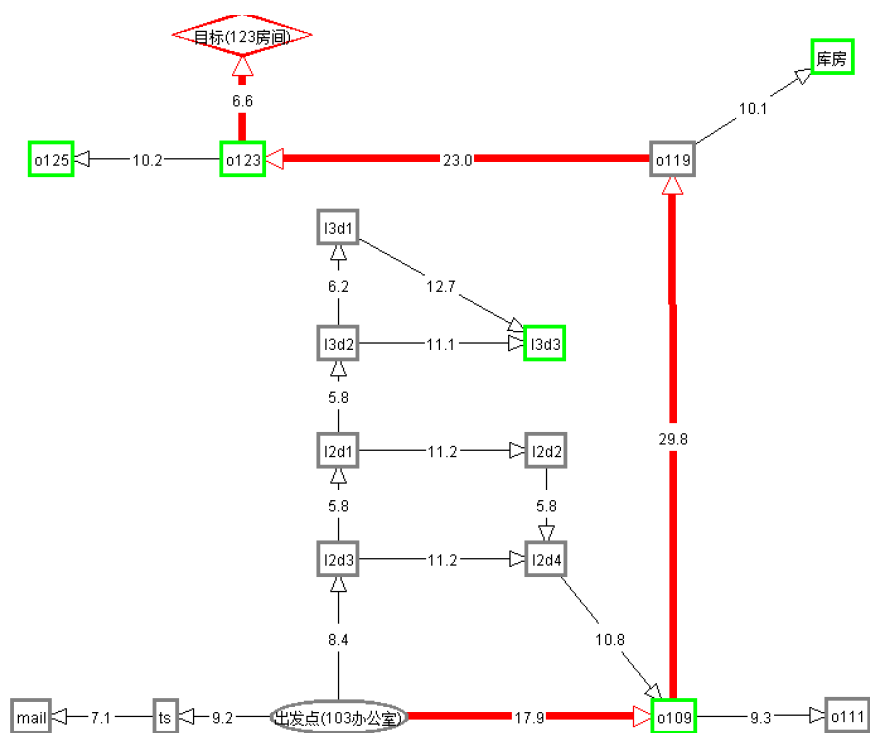
出发点->ts->mail->12d3->12d1->13d2->13d1->13d3->12d2->12d4
->o109->o111->o119->库房->o123->GOAL

最终路径如下：



学生结论：DFS 使用递归，并且一般使用类似栈的数据结构依次访问所有节点，俗称“一条路走到黑”。当搜索空间很大的时候，搜索效率和图的储存结构有一定关系，如果陷入一条不能通行的分支会导致效率变差；如果图中有环，容易陷入循环无法继续；并且找到的路径不一定是最优路径。

2) BFS



表格如下：

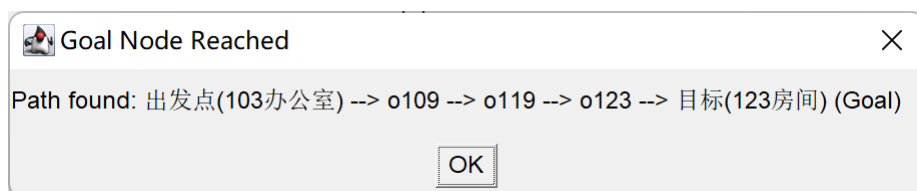
Open	Close	估价函数
{出发点}	{ }	无
{ts,o109,l2d3}	{出发点}	
{o109,l2d3,mail}	{出发点,ts}	
{ l2d3,mail,o111,o119}	{出发点,ts,o109 }	

{ mail,o111,o119,l2d1,l2d4}	{出发点,ts,o109,l2d3}	
{ o111,o119,l2d1,l2d4}	{出发点,ts,o109,l2d3,mail}	
{ o119,l2d1,l2d4}	{出发点,ts,o109,l2d3,mail,o111}	
{ l2d1,l2d4,o123,库房}	{出发点,ts,o109,l2d3,mail,o111,o119}	
{ l2d4,o123,库房,l3d2,l2d2}	{出发点,ts,o109,l2d3,mail,o111,o119,l2d1}	
{ o123,库房,l3d2,l2d2,o109}	{出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4}	
{库房,l3d2,l2d2,o109,目标,o125}	{出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4,o123}	
{l3d2,l2d2,o109,目标,o125}	{出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4,o123,库房}	
{l2d2,o109,目标,o125,l3d1,l3d3}	{出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4,o123,库房,l3d2}	
{o109,目标,o125,l3d1,l3d3,l2d4}	{出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4,o123,库房,l3d2,l2d2}	
{目标,o125,l3d1,l3d3,l2d4,o111,o119}	{出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4,o123,库房,l3d2,l2d2}	
GOAL		

搜索路径:

出发点 -> ts -> o109 -> l2d3 -> mail -> o111 -> o119 -> l2d1 -> l2d4 -> o123 -> 库房 -> l3d2 -> l2d2 -> o109 -> 目标

最终路径如下:

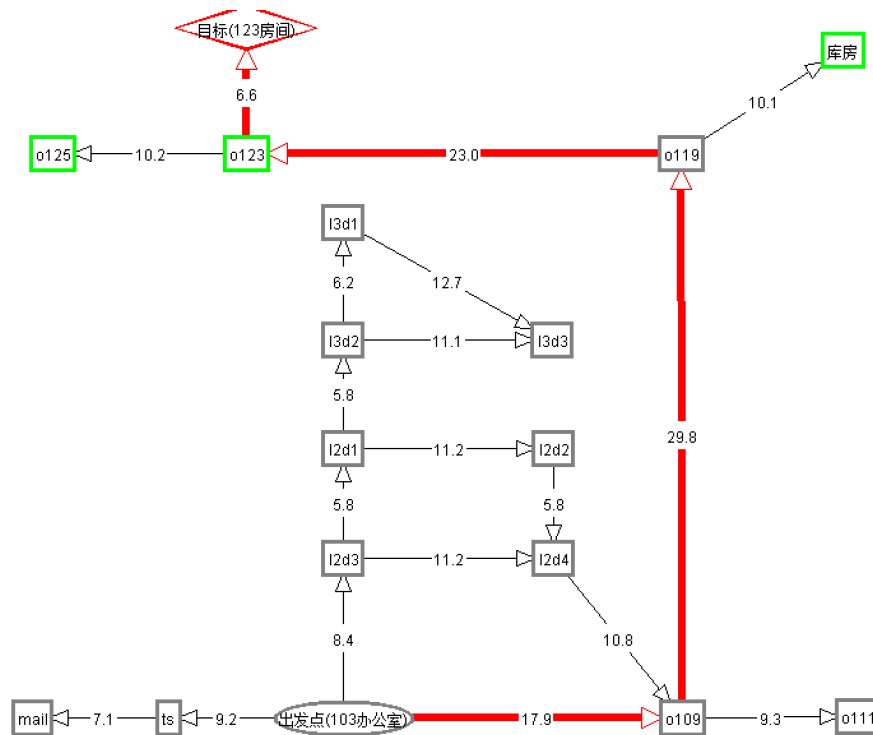


学生结论:

BFS 使用队列的数据结构访问所有节点，也是属于穷举类型的盲目搜索，当有循环时依旧可以进行搜索，但是当图的节点

较多时搜索速度会减慢，效率较低。

3) lowest cost(Dijkstra 算法)



表格如下：

Open	Close	估价函数
{出发点}	{ }	$f(n)=g(n)$
{ts: 9. 2, o109: 17. 9, l2d3: 8. 4}	{出发点}	
{ts: 9. 2, o109: 17. 9, l2d1: 14. 2, l2d4: 19. 6}	{出发点, l2d3}	
{o109: 17. 9, l2d1: 14. 2, l2d4: 19. 6, mail: 16. 3}	{出发点, l2d3, ts}	
{o109: 17. 9, l2d4: 19. 6, mail: 16. 3, l3d2: 20. 0, l2d2: 25. 4}	{出发点, l2d3, ts, l2d1}	
{o109: 17. 9, l2d4: 19. 6, l3d2: 20. 0, l2d2: 25. 4}	{出发点, l2d3, ts, l2d1, mail}	
{l2d4: 19. 6, l3d2: 20. 0, l2d2: 25. 4, o111: 27. 2, o119: 47. 7}	{出发点, l2d3, ts, l2d1, mail, o109}	
{l3d2: 20. 0, l2d2: 25. 4, o111: 27. 2, o119: 47. 7, o109: 30. 4}	{出发点, l2d3, ts, l2d1, mail, o109, l2d4}	

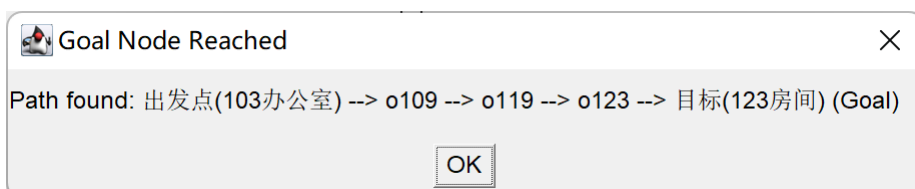
{l2d2:25.4,o111:27.2,o119:47.7,o109:30.4,l3d1:26.2,l3d3:31.1}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2}	
{o111:27.2,o119:47.7,o109:30.4,l3d1:26.2,l3d3:31.1,l2d4:31.2,l3d3:38.9}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2}	
{o111:27.2,o119:47.7,o109:30.4,l3d3:31.1,l2d4:31.2,l3d3:38.9}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1}	
{o119:47.7,o109:30.4,l3d3:31.1,l2d4:31.2,l3d3:38.9}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111}	
{o119:47.7,l3d3:31.1,l2d4:31.2,l3d3:38.9,o111:39.7}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3}	
{o119:47.7,l2d4:31.2,l3d3:38.9,o111:39.7}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3}	
{o119:47.7,o109:42.0,l3d3:38.9,o111:39.7,o109:42.0}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3}	
{o119:47.7,o111:39.7,l3d3:38.9,o119:60.2,o109:42.0,o119:71.8}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3}	
{o111:39.7,l3d3:38.9,o119:60.2,o109:42.0,o119:71.8}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119}	
{o111:39.7,o119:60.2,,o109:42.0,o119:71.8,库房:57.8,o123:70.7}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119}	
{o119:60.2,o109:42.0,o119:71.8,库房:57.8,o123:70.7,o111:51.3}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119}	
{o119:60.2,o119:71.8,库房:57.8,o123:70.7,o111:51.3}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119}	
{o119:60.2,o119:71.8,库房:57.8,o123:70.7}	{出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l	

