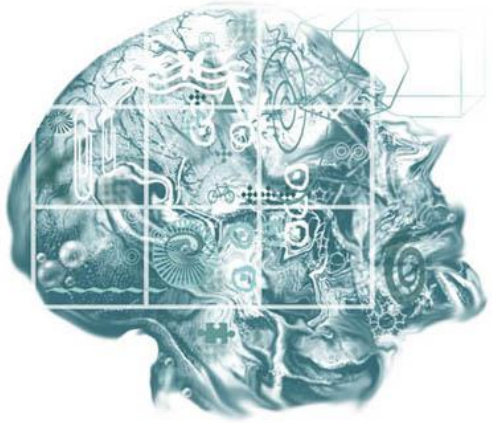


บทที่ 2 ทักษะพื้นฐานการแก้ปัญหาด้วยการค้นหา (Problem-Solving with Search)

นำเสนอโดย
ผู้ช่วยศาสตราจารย์จุฑาวุฒิ จันทรมาลี
หลักสูตรวิทยาการคอมพิวเตอร์
มหาวิทยาลัยสวนดุสิต

เนื้อหาในบทที่ 2 ประกอบด้วย

2.1 ทักษะพื้นฐานในการแก้ปัญหา

2.2 เทคนิคการค้นหา

1. ทักษะพื้นฐานในการแก้ปัญหา

การแก้ปัญญาส่วนใหญ่ในระบบชาญฉลาด จะมองในรูปแบบของการค้นหา (search) เป็นหลัก เช่น

1. การเรียนรู้ของเครื่อง (Machine Learning)

2. การประมวลผลภาษาธรรมชาติ (Natural Language Processing) เป็นต้น

ซึ่งส่วนใหญ่จะใช้การสร้างปริภูมิ (space) ขึ้นมาหนึ่งปริภูมิ โดยสมาชิกในปริภูมิจะแทนตัวเลือกของคำตอบ จากนั้นก็จะทำการค้นหาโดยวิธีการค้นหาที่ดีหรือเหมาะสมที่สุดที่ผู้ใช้งานเลือกนำมาใช้

