Artificial Intelligence

ดร.บุญเสริม กิจศิริกุล
ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย

1. Introduction

Definition

Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.

Intelligent behavior

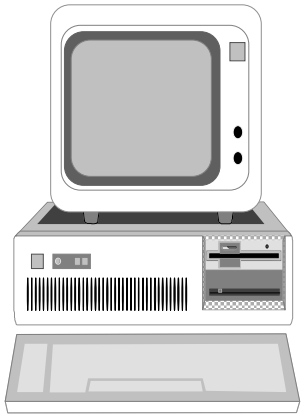
- Learn or understand from experience
- Make sense out of ambiguous or contradictory messages
- Respond quickly and successfully to a new situation
- Use reason in solving problems and directing conduct effectively
- Deal with perplexing situations
- Understand and infer in ordinary, rational ways
- Apply knowledge to manipulate the environment
- Acquire and apply knowledge
- Think and reason

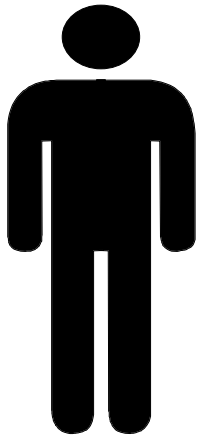
Turing Test

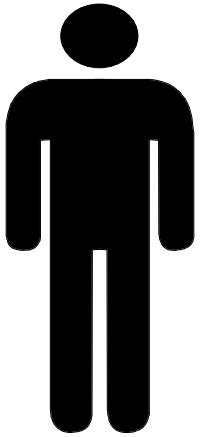
- เราจะรู้ได้อย่างไรว่าเครื่องสามารถคิดเองได้

  Alan Turing เสนอวิธีที่เรียกว่า Turing Test
- ใช้คน 2 คนและเครื่อง 1 เครื่อง
- คนแรกเป็นผู้ตั้งคำถามโดยใช้คีย์บอร์ดและจอภาพ ซึ่งผู้ถามนี้จะอยู่ห้องแยกจากเครื่องและคนที่สอง
- ผู้ถามต้องแยกให้ได้ว่าคำตอบที่มาจากอีกห้องหนึ่งนั้นเป็นของใคร

Turing Test (cont.)







Chinese Room

- นาย A เป็นคนอังกฤษ(พูดภาษาอังกฤษได้ภาษาเดียว)
- A อยู่ในห้อง ๆหนึ่งซึ่งมีกฎสำหรับแปลอักษรจีนชุดหนึ่ง ๆไป เป็นอักษรจีนชุดใหม่
- A รับอักษรจีนจากภายนอกเข้ามาแปล
- A ใช้กฎที่มีอยู่สร้างอักษรชุดใหม่แล้วส่งออกไป (input และ output อาจเป็นคำถามและคำตอบ)
- ถ้าการแปลถูกต้องจะพูดได้ไหมว่า"Aฉลาด","Aเข้าใจอักษรจีน"

Applications of AI

- Natural Language Processing
- Intelligent Retrival from Databases
- Expert Systems
- Theorem Proving
- Robotics
- Automatic Programming
- Scheduling Problems
- Perception Problems

Definitions of AI

- The goal of work in AI is to build machines that performs tasks normally requiring human intelligence
- Research scienctists in AI try to get machines to exhibit behavior that we call intelligent behavior when we observe it in human beings
- The capability of a device to perform functions that are normally associated with human intelligence, such as reasoning, learning and self-improvement.
- AI is the science of making machines do things that would required intelligence if done by men

Definitions of AI (cont.)

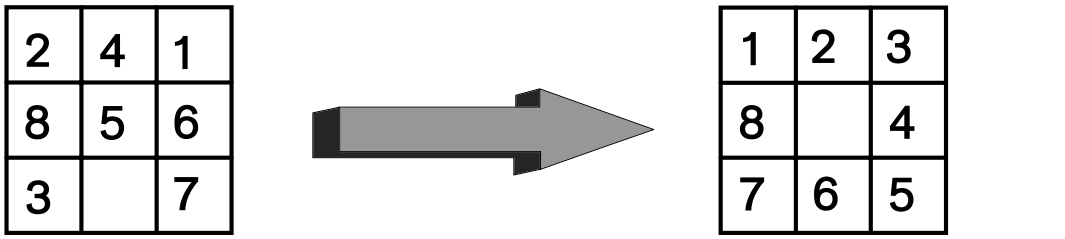
- The field commonly called AI may, perhaps, be described as the totality of attempts to make and understand machines that are able to perform tasks that, until recently, only human beings could perform and to perform them with effectiveness and speed comparable to a human
- AI is defined generally as the attempt to construct mechanisms that perform tasks requiring intelligence when performed by humans

2. Problems, Problem Spaces and Search

- สิ่งที่ต้องทำในการแก้ปัญหาหนึ่ง ๆคือ
  1. นิยามปัญหาอย่างชัดเจน แสดงสถานการณ์ปัจจุบัน (intial situation), สถานการณ์สุดท้าย (final situation)หรือคำตอบของปัญหา
  2. วิเคราะห์ปัญหา
  3. หาความรู้ที่ใช้ในการแก้ปัญหาว่ามีอะไรบ้าง
  4. เลือกเทคนิคแก้ปัญหาที่เหมาะสม

2.1 นิยามปัญหาในรูปของState Space Search

ปัญหา 8-Puzzle: 8-Puzzle ประกอบด้วยถาดขนาด 3x3 ซึ่งภายในบรรจุแผ่นป้ายขนาด 1x1 จำนวน 8 แผ่นในแต่ละแผ่นจะมีหมายเลขแสดง และมีช่องว่างอยู่1ช่องที่แผ่นป้ายสามารถเลื่อนเข้ามาแทนที่ได้ ปัญหาคือให้เลื่อนแผ่นป้ายเหล่านี้จากตำแหน่งเริ่มต้นให้เป็นตำแหน่งสุดท้ายซึ่งเป็นคำตอบ



ตำแหน่งเริ่มต้น ตำแหน่งสุดท้าย

State Space Search

- ในการนิยามปัญหาในรูปของ state space search นั้น
  - นิยาม state space แสดง objects ทั้งหมดที่เกี่ยวข้องกับปัญหา
  - เราให้ initial state แทนตำแหน่งเริ่มต้นและ final state (goal state) แทนตำแหน่งสุดท้าย
  - หากฎที่ใช้เปลี่ยน state จาก state หนึ่งไปอีก state หนึ่ง เราเรียกกฎเหล่านี้ว่า operators
  - ใช้เทคนิคของ search ในการเปลี่ยนจาก initial state ไปยัง goal state

State Space Search (cont.)

Operatorsของ8-Puzzle

- แผ่นป้ายหนึ่งๆจะเลื่อนมาที่ช่องว่างได้ถ้าอยู่ติดกับช่องว่าง และ state จะเปลี่ยนจาก state เดิมไปยัง state ใหม่ซึ่งตำแหน่งของแผ่นป้ายสลับที่กับตำแหน่งของช่องว่าง

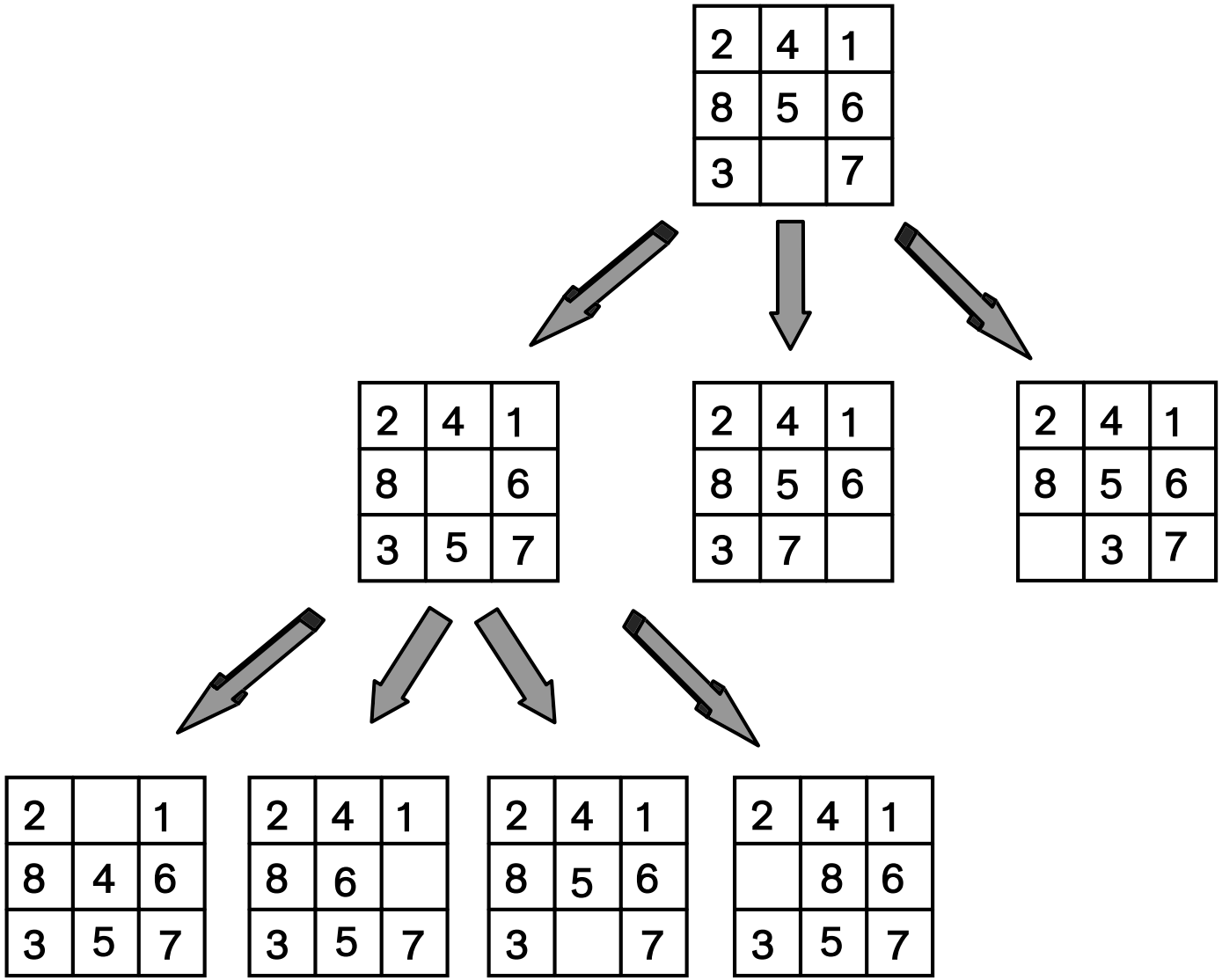
1 ด้านบนของช่องว่างมีแผ่นป้าย -> สลับตำแหน่งของช่องว่างกับแผ่นป้าย