{o119:60.2,o119:71.8,o123:70.7}	2d2,13d1,o111,13d3,0119} {出发点,12d3,ts,12d1,mail,o109,12d4,13d2,12d2,13d1,o111,13d3,0119,库房}	
{o119:71.8,o123:70.7,库房:70.3,o123:83.2}	{出发点,12d3,ts,12d1,mail,o109,12d4,13d2,12d2,13d1,o111,13d3,0119,库房}	
{o119:71.8,o123:70.7,o123:83.2}	{出发点,12d3,ts,12d1,mail,o109,12d4,13d2,12d2,13d1,o111,13d3,0119,库房}	
{o119:71.8,o123:83.2,目标:77.3,o125:80.9}	{出发点,12d3,ts,12d1,mail,o109,12d4,13d2,12d2,13d1,o111,13d3,0119,库房,o123}	
{o123:83.2,目标:77.3,o125:80.9,库房:81.9}	{出发点,12d3,ts,12d1,mail,o109,12d4,13d2,12d2,13d1,o111,13d3,0119,库房,o123}	
GOAL	{出发点,12d3,ts,12d1,mail,o109,12d4,13d2,12d2,13d1,o111,13d3,0119,库房,o123}	

搜索路径:

出发点 -> 12d3 -> ts -> 12d1 -> mail -> o109 -> 12d4 -> 13d2 -> 12d2 -> 13d1 -> o111 -> o109 -> 13d3 -> 12d4 -> 13d3 -> o111 -> o109 -> o119 -> o111 -> 库房 -> o119 -> 库房 -> o123 -> o119 -> 目标

最终路径如下:

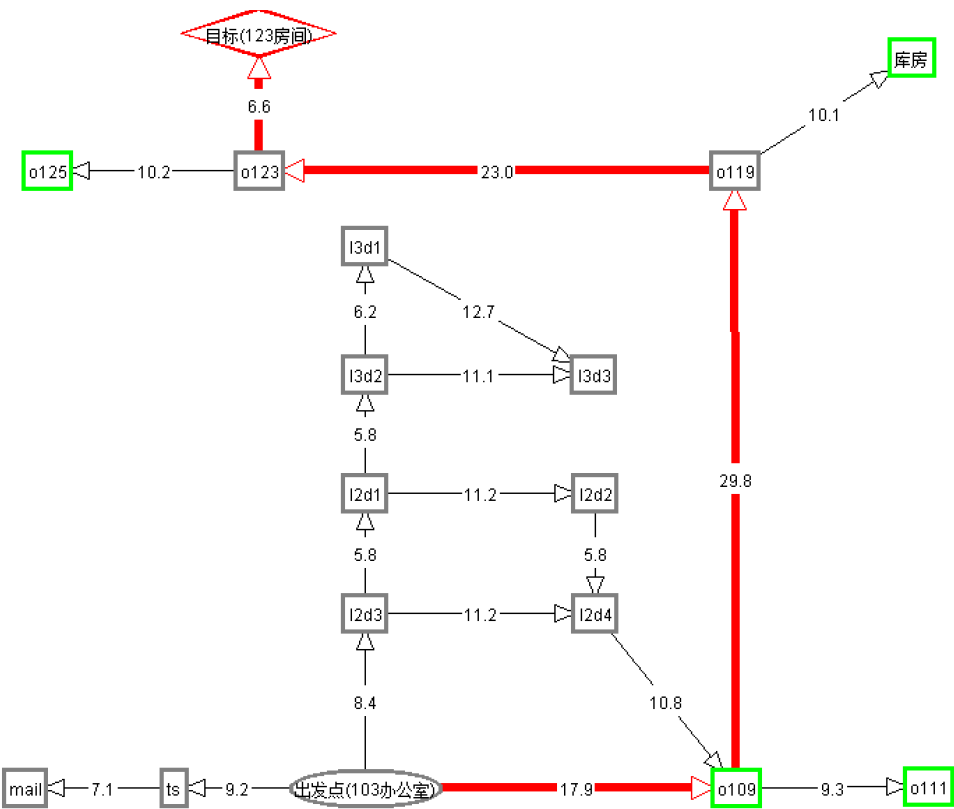


学生结论: 最短路径搜索是在每次记录起点到此刻的搜索代价, 寻找代价最小的节点往下搜索。这样可以保证最终找到的路径是

最短的。但是当搜索路径复杂或是有环时，搜索容易绕圈并且降低，算法效率会大大降低。

4) Best First(贪婪)

○



表格如下：

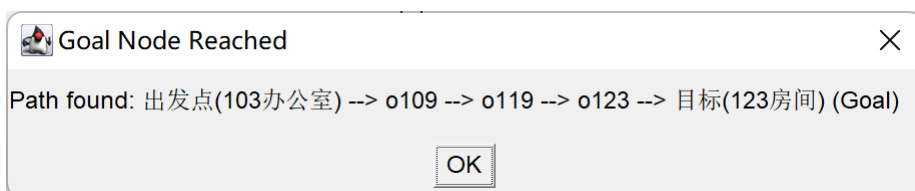
Open	Close	估价函数
{出发点}	{ }	$f(n)=h(n)$
{ts: 36. 6, o109: 43. 1, l2d3: 28. 4}	{出发点}	
{ts: 36. 6, o109: 43. 1, l2d1: 22. 8, l2d4: 32. 4}	{出发点, l2d3}	
{ts: 36. 6, o109: 43. 1, l2d4: 32. 4, l3d2: 17. 2, l2d2: 27. 5}	{出发点, l2d3, l2d1}	
{ts: 36. 6, o109: 43. 1, l2d4: 32. 4, l2d2: 27. 5, l3d1: 11. 4, l3d3: 23. 0}	{出发点, l2d3, l2d1, l3d2}	
{ts: 36. 6, o109: 43. 1, l2d4: 32. 4, l2d2: 27. 5, l3d3: 23. 0, l3d3: 23. 0}	{出发点, l2d3, l2d1, l3d2, l3d1}	

{ts:36.6,o109:43.1,12d4:32.4,12d2:27.5,13d3:23.0}	{出发点,12d3,12d1,13d2,13d1,13d3}	
{ts:36.6,o109:43.1,12d4:32.4,12d2:27.5}	{出发点,12d3,12d1,13d2,13d1,13d3}	
{ts:36.6,o109:43.1,12d4:32.4,12d4:32.4}	{出发点,12d3,12d1,13d2,13d1,13d3,12d2}	
{ts:36.6,o109:43.1,12d4:32.4}	{出发点,12d3,12d1,13d2,13d1,13d3,12d2,12d4}	
{ts:36.6,o109:43.1}	{出发点,12d3,12d1,13d2,13d1,13d3,12d2,12d4}	
{o109:43.1,mail:38.1}	{出发点,12d3,12d1,13d2,13d1,13d3,12d2,12d4,ts}	
{o109:43.1}	{出发点,12d3,12d1,13d2,13d1,13d3,12d2,12d4,ts,mail}	
{o111:48.6,o119:23.9}	{出发点,12d3,12d1,13d2,13d1,13d3,12d2,12d4,ts,mail,o109}	
{o111:48.6,库房:31.5,o123:6.6}	{出发点,12d3,12d1,13d2,13d1,13d3,12d2,12d4,ts,mail,o109,o119}	
{o111:48.6,库房:31.5,目标:0}	{出发点,12d3,12d1,13d2,13d1,13d3,12d2,12d4,ts,mail,o109,o119,o123}	
GOAL		

搜索路径:

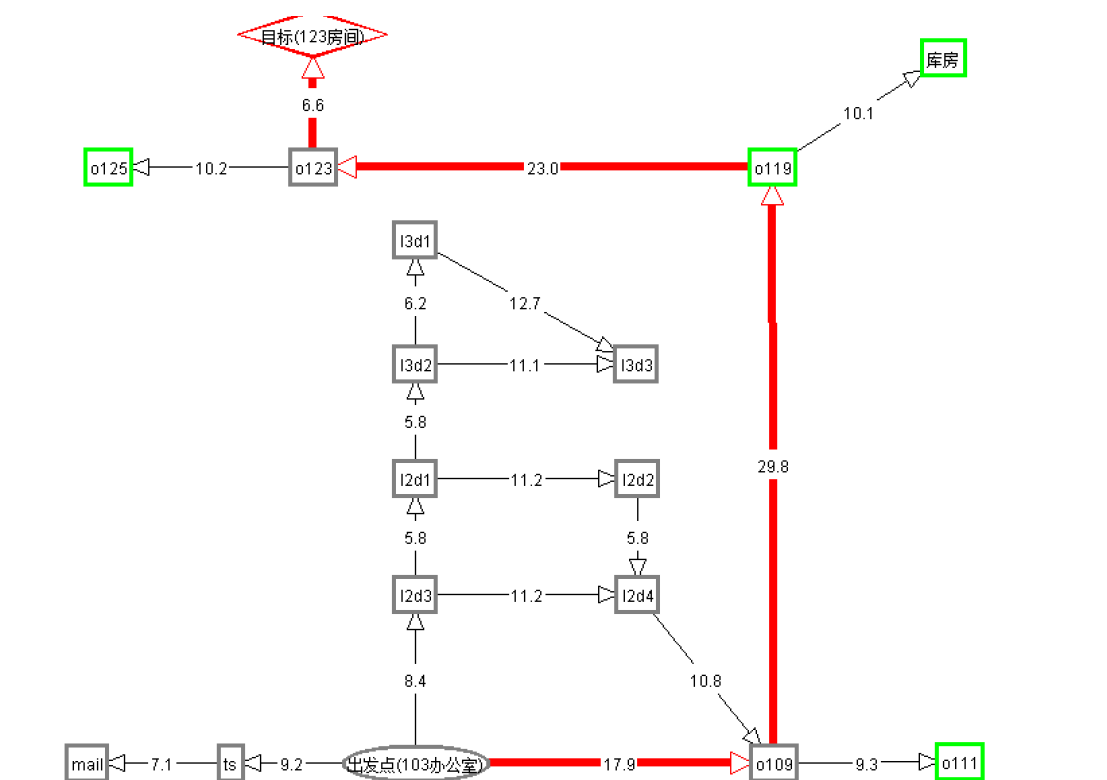
出发点--> 12d3--> 12d1--> 13d2--> 13d1--> 13d3--> 12d2--> 12d4--> ts --> mail--> o109--> o119--> o123

最终路径如下:



学生结论：贪婪搜索，即只扩展当前代价最小的节点(或者说离当前节点最近的点)，也是一种启发式搜索。这样做的缺点就是，目前代价小，之后的代价不一定小，如果解在代价最大的点，那么按照贪婪最佳优先算法，可能就找不到这个解，然后就会陷入死循环。

5) A^*



表格如下：

Open	Close	估价函数
{出发点}	{ }	$f(n) = g(n) + h(n)$
{ts: 45.8, o109: 61.0, l2d3: 36.8}	{出发点}	
{ts: 45.8, o109: 61.0, l2d1: 37.0, l2d4: 52.0}	{出发点, l2d3}	
{ts: 45.8, o109: 61.0, l2d4: 52.0, l3d2: 37.2, l2d2: 52.9}	{出发点, l2d3, l2d1}	
	{出发点, l2d3, l2d1, l3d2}	

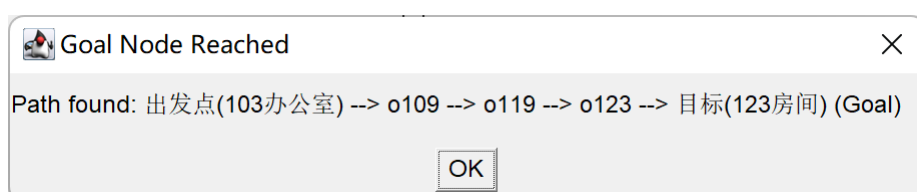
{ts:45.8,o109:61.0,12d4:52.0,12d2:52.9,13d1:37.6,13d3:54.1} {ts:45.8,o109:61.0,12d4:52.0,12d2:52.9,13d3:54.1,13d3:61.9} {mail:54.4,o109:61.0,12d4:52.0,12d2:52.9,13d3:54.1,13d3:61.9} {mail:54.4,o109:61.0,12d2:52.9,13d3:54.1,13d3:61.9,o109:73.5} {mail:54.4,o109:61.0,13d3:54.1,13d3:61.9,o109:73.5,12d4:63.6} {mail:54.4,o109:61.0,13d3:61.9,o109:73.5,12d4:63.6} {o109:61.0,13d3:61.9,o109:73.5,12d4:63.6} {13d3:61.9,o109:73.5,12d4:63.6,o111:75.8,o119:71.6} {o109:73.5,12d4:63.6,o111:75.8,o119:71.6} {o109:73.5,o111:75.8,o119:71.6,o109:85.1} {o109:73.5,o111:75.8,o109:85.1,库房:89.3,o123:77.3} {o111:75.8,o109:85.1,库房:89.3,o123:77.3,o111:88.3,o119:84.1} {o109:85.1,库房:89.3,o123:77.3,o111:88.3,o119:84.1} {o109:85.1,库房:89.3,o111:88.3,o119:84.1,o125:93.0,目标:77.3}	{出发点,12d3,12d1,13d2,13d1} {出发点,12d3,12d1,13d2,13d1,ts} {出发点,12d3,12d1,13d2,13d1,ts,12d4} {出发点,12d3,12d1,13d2,13d1,ts,12d4,12d2} {出发点,12d3,12d1,13d2,13d1,ts,12d4,12d2,13d3} {出发点,12d3,12d1,13d2,13d1,ts,12d4,12d2,13d3,mail} {出发点,12d3,12d1,13d2,13d1,ts,12d4,12d2,13d3,mail,o109} {出发点,12d3,12d1,13d2,13d1,ts,12d4,12d2,13d3,mail,o109} {出发点,12d3,12d1,13d2,13d1,ts,12d4,12d2,13d3,mail,o109} {出发点,12d3,12d1,13d2,13d1,ts,12d4,12d2,13d3,mail,o109,o119} {出发点,12d3,12d1,13d2,13d1,ts,12d4,12d2,13d3,mail,o109,o119} {出发点,12d3,12d1,13d2,13d1,ts,12d4,12d2,13d3,mail,o109,o119} {出发点,12d3,12d1,13d2,13d1,ts,12d4,12d2,13d3,mail,o109,o119,o123}	
--	---	--

GOAL		
------	--	--

搜索路径：

出发点--> 12d3--> 12d1--> 13d2--> 13d1--> ts--> 12d4--> 12d2--> 13d3 --> mail--> o109--> 13d3--> 12d4-->o119-->o109-->o111-->o123

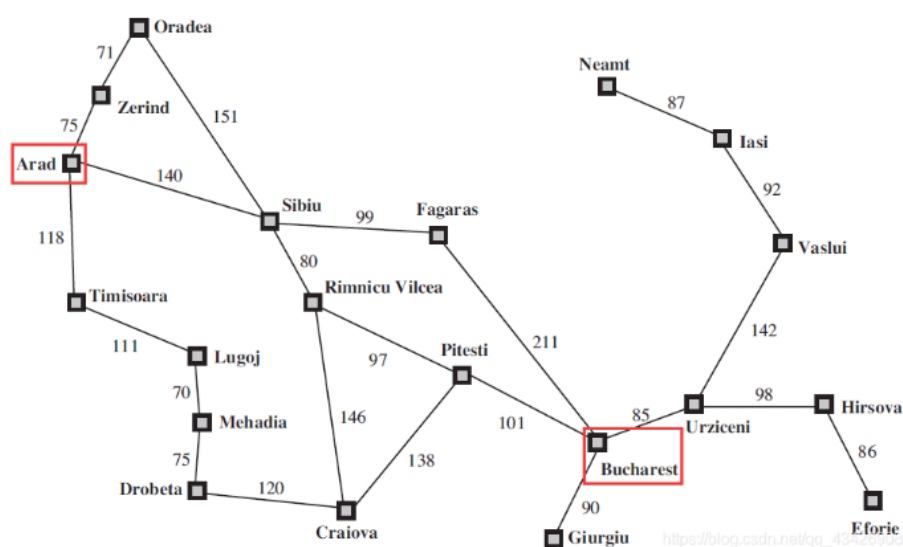
最终路径如下：



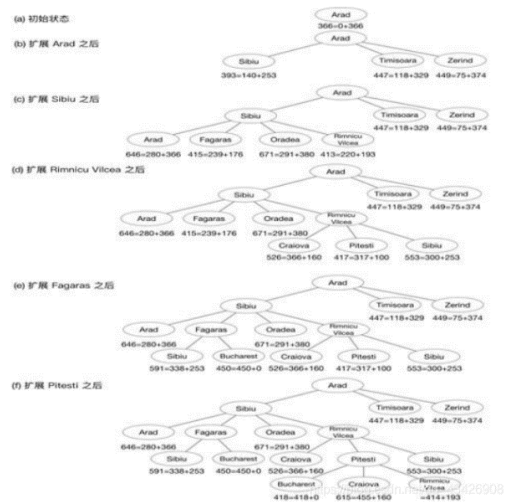
学生结论：A*是一种启发式搜索算法、一种静态路网中求解最短路径最有效的直接搜索方法，也是解决许多搜索问题的有效算法。它兼顾了搜索效率和可靠性，可以根据不同的情况使用不同的启发函数来改变搜索策略，针对每个图找到最优的搜索方式。

三、修改启发函数的比较

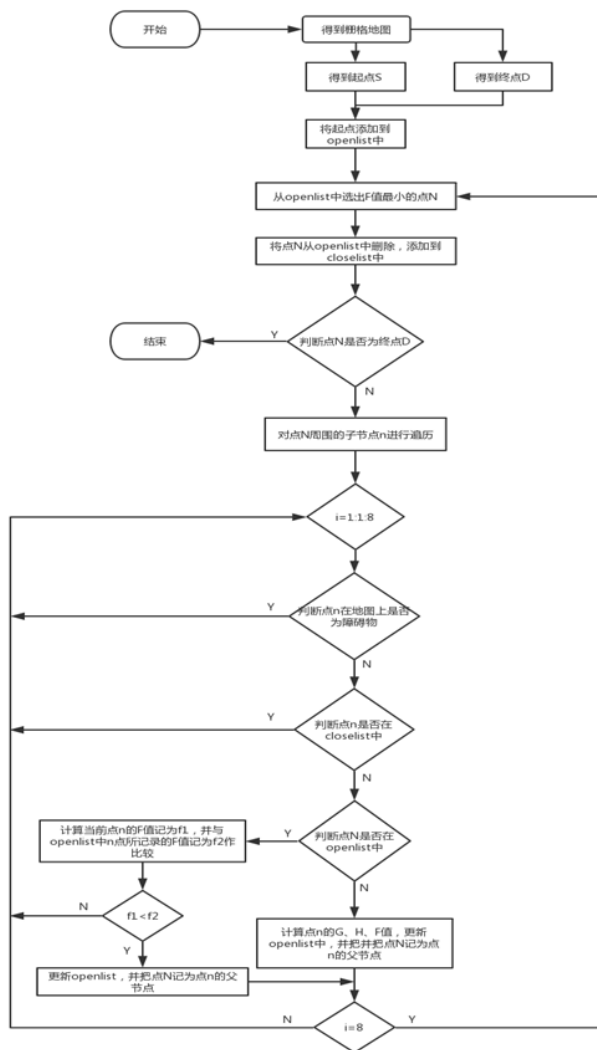
本实验我选择了路径规划进行启发函数的比较，以罗马尼亚问题和迷宫探索问题为例。



△上图为路径规划代码的可视化图。



△上图为 A*算法的可视化图



https://blog.csdn.net/qq_43362405

△流程图如上

罗马尼亚问题结论分析:

BFS 和 DFS 属于盲目搜索, 适用于无权路径规划问题。对于 BFS, 如果路径代价是基于结点深度的非递减函数, 则算法是最优的, 否则不具备最优性。

在盲目搜索策略中, 往往不考虑结点离目标有多远, 也就是说, 在搜索的过程中, 并不考虑目标的状态对当前搜索行为的影响, 而仅仅依赖于当前的状态; 比如在罗马尼亚问题中, 从城市 Arad 到城市 Bucuresti 的搜索, 仅仅只考虑在当前状态下可能导致的代价, 该策略虽然能搜索到最优解, 但是在搜索过程中可能会有很多不必要的操作, 因为搜索的结果只关注那条最优的路径, 但是实际在搜索的过程中可能会搜索到与最优路径无关的结点。而启发式搜索策略则不一样, 他会考虑目标状态对当前状态的影响, 这样做使得在搜索的过程中更大的可能搜索和最优路径相关的状态, 减少不必要的搜索, 增加搜索的效率, 比如说在罗马尼亚问题中, 可以在考虑代价的同时, 还考虑当前结点离目标结点的直线距离来选取下一个要拓展的结点, 而这个直线距离就作为启发式的信息来帮助搜索的决策, 以此更快地搜索到最优路径。

在启发式搜索中, 通常用 $g(n)$ 表示结点 n 当前的代价, 用 $h(n)$ 表示启发函数, 一般来说 $h(n)$ = 结点 n 到目标结点的最小代价路径的代价估计值 (比如在罗马尼亚问题中可以用 Arad 到 Bucuresti 的直线距离来作为从 Arad 到 Bucuresti 的最小代价路径的代价估计值), 用 $f(n)$ 作为选择结点的代价评估值, $f(n)=g(n)+h(n)$, 当 $f(n)$ 越小, 则代表结点越优; 启发式搜索对扩展结点的选择同时依赖于 $g(n)$ 和 $h(n)$, 对于 $h(n)$ 较小的结点, 我们可以认为该结点会让我们离目标结点更近, 所以 $h(n)$ 作为启发式的信息能帮助我们在搜索的时候做出决策。

迷宫探索问题分析：

在 BF 算法中，仅仅采用启发式作为评估值，可能导致不能求到最优解，A* 搜索算法同时考虑到代价，以期解决 BF 算法中的问题，因此 A* 搜索采用的代价评估值 $f(n)$ 为 $h(n)$ 与 $g(n)$ 的和，针对不同的情况采用不同的估价函数，参考代码中有以下几种：

```
float MapSearchNode::GoalDistanceEstimate( MapSearchNode &nodeGoal )
{//曼哈顿距离
    float xd = fabs(float(((float)x - (float)nodeGoal.x)));
    float yd = fabs(float(((float)y - (float)nodeGoal.y)));
    return xd + yd;
}
```

```
float MapSearchNode::GoalDistanceEstimate( MapSearchNode &nodeGoal )
{//直线距离
    float xd = fabs(float(((float)x - (float)nodeGoal.x)));
    float yd = fabs(float(((float)y - (float)nodeGoal.y)));
    return sqrt(xd*xd+yd*yd);
}
```

得到结果如下：

```
Search found goal state
Node position : (1, 7)
Node position : (0, 7)
Node position : (0, 6)
Node position : (0, 5)
Node position : (0, 4)
Node position : (0, 3)
Node position : (0, 2)
Node position : (0, 1)
Node position : (0, 0)
Node position : (1, 0)
Node position : (2, 0)
Node position : (3, 0)
Node position : (4, 0)
Node position : (5, 0)
Node position : (6, 0)
Node position : (7, 0)
Node position : (8, 0)
Node position : (9, 0)
Node position : (10, 0)
Node position : (11, 0)
Node position : (12, 0)
Node position : (13, 0)
Node position : (14, 0)
Solution steps 22
SearchSteps : 23
```

```
Search found goal state
Node position : (1, 7)
Node position : (0, 7)
Node position : (0, 6)
Node position : (0, 5)
Node position : (0, 4)
Node position : (0, 3)
Node position : (0, 2)
Node position : (0, 1)
Node position : (0, 0)
Node position : (1, 0)
Node position : (2, 0)
Node position : (3, 0)
Node position : (4, 0)
Node position : (5, 0)
Node position : (6, 0)
Node position : (7, 0)
Node position : (8, 0)
Node position : (9, 0)
Node position : (10, 0)
Node position : (11, 0)
Node position : (12, 0)
Node position : (13, 0)
Node position : (14, 0)
Solution steps 22
SearchSteps : 26
```


分析结果，知若采用直线距离，会存在探索迷宫过程中走斜线的过程，但这与问题实际情况不同，即使用直线距离的启发式函数是不合理的。

估价函数的值对 A*算法的影响

1. 搜索空间大小：估价函数的值直接影响搜索空间的大小。一个好的估价函数能够准确地评估节点的优劣程度，使得搜索算法可以更快地找到最优解或接近最优解的路径。如果估价函数能够提供精确的节点排序，搜索算法可以首先探索最有希望的路径，从而减少搜索空间的大小，提高搜索速度
2. 启发式指导：估价函数在搜索算法中充当启发式指导的角色。它为搜索算法提供关于哪些节点最有可能接近解的信息，引导搜索算法朝着最有希望的方向前进。通过启发式指导，搜索算法可以更加高效地选择扩展节点，减少不必要的搜索操作，从而提高搜索速度
3. 剪枝操作：估价函数的值可以用于剪枝搜索空间中的无关或不太有希望的分支。通过评估节点的优劣程度，搜索算法可以提前终止或跳过那些被估价函数判定为低优先级的节点，从而减少搜索的时间和计算成本。因此，一个准确的估价函数能够更好地进行剪枝操作，加快搜索速度
4. 误导性风险：估价函数如果存在误导性，可能会导致搜索算法朝着错误的方向前进，延缓搜索速度。如果估价函数的评估不当，可能会导致搜索算法偏离最优路径，进行不必要的探索。
5. 启发式函数的计算成本：如果估价函数的计算成本较高，搜索算法

在每次节点扩展时都需要耗费大量的计算资源。

启发式算法特点：

1. 剪枝：启发式搜索通过启发式函数对节点进行评估，可以剪枝搜索空间中的无关或不太有希望的分支。通过提前排除或跳过估价函数判定为低优先级的节点，搜索算法可以减少不必要的扩展操作，从而降低计算成本。

2. 贪心选择或最佳优先搜索：启发式搜索通常采用贪心选择或最佳优先搜索策略。贪心选择在每一步选择中都优先选择具有最好启发式函数值的节点进行扩展，即选择当前看起来最有希望的节点。最佳优先搜索则在扩展节点时优先选择具有最低估价函数值的节点，以期望找到最优解。

3. 完备性与最优性：如果搜索算法能够覆盖整个搜索空间，并且问题本身是有解的，那么启发式搜索可以找到解。然而，启发式搜索并不能保证找到最优解，特别是当启发式函数存在误导性或不准确时。

4. 存储空间：启发式搜索需要维护一个数据结构来存储待扩展的节点，这需要一定的存储空间。搜索算法的存储空间要求通常随着搜索空间的增大而增加，因此在实际应用中需要考虑存储资源的限制。

A*算法步骤

1. 初始化：将起始节点加入一个待处理节点列表 (open list) 中，并将其估计的成本设为 0。
2. 重复以下步骤直到满足终止条件：

- a. 从待处理节点列表(open list)中选择具有最小估计成本的节点，将其称为当前节点，并将其移出待处理节点列表。
 - b. 如果当前节点是目标节点，则算法终止，路径已找到。
 - c. 否则，将当前节点标记为已访问，并将其加入已处理节点列表(closed list)。
 - d. 对于当前节点的每个邻居节点：
 - 如果邻居节点已在已处理节点列表中，跳过该节点。
 - 如果邻居节点不在待处理节点列表中，将其加入待处理节点列表，并计算其估计成本。
 - 如果邻居节点已在待处理节点列表中，检查通过当前节点到达该邻居节点的路径是否更短。如果更短，更新邻居节点的父节点为当前节点，并更新估计成本。
3. 如果待处理节点列表为空而未找到目标节点，则不存在可行路径。
 4. 如果找到目标节点，则从目标节点回溯到起始节点，通过每个节点的父节点指针，形成最短路径

实验二 推理技术

一、 实验目的

熟悉和掌握产生式系统的运行机制，掌握基于规则推理的基本方法，利用规则演绎解决规划问题。

二、实验内容

1. 对已有的产生式系统(默认的例子)进行演示，同时可以更改其规则库或（和）事实库，进行正反向推理，了解其推理过程和机制。自己建造产生式系统（包括规则库和事实库），然后进行推理，即可以自己输入任何的规则和事实，并基于这种规则和事实进行推理。

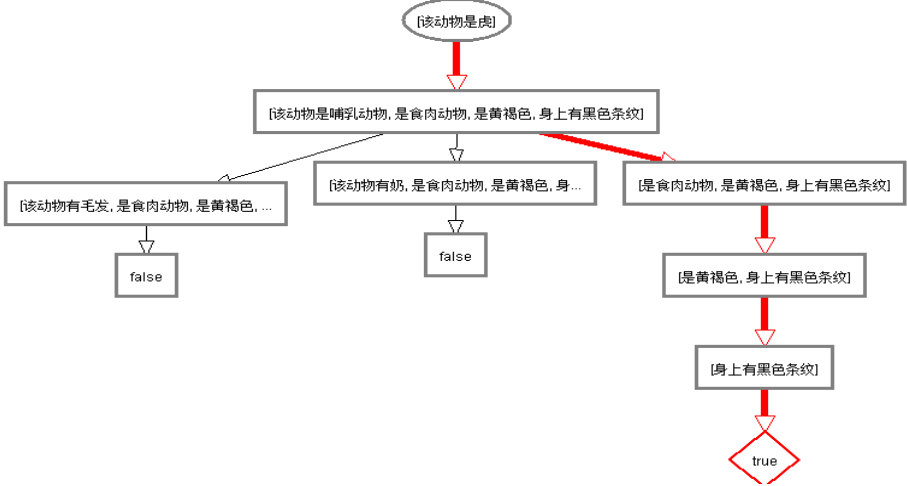
2. 观察并验证简单的推理过程。对照实验过程，自己写一个简单的规则推理，再用实验环境创建一个相应的模型,并在实验中验证或修改它。

三、 实验记录

实验报告如下所示：

产生式系统实验报告表二

姓名	张文睿	年级	计科 2101	指导老师	黄芳	日期	5.23
实验目的	熟悉和掌握产生式系统的运行机制，掌握基于规则推理的基本方法，利用规则演绎解决规划问题。					推理方法	<input type="checkbox"/> 正向推理 <input type="checkbox"/> 反向推理
建立规则库				建立事实库			
该动物是哺乳动物 <- 该动物有毛发. 该动物是哺乳动物 <- 该动物有奶. 该动物是鸟 <- 该动物有羽毛. 该动物是鸟 <- 该动物会飞&会下蛋. 该动物是食肉动物 <- 该动物吃肉. 该动物是食肉动物 <- 该动物有犬齿&有爪&眼盯前方. 该动物是有蹄类动物 <- 该动物是哺乳动物&有蹄. 该动物是有蹄类动物 <- 该动物是哺乳动物&是嚼反刍动物. 该动物是金钱豹 <- 该动物是哺乳动物&是食肉动物&是黄褐色&身上有暗斑点. 该动物是虎 <- 该动物是哺乳动物&是食肉动物&是黄褐色&身上有黑色条纹. 该动物是长颈鹿 <- 该动物是有蹄类动物&有长脖子&有长腿&身上有暗斑点. 该动物是斑马 <- 该动物是有蹄类动物&身上有黑色条纹. 该动物是鸵鸟 <- 该动物是鸟&有长脖子&有长腿&不会飞&有黑白二色. 该动物是企鹅 <- 该动物是鸟&会游泳&不会飞&有黑白二色.				%-----动物识别系统事实集: %--该动物是企鹅 %会游泳. %不会飞. %有黑白二色. %该动物是鸟. %----- %--该动物是鸟 %该动物会飞. %会下蛋. %----该动物是金钱豹 <- 该动物是哺乳动物&是食肉动物&是黄褐色&身上有暗斑点. %该动物有毛发. %是食肉动物. %是黄褐色. %身上有暗斑点. %----该动物是虎 <- 该动物是哺乳动物&是食肉			