2 ด้านขวาของช่องว่างมีแผ่นป้าย -> สลับตำแหน่งของช่องว่างกับแผ่นป้าย

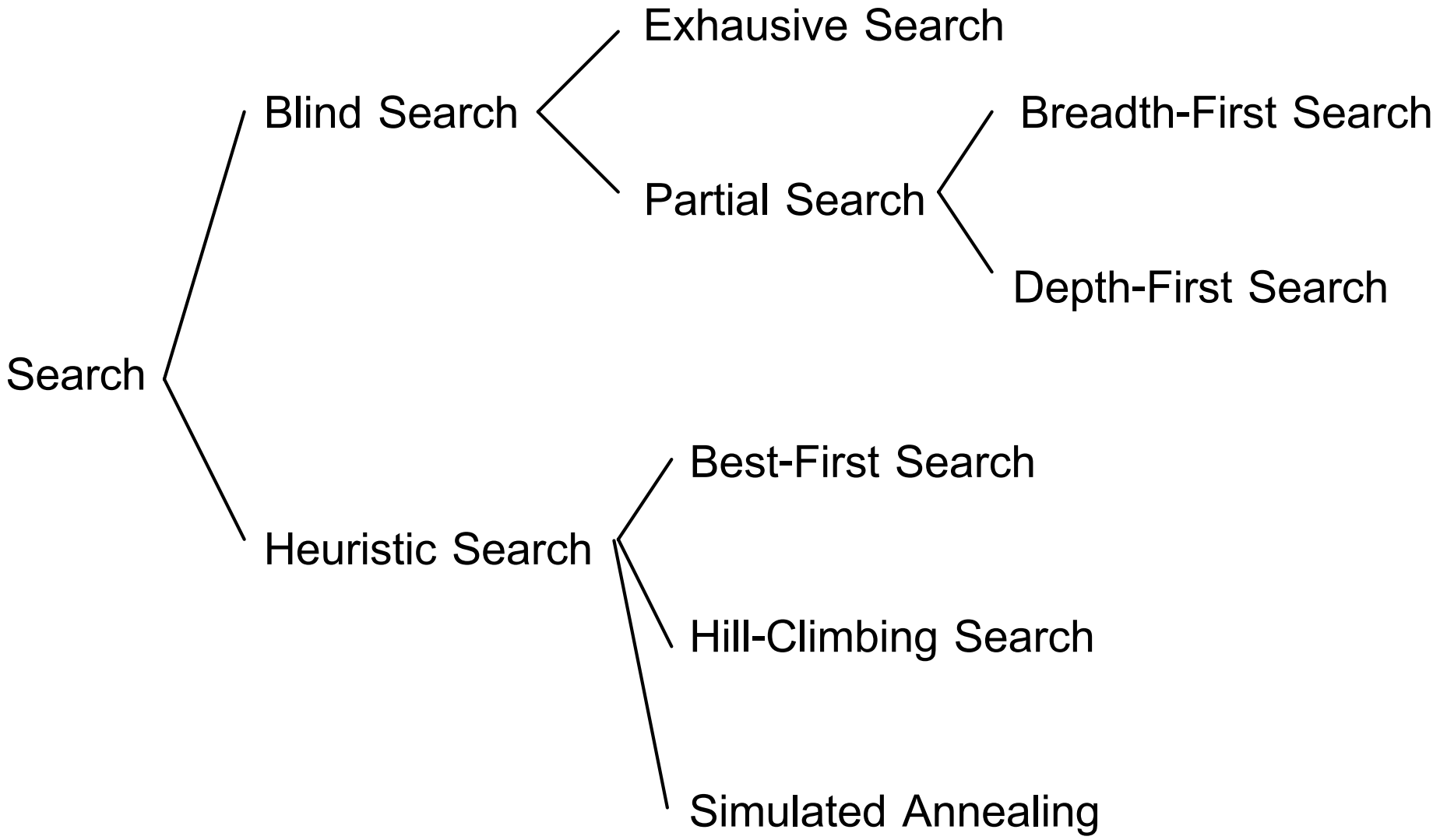
3 ด้านล่างของช่องว่างมีแผ่นป้าย -> สลับตำแหน่งของช่องว่างกับแผ่นป้าย

4 ด้านซ้ายของช่องว่างมีแผ่นป้าย -> สลับตำแหน่งของช่องว่างกับแผ่นป้าย

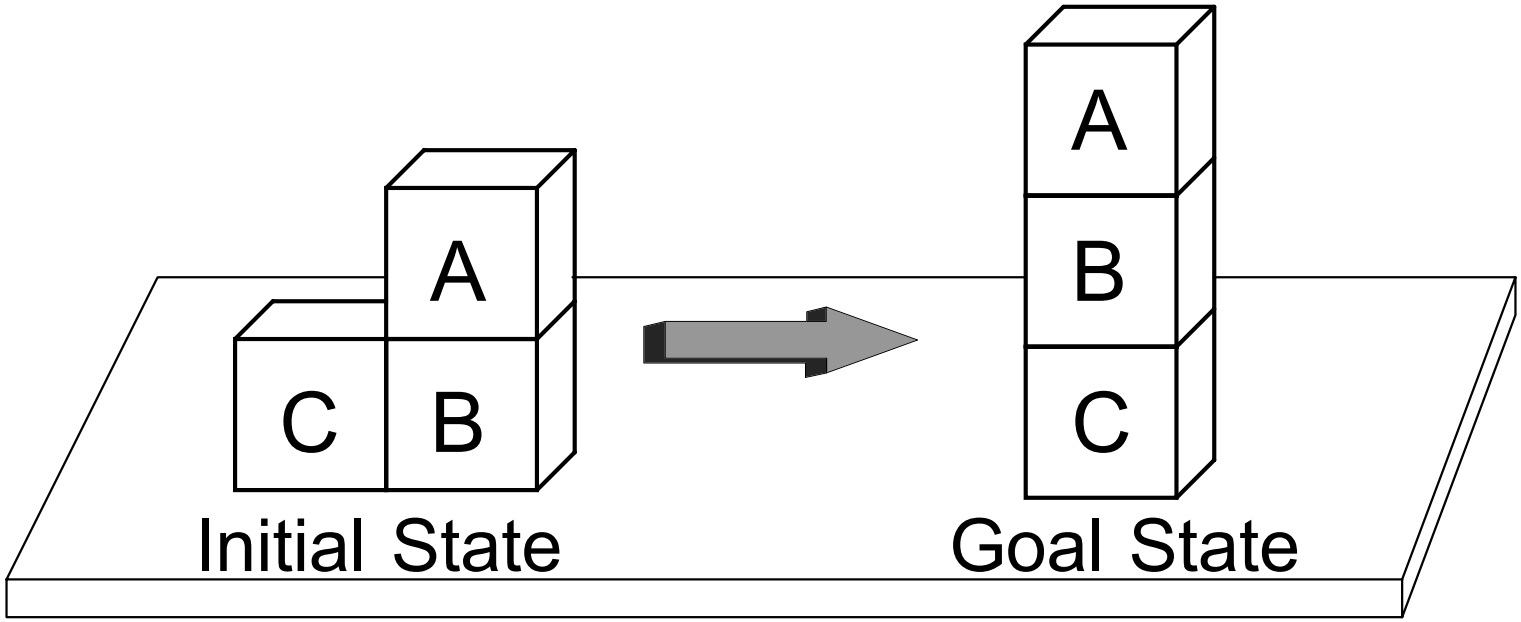
State Space Search (cont.)