	<p>该动物是信天翁 <- 该动物是鸟&善飞.</p>	<p>动物&是黄褐色&身上有黑色条纹.</p> <p>该动物是哺乳动物.</p> <p>是食肉动物.</p> <p>是黄褐色.</p> <p>身上有黑色条纹.</p> <p>%----该动物是长颈鹿 <- 该动物是有蹄类动物&有长脖子&有长腿&身上有暗斑点.</p> <p>%该动物是有蹄类动物.</p> <p>%有长脖子.</p> <p>%有长腿.</p> <p>%身上有暗斑点.</p>
预测结果	该动物是虎	
实验过程及结果(注意观测规则的匹配过程和方法)	<p>创建一个该动物是虎的 query，然后系统根据事实（该动物是哺乳动物，是食肉动物，是黄褐色，身上有黑色条纹）匹配到规则进行推理。</p> 	备注(原因等)
学生结论	通过设定的事实和规则，可以判断一个命题是否为真。推理系统形成一棵以询问为根的有向树，每个节点的分支是或的关系，节点内部的所有条件是与的关系，因此搜索树是一个与或图。如果有一个叶子节点的条件可以满足，那么就说明询问为真，否则询问为假。	
指导老师意见		

为了正向推理验证反向推理结果便利，设计了一个简单的产生式系统：

规则库：

是面向对象 <- 是有接口.

是面向对象 <- 是有类.

是编译语言 <- 是需要编译运行.

是脚本语言 <- 是不舒服

是脚本语言 <- 是效率较低.

是脚本语言 <- 是不需要编译运行&是逐语句运行.

是高级语言 <- 是面向对象.

是高级语言 <- 是简单&是效率较低.

是效率较高 <- 是编译语言.

是底层语言 <- 是繁琐&编译语言&直接操作内存.

是不舒服 <- 是代码很难看.

是 go lang <- 是高级语言&是不舒服.

是 python <- 是高级语言&是脚本语言.

是 java <- 是高级语言&是编译语言.

是 asm <- 是底层语言.

事实库：

是代码很难看.

是有类.

预测结果：是 go lang

实验验证过程如下：

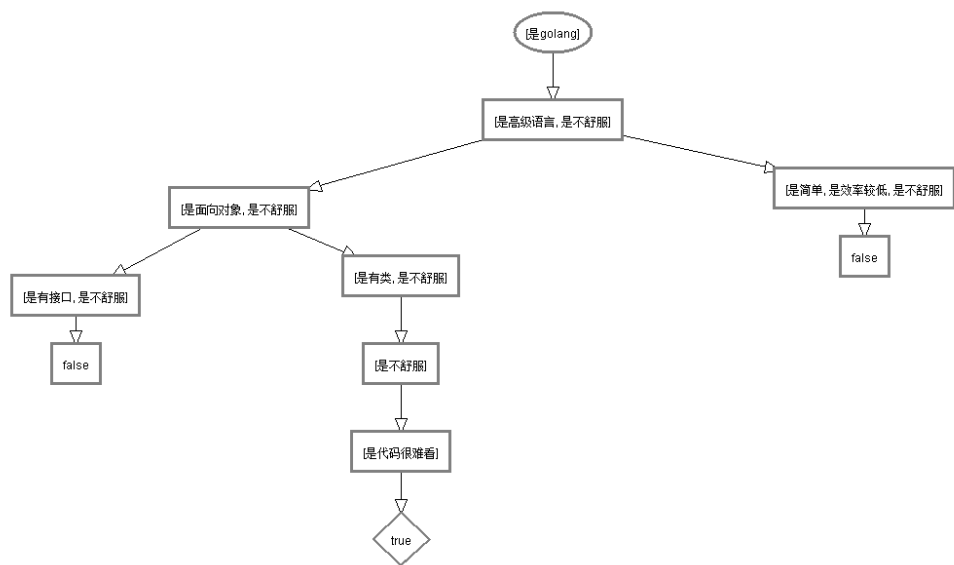


图 1: Query——是 golang

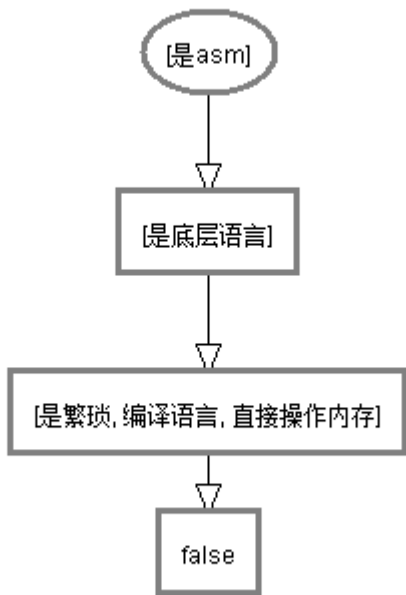


图 2: Query——是 asm

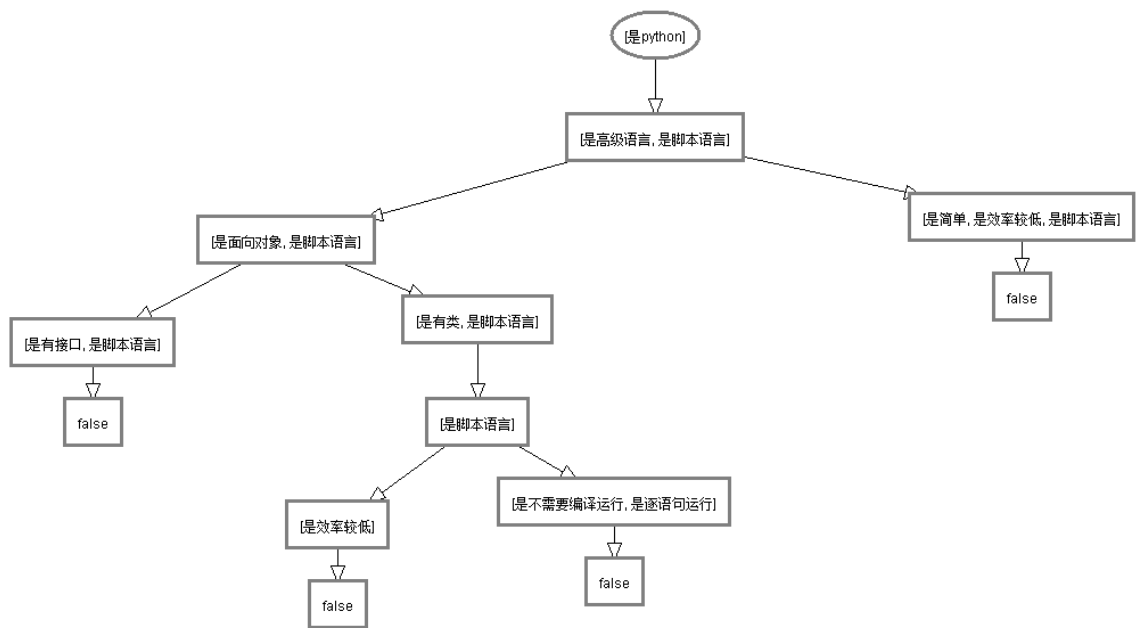


图 3: Query——是 python

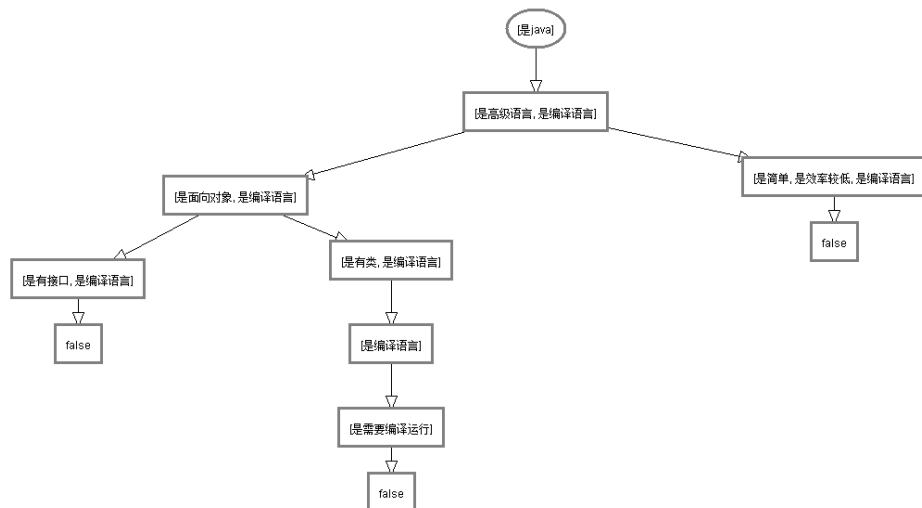


图 4: Query——是 java

反向推理过程中，假设一个询问，根据规则向上推（可能可以推出多个分支），遇到事实即“消去”，最终只有“是 go lang”的询问能够找到一条通路完全消除，即证明该询问为真，其他的则为假。

《人工智能》实验指导书

本实验课程是计算机、智能、物联网等专业学生的一门专业课程，通过实验，帮助学生更好地掌握人工智能中涉及的相关概念、算法，具体包括掌握不同搜索策略的设计思想、步骤、性能；掌握推理的基本方法；掌握神经网络的基本原理和学习机制。实验要求实验前复习课程中的有关内容，准备好实验数据，设计或编程要独立完成，完成实验报告。

实验三 神经网络

一、实验目的

掌握 BP 神经网络的算法原理，用 BP 神经网络算法对测试集进行预测。

二、实验内容

1. BP 神经网络(必做)

通过实验平台 `pd.read_csv` 读取右侧数据集 `datatest.csv`，根据每一步的提示，完成相应代码，依次完成读取数据，提取特征值和标签值，对标签值进行独热编码，划分训练集和测试集，建立模型，并训练模型，对数据进行预测，将概率值转换为标签值，评估预测的准确率。