2.2 Search Techniques in AI



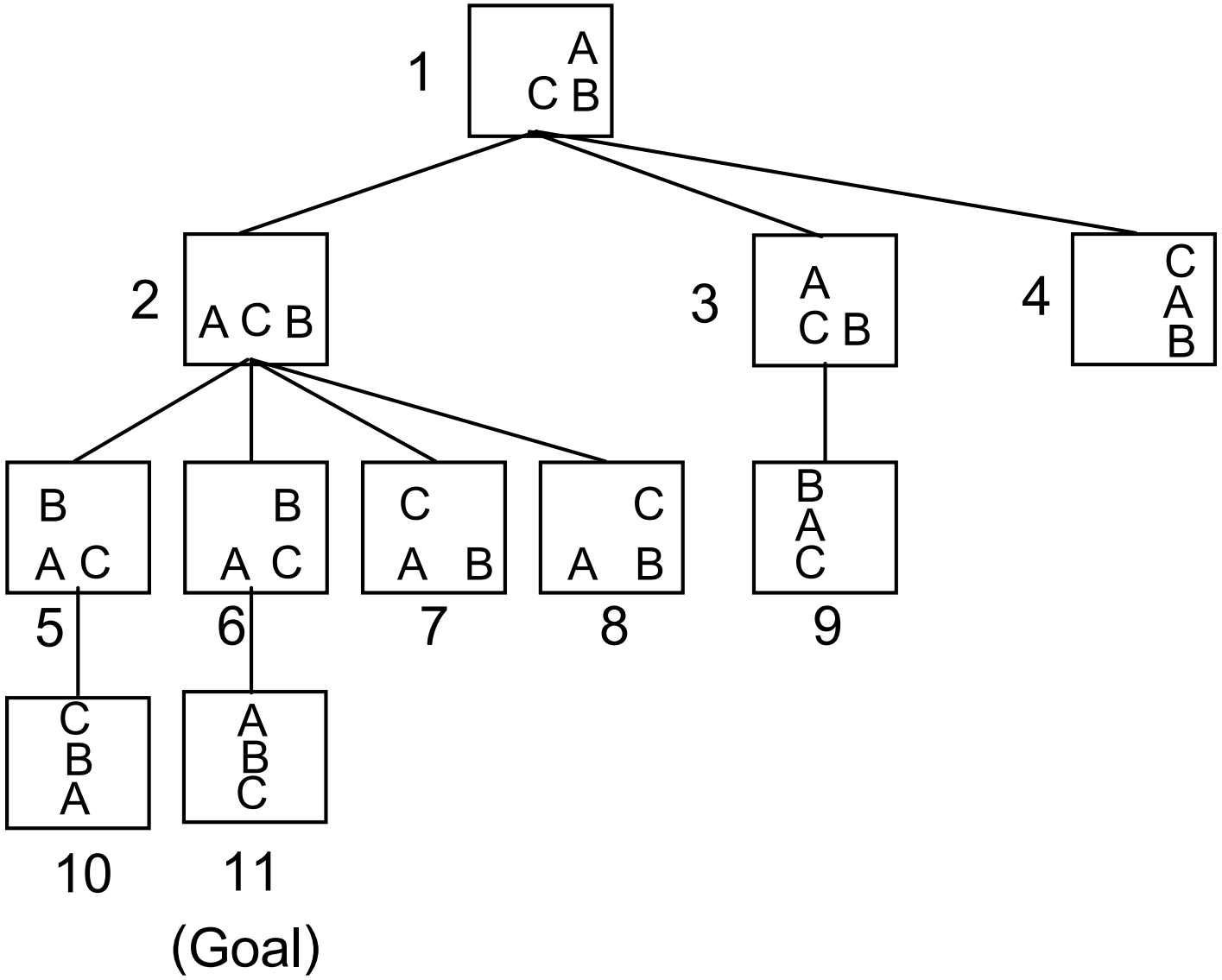
Block World Problem



Operators

1. block X ไม่มี block อื่นทับ -> วาง X บนโต๊ะ
2. block X และ Y ไม่มี block อื่นทับ -> วาง X บน Y

BFS of Block World Problem

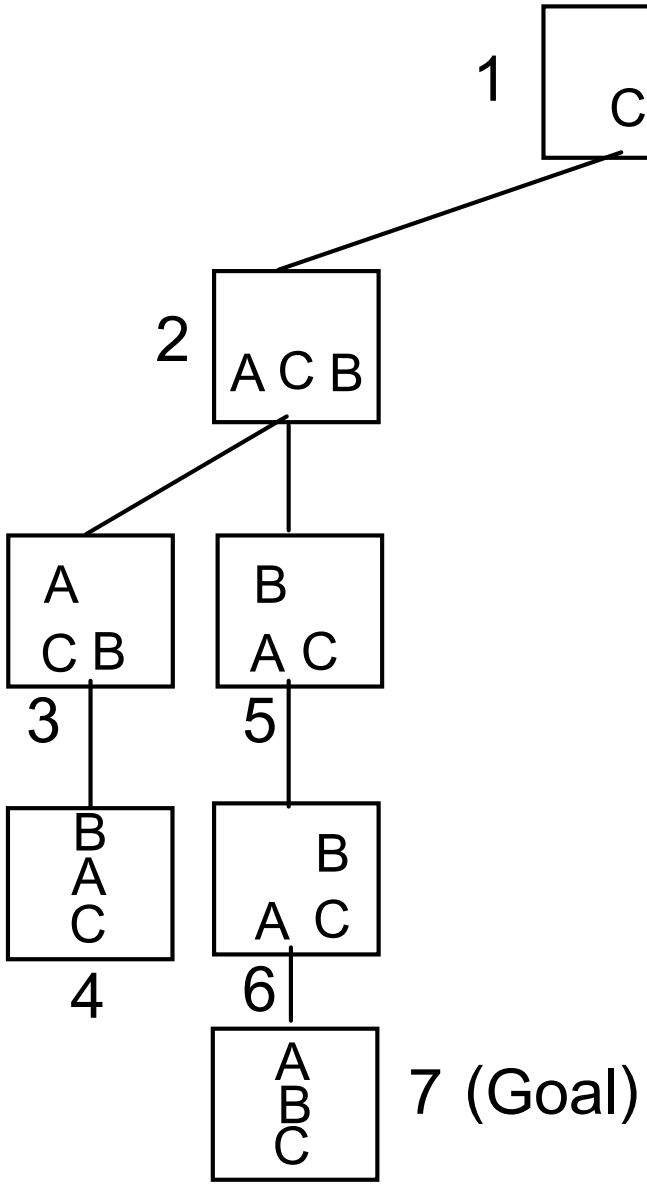


Breadth-First Search

Algorithm Breath-First Search

1. NODE-LIST := {initial state}
2. *UNTIL* ค้นพบ goal state หรือ NODE-LIST ว่าง *DO*
   2.1 ดึงสมาชิกตัวแรก(ให้ชื่อว่า E)ออกจาก NODE-LIST
   2.2 *FOR EACH* operator ที่ match กับ E *DO*
      2.2.1 ใช้ operator นั้นสร้าง state ใหม่
      2.2.2 *IF* state ใหม่เป็น goal state *THEN* quit และคืนค่า state นี้
         ELSE เพิ่ม state ใหม่นี้เข้าที่ท้ายของ NODE-LIST

DFS of Block World Problem



Depth-First Search

Algorithm Depth-First Search

1. *IF* initial state=goal state *THEN* quit และคืนค่า success

   ELSE UNTIL success หรือ failure *DO*

      1.1 สร้าง successor (ให้ชื่อว่า E) ของ initial state

         โดยoperator

         IF ไม่มี successor *THEN* คืนค่า failure

      1.2 เรียก Depth-First Search โดยให้ E เป็น initial state

      1.3 *IF* มีการคืนค่าของ success *THEN* คืนค่า success

         ELSE ทำซ้ำลูปนี้

Comparison Between BFS and DFS

ข้อดีของDepth-First Search

- ใช้ memory น้อยกว่า Breadth-First Search เพราะว่า states ใน current path เท่านั้นที่ถูกเก็บ
- ถ้าโชคดี Depth-First Search จะพบคำตอบโดยไม่ต้องค้นหา space มากเกินความจำเป็น แต่ถ้าเป็น Breadth-First Search states ที่ระดับ n+1 จะถูกกระจายออกมาก็ต่อเมื่อstatesที่ระดับ n ทุกตัวถูกกระจายหมดแล้วเท่านั้น

Comparison Between BFS and DFS (Cont.)

ข้อดีของBreadth-First Search

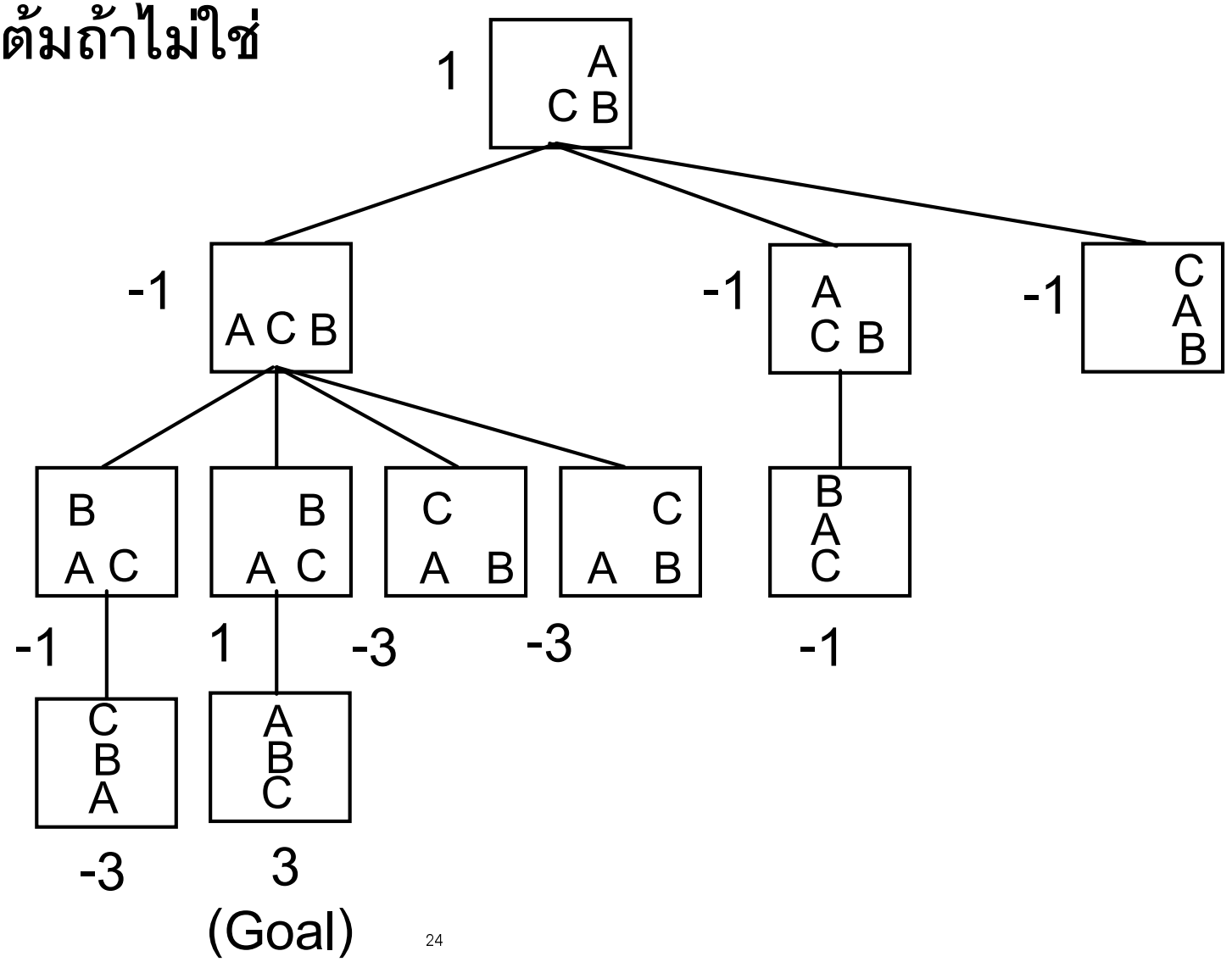
- จะไม่ติด path ที่ลึกมาก ๆโดยไม่พบคำตอบ
- ถ้ามีคำตอบ Breadth-First Search ประกันได้ว่าจะพบคำตอบแน่ ๆ และจะได้ path ที่สั้นที่สุดด้วย (สมมุติว่าระยะห่างหรือ cost ระหว่าง 2 states ใด ๆมีค่าเท่ากันหมด)
- สาเหตุที่สามารถรับประกันอย่างนี้ได้เพราะ paths ที่ยาวกว่าจะไม่ถูกกระจายออกมา ถ้า paths ที่สั้นกว่ายังกระจายไม่หมดทุก paths

Heuristic Search

- Heuristic เป็นเทคนิคที่ใช้เพิ่มประสิทธิภาพของกระบวนการ search โดยยอมให้ขาดความสมบูรณ์ (completeness)คืออาจไม่พบคำตอบ
- Heuristic function เป็นฟังก์ชั่นที่ map จาก state ไปยังตัวเลขที่ชี้ว่า state นี้เข้าใกล้ goal state มากเท่าไร (ยิ่งมากเท่าไรยิ่งมีโอกาสที่จะเปลี่ยนเป็น goal state มากเท่านั้น)
- Heuristic เป็นสิ่งที่ใช้ไกด์ search ว่าควรจะค้นหาไปในทิศทางใด
- ถ้า heuristic function ดี เราจะไม่ต้องกระจายstateที่ไม่จำเป็นในการนำไปสู่ goal state เลย
- แต่ถ้าไม่ดีอาจทำให้กระบวนการ search หลงไปในทิศทางที่ผิดได้

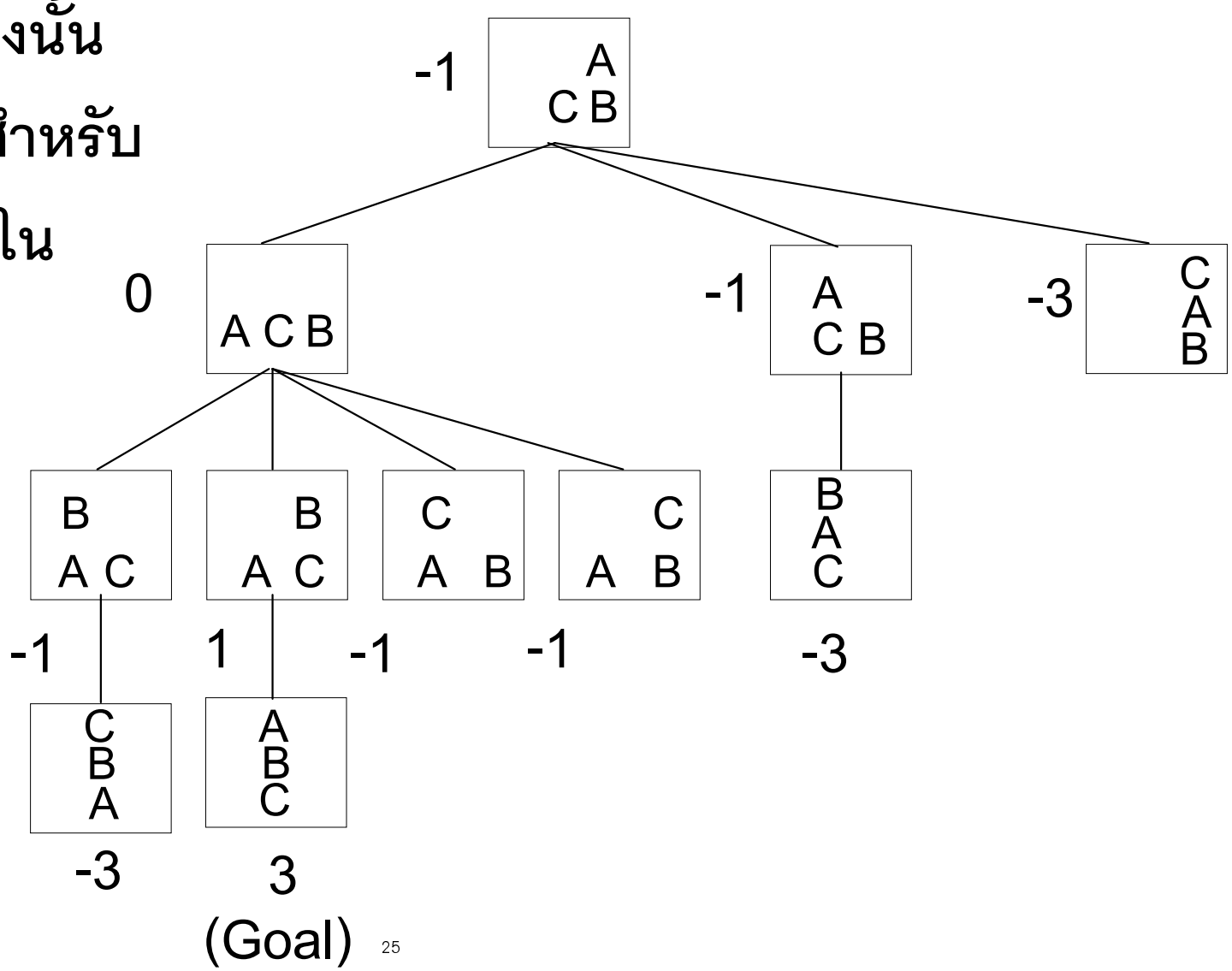
Examples of Heuristic Functions

h1 : บวก1แต้มให้กับทุกblockที่วางอยู่บนสิ่งที่มันควรอยู่
และลบ1แต้มถ้าไม่ใช่



Examples of Heuristic Functions

h2 : สำหรับทุกblocksที่อยู่บนโครงสร้างที่ถูก บวก1แต้มให้กับทุกblocksที่อยู่ในโครงสร้างนั้น และลบ1แต้มสำหรับทุกblocksที่อยู่ในโครงสร้างที่ผิด