说明：数据集 `datatest.csv` 有 16 列，前 15 列为特征值，最后 1 列为标签值，并且标签值的类型只有两种。

2. 选做

- (1) 认识 M-P 神经元
- (2) 搭建神经网络前向传输
- (3) 训练神经网络模型
- (4) 使用 `sigmoid` 激活函数
- (5) 使用 `tanh` 激活函数
- (6) 计算交叉熵
- (7) 计算均方误差
- (8) 随机梯度下降计算最小值
- (9) 构建房间预测模型

三、实验环境

中南大学 AI 能力支撑平台。



四、 实验步骤

- 1.进入神经网络可视化实验环境，基本实验步骤如下：
- (1)进入实验环境；
 - (2)选择相关的实验模块；
 - (3)根据实验指导进行相应模块代码学习和编写；
 - (4)运行代码；
 - (5)观测运行结果；
 - (6)修改相应地参数观察结果变化。
- 2.编写程序实现所选模块功能。

五、实验结论

- 包括做实验的目的、方法、过程等，具体要写成实验报告，见后附表三。
- 1、BP 网络的基本结构及 BP 算法的代码编写。
 - 2、试述参数变化对 BP 网络推理结果的影响。

附：神经网络实验报告表三

姓名	张文睿	指导老师:黄芳 汪洁	日期:23.6 .5
实验目的	掌握 BP 神经网络的算法原理，用 BP 神经网络算法对测试集进行预测		

读取数据，查看前5行	<pre>import pandas as pd data=pd.read_csv('/data/shixunfiles/185be7a01dd636dd3fc5a3b6a4dd84cc_1577440382803.csv') data.head()</pre>	<p><code>data.head()</code>用以查看前五行数据，若需打印前五行，则可改为 <code>print(data.head())</code></p>
提取特征值和标签值	<pre>X = data.iloc[:, :-1].values y = data.iloc[:, -1].values</pre>	<p><code>data.iloc[:, :-1].values</code> 用以提取 <code>data</code> 的所有行，和除了最后一列以外的所有列，将提取出特征变量的数据。</p> <p><code>data.iloc[:, -1].values</code> 用以提取 <code>data</code> 的所有行，和最后一列，将提取出目标变量的数据。</p> <p>若修改为 <pre>X= data.iloc[:, :-2].values y= data.iloc[:, -1].values</pre> 则 <code>X</code> 将变成 <code>data</code> 中所有行，除最后两列外的所有数据 <code>y</code> 将变成 <code>data</code> 倒数第二列的值</p>
对标签值进行独热编码	<pre>y = y.reshape(len(y),1) from sklearn.preprocessing import OneHotEncoder ohe = OneHotEncoder() y = ohe.fit_transform(y).toarray()</pre>	<p>首先将 <code>y</code> 的 <code>shape</code> 调整为(样本数, 1)的二维数组形式。</p> <p>然后调用 <code>fit_transform</code> 方法，将 <code>y</code> 进行独热编码转换，且该方法会同时拟合编码器的参数并对 <code>y</code> 进行编码转换。</p> <p>最后将独热编码后的结果，通过 <code>toarray</code> 方法，将结果转换为密集矩阵。</p>
划分训练集	<pre>from sklearn.model_selection import train_test_split x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=666)</pre>	<p><code>train_test_split</code> 函数用于将数据集分为数据集和测试集。本段代码将数据集划分为 80% 的训练集和 20% 的测试集。经过该部分代码后，<code>x_train</code> 为训练集的特征变量，<code>x_test</code> 为测试集的特征变量；</p>

和测试集		<p><code>y_train</code> 为训练集的目标变量，<code>y_test</code> 为测试集的目标变量；</p> <p>若 <code>test_size</code> 修改为 0.25，则将数据集划分为 75% 的训练集和 25% 的测试集。</p>
建立模型，并进行训练	<pre>bp=bpnetwork() bp.fit(x_train,y_train,100,200,0.01,2)</pre>	<p>第一行代码创建了一个名为 <code>bp</code> 的神经网络</p> <p>第二行调用了 <code>fit</code> 方法来训练神经网络模型，<code>x_train</code> 是训练集的特征变量，<code>y_train</code> 是训练集的目标变量，迭代次数设置为 100，每次选择训练集中的 200 个样本进行训练，0.01 是学习率，即更新权重时的步长大小，2 是隐藏层的数量。</p> <p>若将第二行代码修改为 <code>bp.fit(x_train,y_train,200,100,0.005,2)</code>，则迭代次数变为 200，单次选择样本量为 100，学习率为 0.005，隐藏层数目为 2</p>
用训练的模型进行预测	<pre>ypredict= bp.predict(x_test)</pre>	<p>调用 <code>predict</code> 方法，对输入数据进行预测</p>
将概率值转换为标签值	<pre>p = np.argmax(ypredict,axis =1)#找到概率值最大的那个位置 y_test = np.argmax(y_test,axis =1)</pre>	<p>第一行代码，会在预测结果的第一行上找到最大值的索引位置。因 <code>ypredict</code> 的每一行代表一个样本的预测结果，故此处目的是找到每个样本预测结果中概率值最大的类别所在的位置，并将结果存储在变量 <code>p</code> 中。</p> <p>第二行代码，第一行代码，会在预测结果的第一行上找到最大值的索引位置。因 <code>y_test</code> 的每一行代表一个样本的预测结果，故此处目的是找到每个</p>

		<p>样本预测结果中概率值最大的类别所在的位置，并将结果存储在变量 <code>y_test</code> 中。</p>
评估预测准确率	<pre>acc = np.mean(p==y_test) print('准确率为%.4f'%acc)</pre>	<p>第一行代码，通过比较 <code>p</code> 和 <code>y_test</code> 中的每个元素，计算其相等的比例，然后 <code>mean</code> 方法将 <code>True</code> 的比例计算为准确率。</p> <p>第二行代码，将准确率以小数点后四位的格式打印输出。</p>

`__init__(self)`: 初始化函数, 用于初始化神经网络的权重和偏置。在这个函数中, `self.w0`、`self.w1`、`self.b0`、`self.b1` 被初始化为 `None`, 表示它们的值暂时为空。

`hidden_in(self, feature, w0, b0)`: 计算隐藏层的输入函数。输入参数包括特征数据 `feature`、输入层到隐藏层之间的权重 `w0` 和偏置 `b0`。首先, 将特征数据转换为矩阵形式, 并获取特征的行数 `m`。然后, 通过矩阵乘法计算隐藏层的输入 `hidden_in = feature * w0`。接下来, 对每一行的隐藏层输入加上对应的偏置 `b0`。最后, 返回计算得到的隐藏层输入。

`sig(self, x)`: Sigmoid 函数, 将输入值 `x` 转换为概率值。这里使用了 Sigmoid 函数的常见表达式 $1.0 / (1 + \text{np.exp}(-x))$, 通过 `np.exp` 函数计算指数值, 并将其应用于逐元素计算。

`hidden_out(self, hidden_in)`: 计算隐藏层的输出函数。输入参数为隐藏层的输入 `hidden_in`。该函数通过调用 `self.sig` 方法, 将隐藏层的输入应用于 Sigmoid 函数, 从而得到隐藏层的输出 `hidden_output`。

`predict_in(self, hidden_out, w1, b1)`: 计算输出层的输入函数。输入参数包括隐藏层的输出 `hidden_out`、隐藏层到输出层之间的权重 `w1` 和偏置 `b1`。通过矩阵乘法计算输出层的输入 `predict_in = hidden_out * w1`, 然后对每一行的输出层输入加上对应的偏置 `b1`。最后, 返回计算得到的输出层输入。

`predict_out(self, predict_in)`: 计算输出层的输出函数。输入参数为输出层的输入 `predict_in`。该函数通过调用 `self.sig` 方法, 将输出层的输入应用于 Sigmoid 函数, 从而得到输出层的输出 `result`。

`partial_sig(self, x)`: 计算 Sigmoid 函数的偏导数。输入参数为矩阵 `x`, 函数根据偏导数的计算公式 $\text{sig}(x) * (1 - \text{sig}(x))$, 对输入矩阵中的每个元素逐个计算并返回。

`fit(self, feature, label, n_hidden, maxcycle, alpha, n_output)`: BP 神经网络的训练方法。输入参数包括特征数据 `feature`、标签数据 `label`、隐藏层的节点个数 `n_hidden`、最大迭代次数 `maxcycle`、学习率 `alpha` 和输出层的节点个数 `n_output`。在该方法

中，首先初始化网络的权重和偏置，使用随机数生成并经过一定的缩放。然后进行训练过程，根据最大迭代次数循环执行以下步骤：

fit 函数中，分为初始化和训练两部分。

初始化部分：

```
self.w0 = np.mat(np.random.rand(n,n_hidden))
```

首先，使用 `np.random.rand(n,n_hidden)` 生成一个大小为 `(n, n_hidden)` 的随机矩阵 `w0`。`np.random.rand()` 函数会生成一个由 0 到 1 之间的随机数填充的数组。`n` 表示输入层的节点数，`n_hidden` 表示隐藏层的节点数。

```
self.w0 = self.w0*(8.0*sqrt(6)/sqrt(n+n_hidden)) - np.mat(np.ones((n,n_hidden))) * (4.0*sqrt(6)/sqrt(n+n_hidden))
```

接下来，对刚才生成的随机矩阵 `w0` 进行缩放和平移操作。

`(8.0*sqrt(6)/sqrt(n+n_hidden))` 表示缩放因子，可以确保权重值适合神经网络的训练。

`np.mat(np.ones((n,n_hidden)))` 创建一个大小为 `(n, n_hidden)` 的全 1 矩阵，表示平移量。

最后，通过将缩放和平移应用于随机矩阵 `w0`，更新 `self.w0` 的值。

```
self.b0 = np.mat(np.random.rand(1,n_hidden))
```

类似地，使用 `np.random.rand(1,n_hidden)` 生成一个大小为 `(1, n_hidden)` 的随机矩阵 `b0`，表示隐藏层的偏置。

```
self.b0 = self.b0*(8.0*sqrt(6)/sqrt(n+n_hidden)) - np.mat(np.ones((1,n_hidden))) * (4.0*sqrt(6)/sqrt(n+n_hidden))
```