Examples of Heuristic Functions

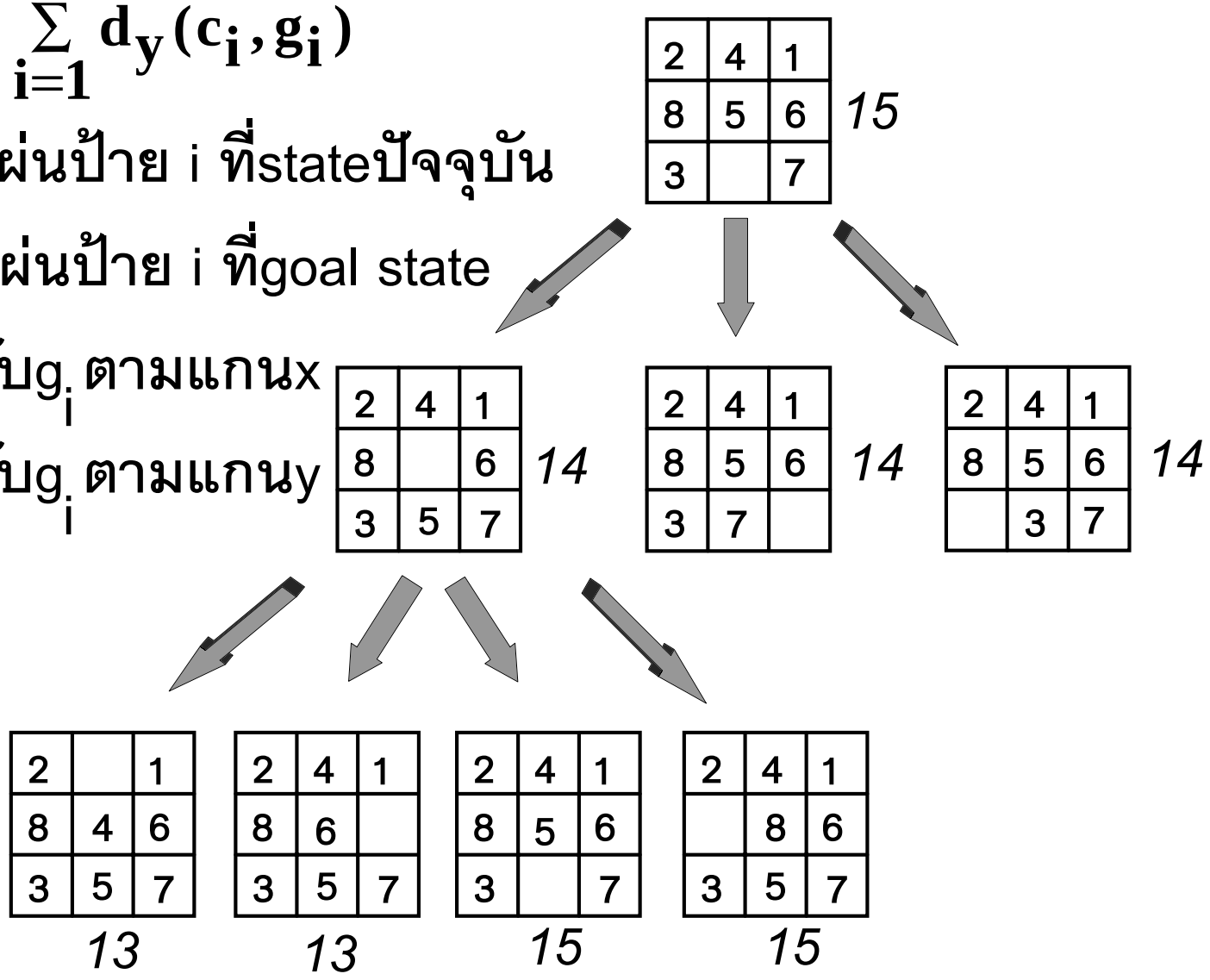
$$h = \sum_{i=1}^{8} d_x(c_i, g_i) + \sum_{i=1}^{8} d_y(c_i, g_i)$$

c_i - co-ordinateของแผ่นป้าย i ที่stateปัจจุบัน

g_i - co-ordinateของแผ่นป้าย i ที่goal state

d_x - ระยะห่างของc_iกับg_iตามแกนx

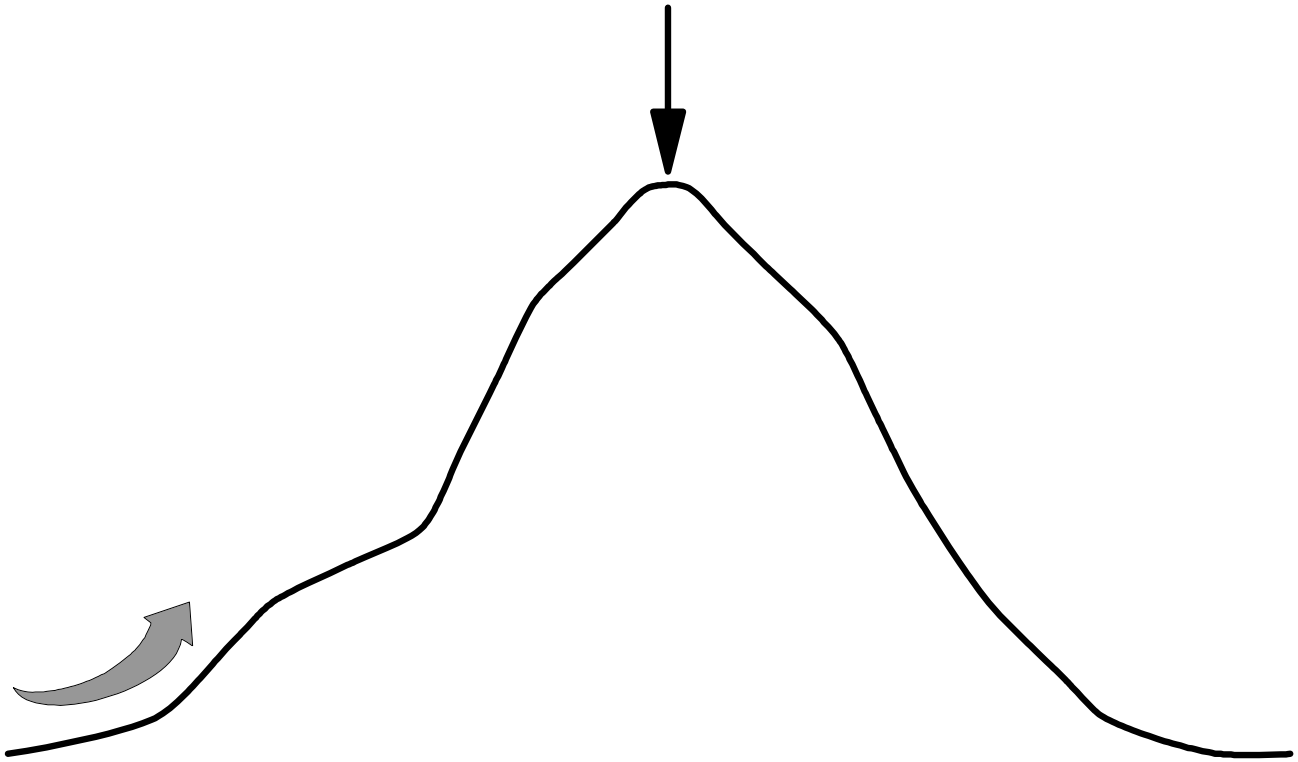
d_y - ระยะห่างของc_iกับg_iตามแกนy



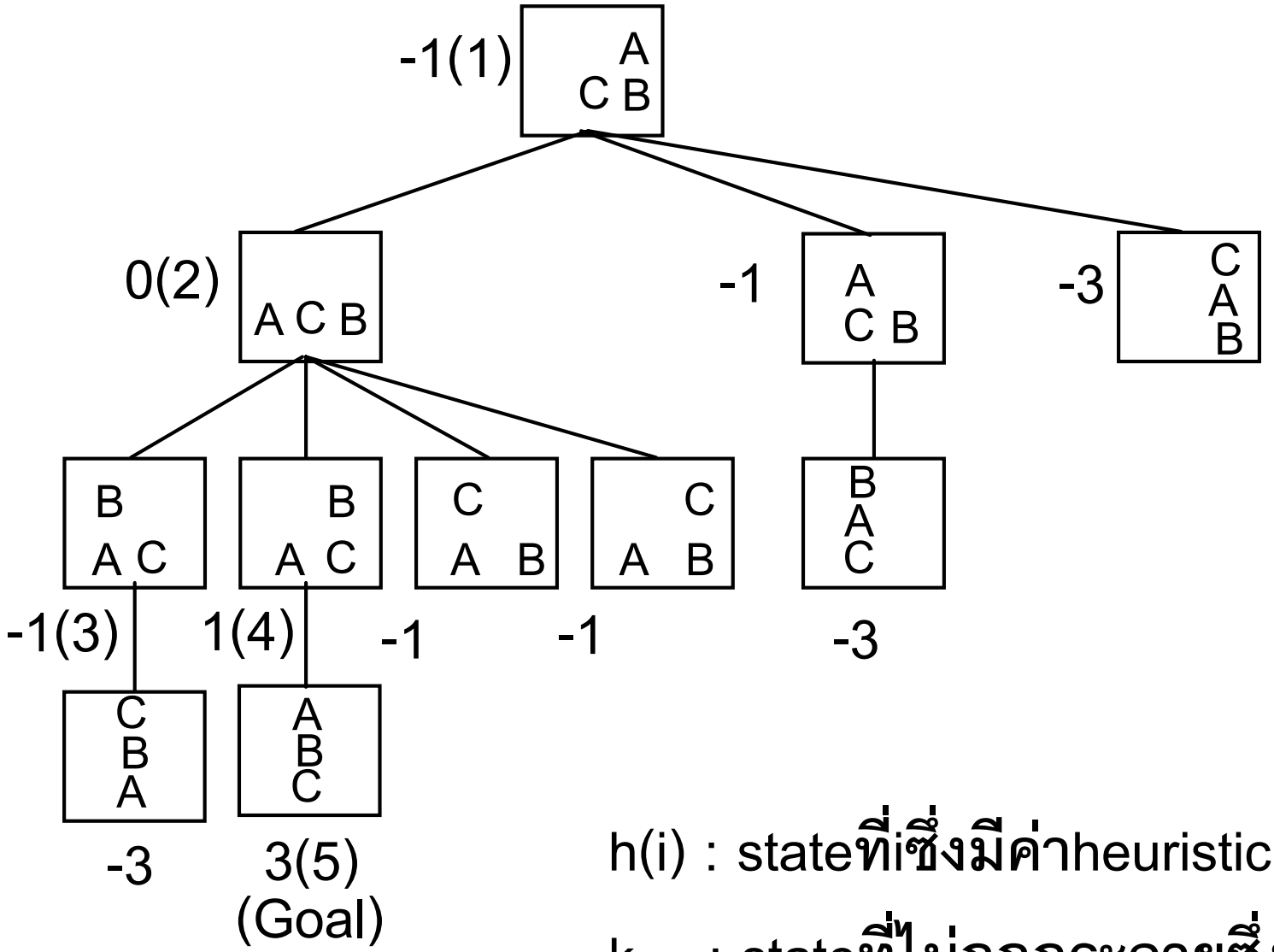
* กรณีนี้ค่าhยิ่งน้อยยิ่งดี

(ต้องการminimize)