对刚才生成的随机矩阵 `b0` 进行缩放和平移操作，方式与权重 `w0` 相似。

`(8.0*sqrt(6)/sqrt(n+n_hidden))` 表示缩放因子，`(4.0*sqrt(6)/sqrt(n+n_hidden))` 表示平移量。

最后，通过将缩放和平移应用于随机矩阵 `b0`，更新 `self.b0` 的值。

类似地，通过上述步骤，初始化输出层的权重 `self.w1` 和偏置 `self.b1`，使用与输入层到隐藏层相似的方法进行初始化。

前向传播：

计算隐藏层的输入 `hidden_input`，调用 `self.hidden_in` 方法传入特征数据、输入层到隐藏层之间的权重和偏置。

计算隐藏层的输出 `hidden_output`，调用 `self.hidden_out` 方法传入隐藏层的输入。

计算输出层的输入 `output_in`，调用 `self.predict_in` 方法传入隐藏层的输出、隐藏层到输出层之间的权重和偏置。

计算输出层的输出 `output_out`，调用 `self.predict_out` 方法传入输出层的输入。

反向传播：

计算输出层到隐藏层之间的残差 `delta_output`，根据误差和偏导数计算得到。

计算输入层到隐藏层之间的残差 `delta_hidden`，根据输出层残差和偏导数计算得到。

更新权重和偏置，即进行梯度下降：

更新隐藏层到输出层之间的权重 `self.w1`，根据隐藏层的输出和输出层残差计算得到。

更新隐藏层到输出层之间的偏置 `self.b1`，根据输出层残差的列求和并乘以学习率。

更新输入层到隐藏层之间的权重 `self.w0`，根据特征数据和隐藏层残差计算得到。

更新输入层到隐藏层之间的偏置 `self.b0`，根据隐藏层残差的列求和并乘以学习率。

`predict(self, x_test)`: 对给定的测试样本进行预测。输入参数为测试样本 `x_test`，这里假设为一个 NumPy 数组。在该方法中，将测试样本转换为矩阵形式，并依次调用 `self.hidden_in`、`self.hidden_out`、`self.predict_in` 和 `self.predict_out` 方法，最终将预测结果转换为 NumPy 数组并返回。

这些函数共同组成了一个简单的 BP 神经网络类，用于训练和预测任务

运行结果展示：

按照默认参数运行程序，得到结果如图 1 所示，准确率为 0.9649

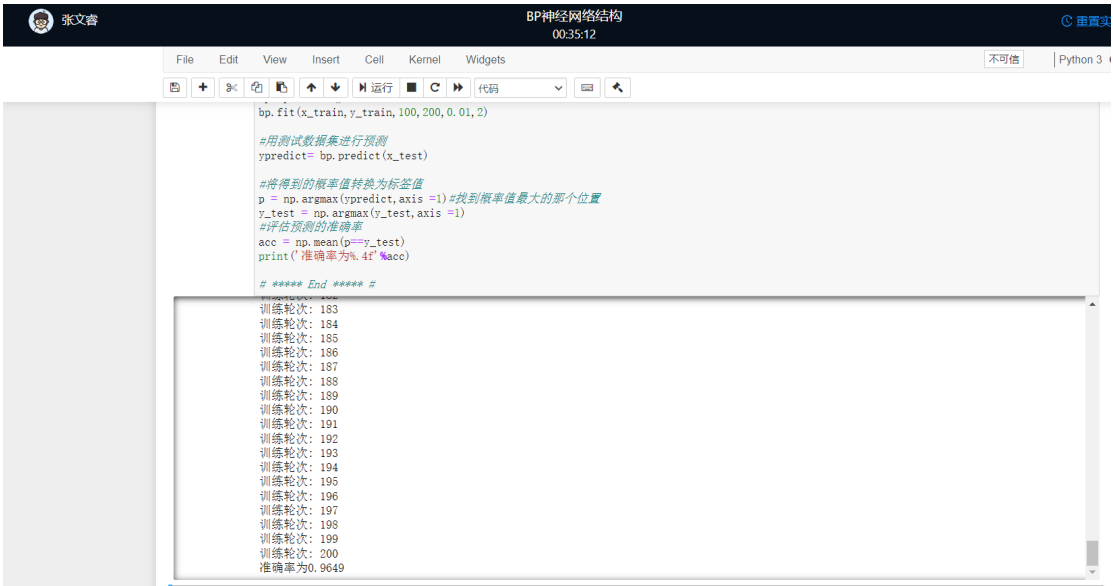


图 1：默认参数准确率

仅修改训练集和测试集的比例，得到结果如图 2 所示，准确率为 0.9860

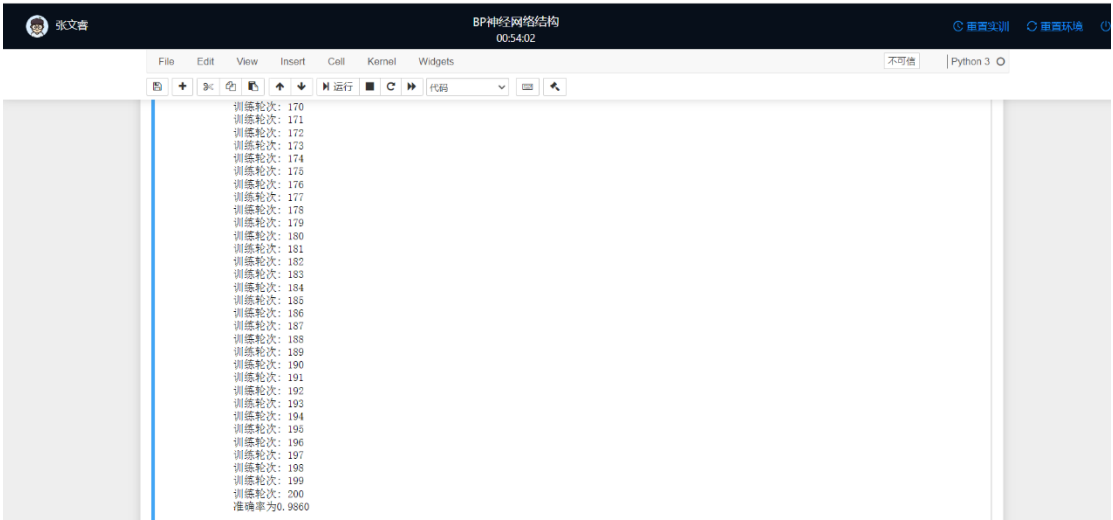
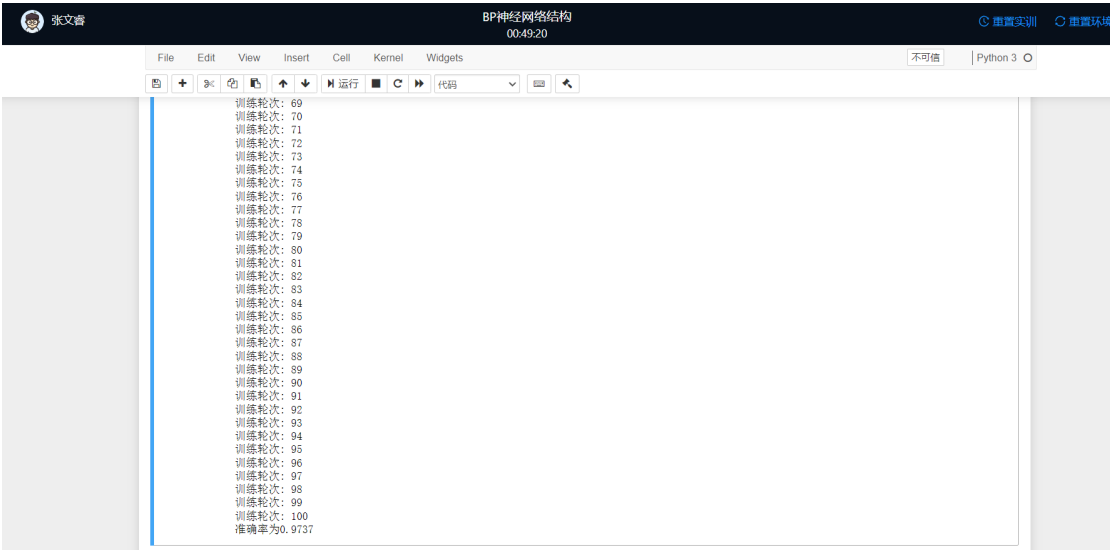


图 2：测试集比例 25%对应准确率

仅修改训练轮次，得到结果如图 3 所示，准确率为 0.9737

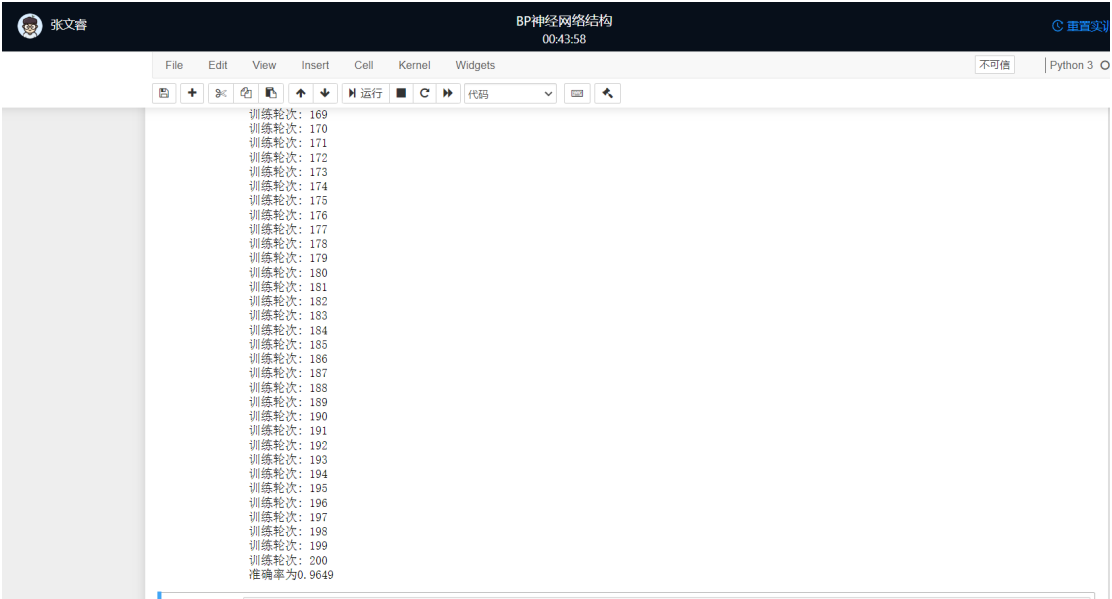


The screenshot shows a Jupyter Notebook titled "BP神经网络结构" (BP Neural Network Structure) with a timestamp of 00:49:20. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets) and a toolbar with icons for running, saving, and other actions. The code cell contains a list of training epochs from 69 to 100, followed by the accuracy result: 准确率为0.9737.

```
训练轮次: 69  
训练轮次: 70  
训练轮次: 71  
训练轮次: 72  
训练轮次: 73  
训练轮次: 74  
训练轮次: 75  
训练轮次: 76  
训练轮次: 77  
训练轮次: 78  
训练轮次: 79  
训练轮次: 80  
训练轮次: 81  
训练轮次: 82  
训练轮次: 83  
训练轮次: 84  
训练轮次: 85  
训练轮次: 86  
训练轮次: 87  
训练轮次: 88  
训练轮次: 89  
训练轮次: 90  
训练轮次: 91  
训练轮次: 92  
训练轮次: 93  
训练轮次: 94  
训练轮次: 95  
训练轮次: 96  
训练轮次: 97  
训练轮次: 98  
训练轮次: 99  
训练轮次: 100  
准确率为0.9737
```