Hill Climbing



An Example of Hill Climbing



h(i) : stateที่iซึ่งมีค่าheuristicเท่ากับh

k : stateที่ไม่ถูกกระจายซึ่งมีค่าเท่ากับk

Hill Climbing

Algorithm Simple Hill Climbing

1. Evaluate initial state

   IF initial state=goal state *THEN* คืนค่า initial state และ quit

   ELSE current state := initial state

2. *UNTIL* พบgoal state หรือ ไม่มี operator ที่ใช้เปลี่ยน current state ได้ *DO*

   2.1 เลือก operator ที่ยังไม่ได้ใช้กับ current state เพื่อผลิต new state

   2.2 Evaluate new state

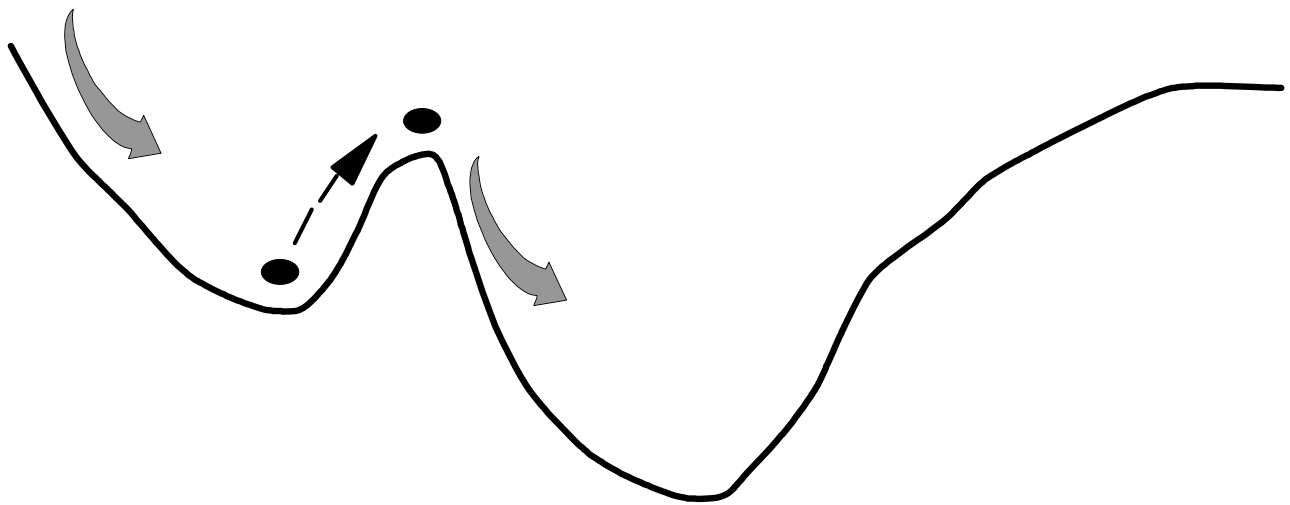
      IF new state=goal state *THEN* คืนค่า new state และ quit

      ELSE IF ค่า heuristic ของ new state ดีกว่า

         THEN current state := new state

         ELSE IF ค่า heuristic ของ new state ไม่ดีกว่า *THEN* ทำต่อ

Simulated Annealing



- เป็น variation หนึ่งของ hill climbing ซึ่งยอมให้ search ไปในทางที่
- ไม่ดีในช่วงเริ่มต้นของกระบวนการ search
- เป็น simulation ของ annealing ใน physics ซึ่งเป็นกระบวนการหลอมละลายของของแข็งจากจุดที่อุณหภูมิสูงและค่อย ๆเย็นลง แล้วจึงเข้าสู่สภาวะสุดท้ายซึ่งเป็นสภาวะที่มีพลังงานต่ำสุด

Simulated Annealing

- โดยทั่วไปสสารจะเปลี่ยนจาก higher energy state ไป lower energy state แต่ก็มีความน่าจะเป็น(p)ที่จะเป็นจาก lower energy state ไป higher energy state ตาม

$$p = e^{-\Delta E / kT}$$

โดยที่ ΔE คือ ระดับพลังงานที่เปลี่ยน(มีค่าเป็นตัวเลขบวก)

T คือ อุณหภูมิ

k คือ ค่าคงที่

- ให้ความน่าจะเป็นที่ search จะไปในทิศทางที่ไม่ดีด้วยความน่าจะเป็น

$$p = e^{-\Delta E / T}$$

โดยที่ ΔE คือ ค่าของheuristicที่เปลี่ยนไป,

T คือ อุณหภูมิ(annealing schedule)

Simulated Annealing

Algorithm Simulated Annealing

1. Evaluate initial state

   IF initial state=goal state *THEN* คืนค่า initial state และ quit

   ELSE current state := initial state

2. BEST-SO-FAR := current state

3. T := constant

4. *UNTIL* พบคำตอบ หรือ ไม่มี operator ที่ใช้เปลี่ยน current state ได้ *DO*

   4.1 เลือก operator ที่ยังไม่ได้ใช้กับ current state เพื่อผลิต new state

   4.2 Evaluate new state

Simulated Annealing

dE := (ค่าheuristicของcurrent state)- (ค่าheuristicของnew state)

IF new state=goal state *THEN* คืนค่า new state และ quit

ELSE IF ค่า heuristic ของ new state ดีกว่าของ current state

THEN current state := new state;

IF ค่า heuristic ของ new state ดีกว่าของ BEST-SO-FAR

THEN BEST-SO-FAR := new state

ELSE ให้ current state มีค่าเป็น new state ด้วยความน่าจะเป็น

เท่ากับ p

IF p มีค่ามากกว่าค่า random(0ถึง1)

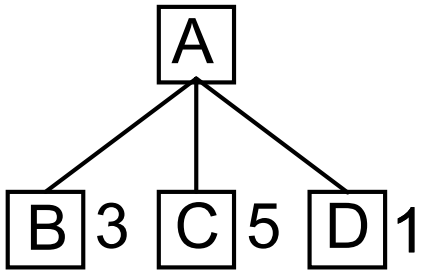
THEN current state := new state

4.3 เปลี่ยนค่า T ตามความจำเป็น

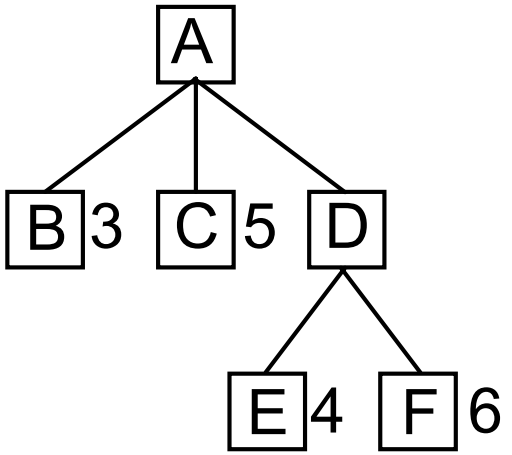
5. คืนค่า BEST-SO-FAR เป็นคำตอบ

Best-First Search

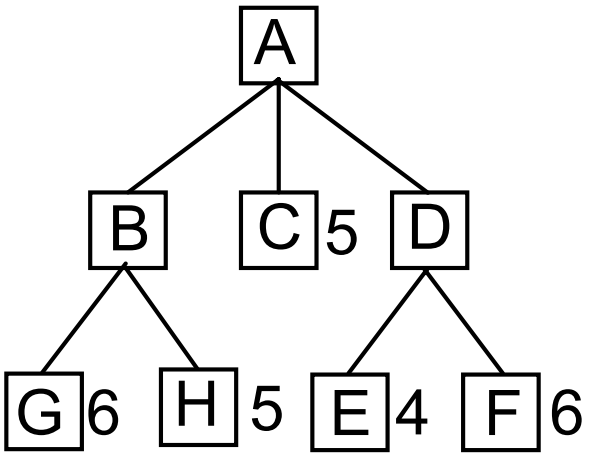
step 1



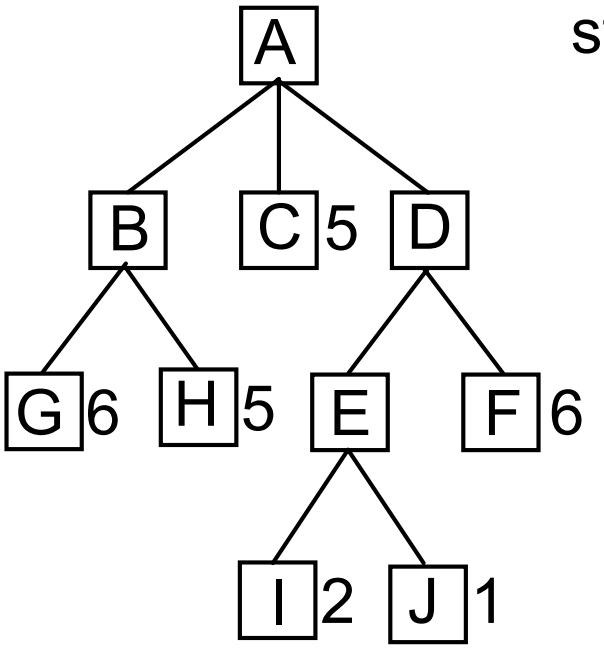
step 2



step 3



step 4



step 5

* ต้องการminimize

Best-First Search

Algorithm Best-First Search

1. OPEN := {initial state}
2. *UNTIL* พบ goal state หรือ ไม่มี state เหลืออยู่ใน OPEN *DO*
   2.1 ดึง state ที่มีค่า heuristic ดีที่สุดออกจาก OPEN และให้ชื่อว่า X
   2.2 *IF* (X=goal state) *THEN* คืนค่า X เป็นคำตอบ
   2.3 สร้าง successors ของ state นั้น
   2.4 *FOR EACH* successor *DO*
      2.4.1 *IF* successor ยังไม่เคยถูกสร้างขึ้นมาเลย *THEN*
         เติม successor ใส่ใน OPEN และจำ parent state ไว้
      2.4.2 *IF* successor เคยถูกสร้างขึ้นมาแล้วและpathใหม่นี้ดีกว่าเดิม
         THEN เปลี่ยน parent state และ update cost ใหม่

Best-First Search

- Best-First Search ทำงานคล้ายกับ hill-climbing โดยมีข้อแตกต่างคือ
  - ในกรณีของ hill-climbing จะเลือก state ที่ดีที่สุดและเดินไปทางนั้น โดยที่ states อื่น ๆที่ถูกสร้างจาก parent state เดียวกันจะถูกทิ้งไป แต่ในกรณีของ Best-First Search จะเก็บ states เหล่านี้ไว้ เพื่อใช้ในอนาคต เมื่อ path ที่เดินไปไม่ดีเท่า states เหล่านี้
  - Hill-climbing จะหยุดเมื่อ successors มีค่าไม่ดีเท่า current state แต่ในกรณีของ Best-First Search state ที่มีค่า heuristic ดีที่สุดในปัจจุบันจะถูกเลือกแม้ว่าจะมีค่าไม่ดีเท่า state ที่เคยเลือกก็ตาม

A* Algorithm

- Best-First Search เป็น algorithm อย่างง่ายของalgorithm A*
- ในบางปัญหาเราแยกheuristic function(f' -- เป็นฟังก์ชั่นโดยประมาณของ f ซึ่งเป็นฟังก์ชั่นจริงที่ไม่รู้)ของ state s ใด ๆออกเป็น 2 ส่วนคือ

$$f'(s) = g(s) + h'(s)$$

โดยที่ g คือฟังก์ชั่นที่คำนวณ cost จาก initial state ถึง current state

h' คือฟังก์ชั่นที่ประมาณ(estimate)cost จาก current state ถึง goal state

f' จึงเป็นฟังก์ชั่นที่ประมาณ cost จาก initial state ถึงgoal state (stateที่ดี f'จะมีค่าน้อย)

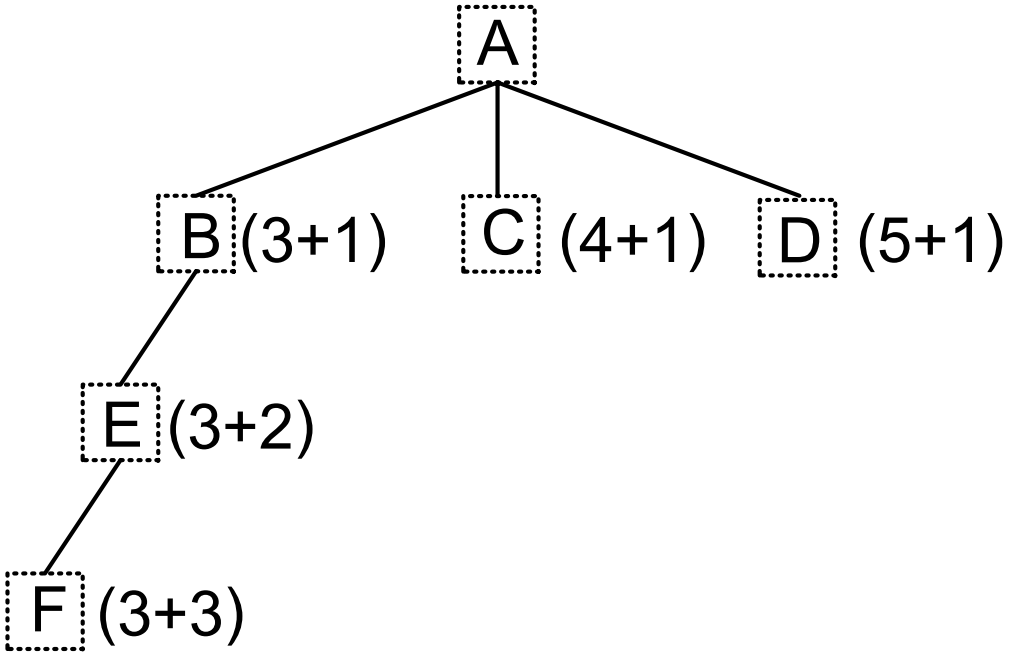
A* Algorithm

จุดเด่นของA*ที่แตกต่างจากBest-First Search

- การใช้ g ร่วมในการประมาณค่าf'ทำให้การเลือก state ไม่เพียงพิจารณาแค่ว่า state ดีเท่าไร(ซึ่งประมาณจากh') แต่ยังรวมไปถึงว่า path ที่นำไปสู่ state ดีแค่ไหนด้วย
- โดยการใช้ g state ที่ประมาณว่าใกล้ goal stateมากที่สุดอาจจะไม่ถูกกระจายก็ได้
- โดยปกติ A*จะหาคำตอบซึ่งpathที่นำไปสู่คำตอบเสีย cost น้อยที่สุด
- ถ้าให้ g ของทุก state เป็น 0 state ที่ใกล้ goal state จะถูกกระจายก่อน
- ถ้าต้องการหาคำตอบโดยให้ step น้อยที่สุด ก็ให้ g ของทุก step เป็น 1

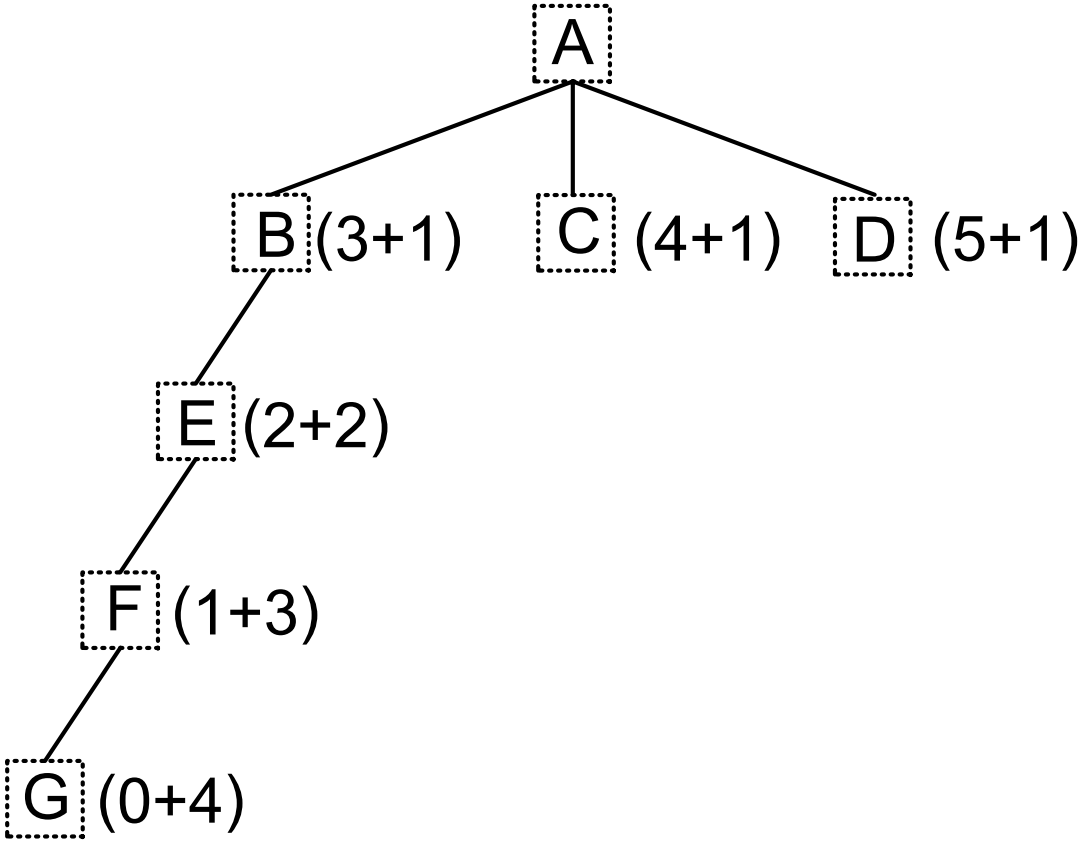
A* Algorithm

- ถ้า h' เป็น under-estimating function ของ h(ฟังก์ชั่นที่คืนค่า cost จาก current state ถึง goal state ซึ่งค่าที่คืนนั้นถูกต้องเสมอ) เราประกันได้ว่า path ที่ได้จาก A*นั้นเป็น optimal path