图 3：训练轮次 100 次对应准确率

仅修改学习率，得到结果如图 4 所示，准确率为 0.9649



The screenshot shows a Jupyter Notebook titled "BP神经网络结构" (BP Neural Network Structure) with a timestamp of 00:43:58. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets) and a toolbar with icons for running, saving, and other actions. The code cell contains a list of training epochs from 169 to 200, followed by the accuracy result: 准确率为0.9649.

```
训练轮次: 169  
训练轮次: 170  
训练轮次: 171  
训练轮次: 172  
训练轮次: 173  
训练轮次: 174  
训练轮次: 175  
训练轮次: 176  
训练轮次: 177  
训练轮次: 178  
训练轮次: 179  
训练轮次: 180  
训练轮次: 181  
训练轮次: 182  
训练轮次: 183  
训练轮次: 184  
训练轮次: 185  
训练轮次: 186  
训练轮次: 187  
训练轮次: 188  
训练轮次: 189  
训练轮次: 190  
训练轮次: 191  
训练轮次: 192  
训练轮次: 193  
训练轮次: 194  
训练轮次: 195  
训练轮次: 196  
训练轮次: 197  
训练轮次: 198  
训练轮次: 199  
训练轮次: 200  
准确率为0.9649
```

图 4：学习率 0.02 对应准确率

根据实验结果可知，学习率、测试集训练集的比例、训练轮次和隐层数目是影响 BP 神经网络的预测准确率的关键参数。它们的设置会对神经网络的训练过程和最终的预测性能产生影响。

学习率 (Learning Rate)：学习率决定了在每次权重更新时调整权重的步长大小。过大的学习率可能导致权重更新过大，训练不稳定或发散；而过小的学习率可能导致收敛缓慢，需要更多的训练轮次才能达到较好的性能。因此，适当选择学习率可以帮助网络更快地收敛并取得更好的预测准确率。

测试集训练集的比例 (Train-Test Split Ratio)：将数据集划分为训练集和测试集是为了评估模型在未见过的数据上的泛化能力。测试集的比例选择过小可能导致测试集样本不足以准确评估模型的性能，而选择过大可能会导致训练集样本不足以训练出具有良好泛化能力的模型。因此，选择适当的训练集和测试集比例可以帮助避免过拟合或欠拟合，从而提高预测准确率。

训练轮次 (Number of Training Epochs)：训练轮次表示模型在整个训练集上迭代的次数。过少的训练轮次可能导致模型无法充分学习训练集的特征，而过多的训练轮次可能导致模型过拟合训练集的特征，泛化能力下降。因此，选择适当的训练轮次可以帮助在训练集上充分学习特征并在测试集上获得良好的预测准确率。

隐层数目 (Number of Hidden Layers)：神经网络的隐层数目决定了网络的复杂性和表示能力。较少的隐层可能导致网络无法学习复杂的非线性关系，而过多的隐层可能导致网络过于复杂，难以训练和泛化。因此，选择适当的隐层数目可以平衡网络的复杂性和训练的有效性，从而影响预测准确率。

正则化优化神经网络

一、 实验目的

本关任务：编写一个能使用 L2 正则化优化神经网络的小程序

二、 实验内容

任务需要补全文件中 **Begin-End** 中间的代码，实现 L2 正则化技术来解决过拟合问题，提高验证集上的准确率，具体分为如下几个部分：

导入训练集与测试集数据；

设置添加 L2 正则化的 adam 优化器；

搭建神经网络并进行训练。

三、 实验环境

中南大学 AI 能力支撑平台

四、 实验步骤

- ① 导入训练集与测试集数据
- ② 设置添加 L2 正则化的 adam 优化器
- ③ 搭建神经网络并进行训练
- ④ 会根据搭建的模型精度是否可以超越 90%进行判断，若超过 90%会说输出预期结果，否则会输出“模型仍需进行调整”

五、 程序介绍

```
batch_size_train = 64、batch_size_test = 1000、learning_rate = 0.01、  
log_interval = 10、random_seed = 1
```

这些变量定义了批处理大小、学习率、日志间隔和随机种子等超参数。

```
torch.manual_seed(random_seed)
```

这行代码设置随机种子，用于使实验可复现。

`train_loader` 和 `test_loader` 定义了训练数据集和测试数据集的数据加载器。

`torchvision.datasets.MNIST` 用于加载 MNIST 数据集，`train=True` 表示加载训练集，`train=False` 表示加载测试集。

`torchvision.transforms.Compose` 用于定义数据预处理的组合，例如将图像转换为张量和归一化等。

`batch_size` 表示每个批次的样本数，`shuffle=True` 表示在每个迭代中随机打乱数据。

```
class Net(nn.Module):  
    def __init__(self):
```

这部分代码定义了一个名为 `Net` 的神经网络模型类，并在 `__init__` 方法中定义了神经网络的结构。

神经网络有 7 个全连接层(`fc1` 到 `fc7`)，其中前 6 层的输出大小为 700，最后一层的输出大小为 10，对应 10 个类别。

```
    def forward(self, x):
```

这个方法定义了神经网络的前向传播过程，即将输入 `x` 传递到网络中，生成输出。

首先，将输入展平为一维向量，然后通过 `ReLU` 激活函数依次通过每个全连接层。

最后，通过 `F.log_softmax` 函数对输出进行 `log_softmax` 操作，得到概率分布。

```
network = Net() 和 optimizer = optim.Adam(network.parameters(),  
lr=learning_rate, weight_decay=0.0018)
```

这部分代码创建了一个 `Net` 类的实例作为神经网络模型。

`optim.Adam` 是一个优化器，用于更新神经网络的参数。

`network.parameters()` 获取神经网络模型的参数，用于优化器的更新操作。

`lr=learning_rate` 表示学习率，`weight_decay` 表示 L2 正则化的权重衰减系

数。

`def train(epoch):`和 `def test():`

这两个函数分别定义了训练过程和测试过程。

`train(epoch)`函数用于进行一轮训练，它遍历训练数据集的每个批次，计算损失并更新网络参数。

`test()`函数用于在测试数据集上评估模型的性能，计算损失和准确率。

`train(1)`

这行代码调用 `train` 函数开始进行一轮训练。

`epoch` 表示训练的轮数，这里设置为 1。

这段代码的作用是定义了一个具有 7 个全连接层的神经网络模型，并在 MNIST 数据集上进行训练和测试。它使用 Adam 优化器来更新模型参数，通过交叉熵损失函数计算损失，并使用 ReLU 激活函数进行非线性变换。最后，打印训练过程中的损失信息和在测试集上的准确率

六、 程序结果展示

程序运行结果如图 5 所示：



图 5：正则化优化神经网络结果

七、 代码附录

```
import warnings
warnings.filterwarnings("ignore")
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader

batch_size_train = 64

batch_size_test = 1000
learning_rate = 0.01
log_interval = 10
random_seed = 1
torch.manual_seed(random_seed)

#####begin#####
# 导入数据
# 数据路径: /data/workspace/myshixun/data/
train_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('/data/workspace/myshixun/data/',
                               train=True,
                               download=True,
                               transform=torchvision.transforms.Compose([
                                   torchvision.transforms.ToTensor(),
                                   torchvision.transforms.Normalize(
                                       (0.1307,), (0.3081,))
                               ])),
    batch_size=batch_size_train, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('/data/workspace/myshixun/data/',
                               train=False,
                               download=True,
                               transform=torchvision.transforms.Compose([
                                   torchvision.transforms.ToTensor(),
                                   torchvision.transforms.Normalize(
                                       (0.1307,), (0.3081,))
                               ])),
    batch_size=batch_size_test, shuffle=True)

# 搭建神经网络
class Net(nn.Module):
```

```

def __init__(self):
    super(Net, self).__init__()
    self.fc1 = nn.Linear(784, 700)
    self.fc2 = nn.Linear(700, 700)
    self.fc3 = nn.Linear(700, 700)
    self.fc4 = nn.Linear(700, 700)
    self.fc5 = nn.Linear(700, 700)
    self.fc6 = nn.Linear(700, 700)
    self.fc7 = nn.Linear(700, 10)

def forward(self, x):
    x = x.view(-1, 784)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = F.relu(self.fc3(x))
    x = F.relu(self.fc4(x))
    x = F.relu(self.fc5(x))
    x = F.relu(self.fc6(x))
    x = self.fc7(x)
    return F.log_softmax(x)

```

创建网络与优化器设置

#####end#####

```

network = Net()
optimizer = optim.Adam(network.parameters(), lr=learning_rate,
weight_decay=0.0018)

```

训练函数

```

def train(epoch):
    network.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data = data
        target = target
        optimizer.zero_grad()
        output = network(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{}] ({:.0f}%) \t Loss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

```

测试函数

def test():

 network.eval()

 test_loss = 0

 correct = 0

 with torch.no_grad():

 for data, target in test_loader:

 data = data

 target = target

 output = network(data)

 test_loss += F.nll_loss(output, target, size_average=False).item()

 pred = output.data.max(1, keepdim=True)[1]

 correct += pred.eq(target.data.view_as(pred)).sum()

test_loss /= len(test_loader.dataset)

print("\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%) \n".format(

 test_loss, correct, len(test_loader.dataset),

 100. * correct / len(test_loader.dataset)))

return 100 * float(correct) / float(len(test_loader.dataset))

train(1)