h' underestimates h

A* Algorithm



h' overestimates h

Means-Ends Analysis

- Means-ends analysis เป็นเทคนิคในการแก้ปัญหาแบบหนึ่ง ซึ่งแตกต่างจาก state-space search ตรงที่วิธีเลือก path ต่อไปที่จะลองกระจาย
- สมมุติฐานของวิธีนี้คือ
  - สามารถนิยามความแตกต่าง(difference)ระหว่าง current state กับ goal state
  - สามารถแบ่ง operators ตามประเภทของความแตกต่างซึ่ง operators เหล่านี้สามารถทำให้ลดลงหรือหมดไป
- เมื่อพบความแตกต่างระหว่าง current state กับ goal state แล้ว operator ที่สามารถลดความแตกต่างจะถูกเลือก

Means-Ends Analysis

- ในกรณีที่ไม่สามารถใช้ operator ได้โดยตรง จะตั้ง subproblem ใหม่ที่สามารถใช้ operator นี้
- โปรแกรมทางAIที่ใช้ means-ends analysis ตัวแรกคือ General Problem Solver(GPS)
- Means-ends analysis จะต้องมีเซ็ตของ rules ที่จะเปลี่ยน state หนึ่งๆ ไป state ใหม่และมีโครงสร้างข้อมูลที่เรียกว่า difference table
- Operatorประกอบด้วย
  - precondition : เงื่อนไขที่จะใช้ operator
  - result : อธิบาย state ใหม่ที่เปลี่ยนไปจากเดิมหลังจากใช้ operator
- Difference table ใช้กำหนด operator ที่สามารถลดความแตกต่างที่ให้มา

Means-Ends Analysis (Example)

ตัวอย่างปัญหา :

- หุ่นยนต์(robot)ต้องการเคลื่อนย้ายโต๊ะ(desk)ที่มีสิ่งของ 2อย่างอยู่ข้างบนจากห้องหนึ่งไปอีกห้องหนึ่ง และสิ่งของทั้ง2ต้องถูกย้ายไปด้วย
- Operator และ difference table ที่ใช้แสดงในรูปหน้าถัดไป
- ความแตกต่างที่สำคัญระหว่าง start state และ goal state คือตำแหน่งของโต๊ะ ซึ่งอาจลดได้โดย PUSH หรือ CARRY
- ถ้าเลือก CARRY, จะเกิด subgoals คือ at(robot,desk) $\wedge$ small(desk) ซึ่งsubgoal at(robot,desk)ทำสำเร็จโดยWALK แต่subgoal small(desk) ไม่สามารถทำให้สำเร็จได้

ตัวอย่างOperatorsในMeans-Ends Analysis

Opertor	*Precondition*	*Results*
PUSH(obj,loc)	at(robot,obj) ∧	at(obj,loc) ∧
	large(obj) ∧	at(robot,loc)
	clear(obj) ∧	
	armempty	
CARRY(obj,loc)	at(robot,obj) ∧	at(obj,loc) ∧
	small(obj)	at(robot,loc)
WALK(loc)	none	at(robot,loc)
PICKUP(obj)	at(robot,obj)	holding(obj)
PUTDOWN(obj)	holding(obj)	¬ holding(obj)
PLACE(obj1,obj2)	at(robot,obj2) ∧	on(obj1,obj2)
	holding(obj1)	

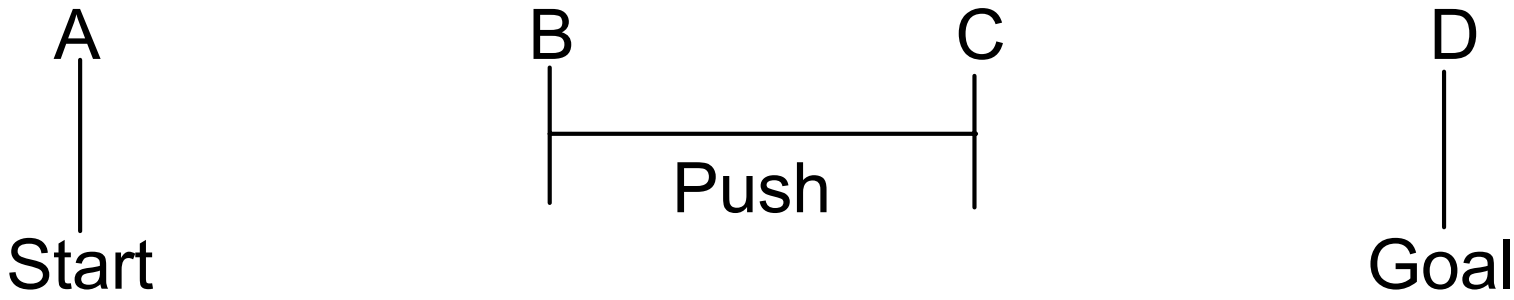
ตัวอย่างของDifference Table

-

	Push	Carry	Walk	Pickup	Putdown	Place
Move object	*	*				
Move robot			*			
Clear object				*		
Get object on object						*
Get arm empty					*	*
Be holding object				*		

Means-Ends Analysis (Example)

- จากการเลือกPUSH ความแตกต่างระหว่างAกับB และCกับDจะต้องถูกกำจัดให้หมดไป



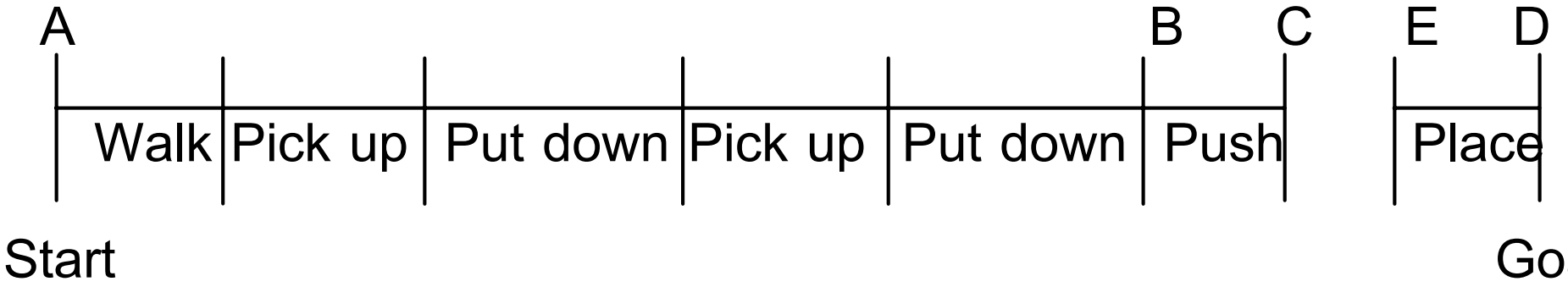
- ในจำนวน preconditions 4ตัวของPUSHนั้น at(robot,desk) และ clear(desk) จะเป็น subgoals ต่อไป
- at(robot,desk) ทำสำเร็จได้โดย WALK
- clear(desk) ทำสำเร็จได้โดย PICKUP,PUTDOWN,PICKUP,PUTDOWN

Means-Ends Analysis (Example)

- และใช้ PLACE เพื่อย้ายสิ่งของ ซึ่งมี preconditions คือ

  at(robot,desk) ∧ holding(thing1)

  ที่จุดนี้ความแตกต่างระหว่าง C และ E จะต้องถูกกำจัดต่อไป



- ความแตกต่างระหว่าง C กับ E สามารถกำจัดโดยใช้ WALK, PICKUP และ CARRY

Means-Ends Analysis (Algorithm)

Algorithm : MEA(CURRENT,GOAL)

1. *IF* CURRENT=GOAL *THEN* return success
2. *IF* CURRENT แตกต่างจาก GOAL *THEN*

   เลือกความแตกต่างที่สำคัญที่สุดและกำจัดความแตกต่างนั้นโดย

   UNTIL success or failure *DO*

   2.1 เลือก operator O ที่ยังไม่ได้ลองและสามารถใช้ลดความแตกต่างนี้ได้ ถ้าไม่มีให้คืนค่า failure

   2.2 ใช้ O กับ CURRENT โดยสร้าง descriptions ของ 2 states

   O-START : state ที่ preconditions ของ O satisfy

   O-RESULT : state ที่จะเกิดถ้าใช้ O กระทำกับ O-START

   2.3 *IF* (FIRST-PART <-- MEA(CURRENT,O-START)) and

   (LAST-PART <-- MEA(O-RESULT,GOAL)) success

   THEN คืนค่า ผลของ concat FIRST-PART,O และ LAST-PART