1. ทักษะพื้นฐานในการแก้ปัญหา

ทักษะที่ใช้ในการแก้ปัญหา คือ

1. นิยามปัญหาอย่างชัดเจน เช่น Initial-->Final Situation เป็นต้น

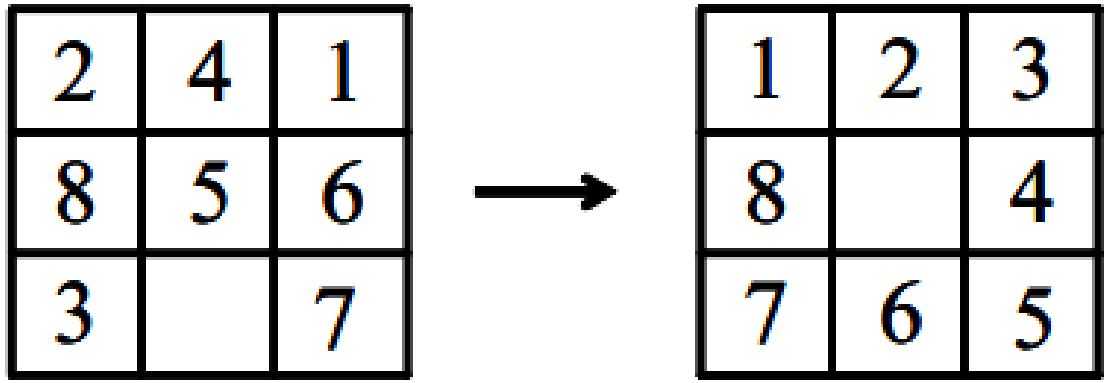
2. วิเคราะห์ปัญหา

3. หาความรู้ที่ใช้ในการแก้ปัญหาว่ามีอะไรบ้าง

4. เลือกเทคนิคการแก้ปัญหาที่เหมาะสม

1. ทักษะพื้นฐานในการแก้ปัญหา

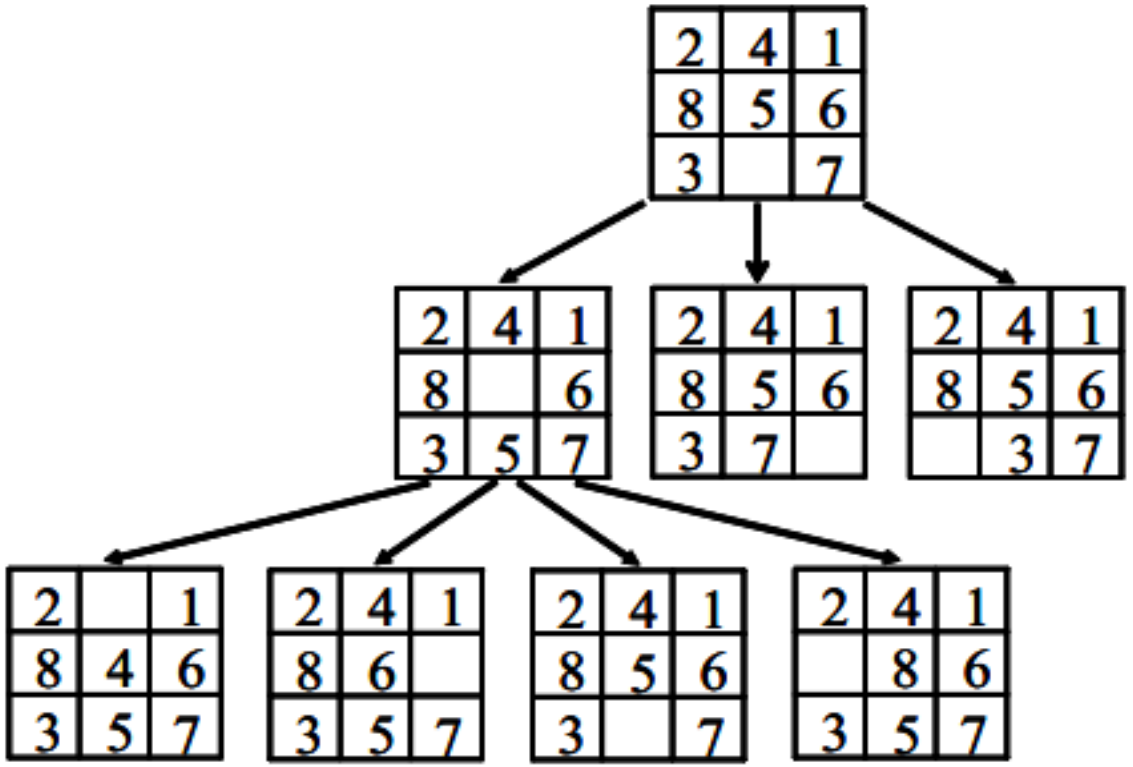
ปัญหา 8-Puzzle



รูปที่ 2–1 8-Puzzle

1. ทักษะพื้นฐานในการแก้ปัญหา

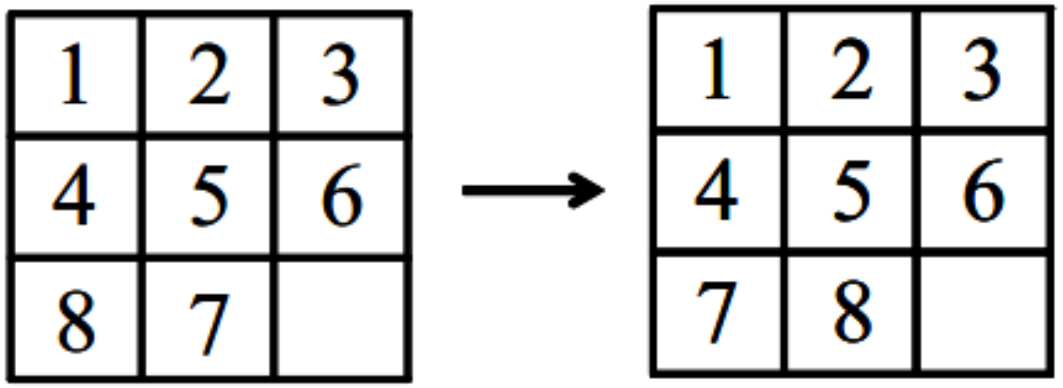
ปัญหา 8-Puzzle



รูปที่ 2–2 บางส่วนของปริภูมิสถานะในปัญหา 8-Puzzle

1. ทักษะพื้นฐานในการแก้ปัญหา

ปัญหา 8-Puzzle



รูปที่ 2–3 ตัวอย่างคำตอบที่เข้าถึงไม่ได้ด้วยสถานะเริ่มต้น

2. เทคนิคการค้นหา

เทคนิคการค้นหา สามารถแบ่งประเภท ได้ดังนี้

1. การค้นหาแบบบอด (Blind Search)

    1.1 การค้นบางส่วน (Partial Search)

        1.1.1 การค้นหาแนวกว้างก่อน (Breadth-First-Search)

        1.1.2 การค้นหาแนวลึกก่อน (Depth-First-Search)

    1.2 การค้นหาทั้งหมด (Exhaustive Search)

2. เทคนิคการค้นหา

เทคนิคการค้นหา สามารถแบ่งประเภท ได้ดังนี้

2. การค้นหาแบบฮิวริสติก (Heuristic Search)

2.1 อัลกอริทึมแบบปีนเขา (Hill-Climbing Search)

2.2 อัลกอริทึมแบบอบเหนียวจำลอง (Simulated annealing Search)

2.3 การค้นหาดีที่สุดก่อน (Best-First-Search)

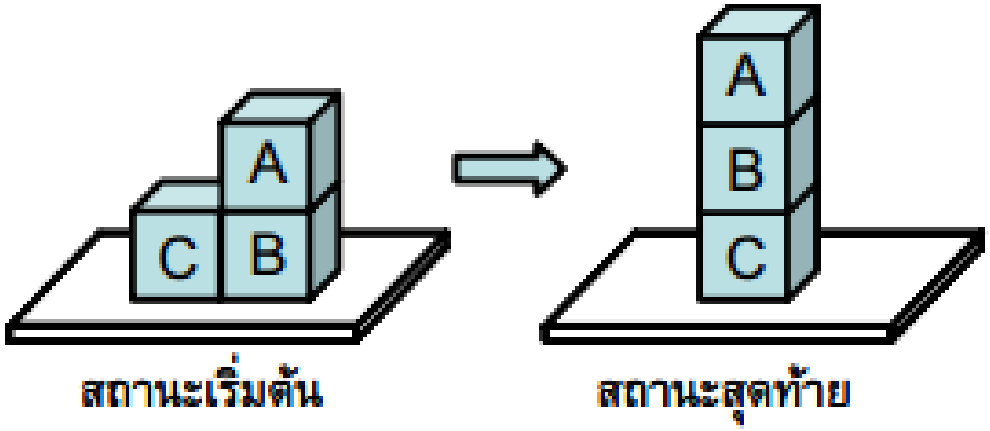
2.4 การค้นหาแบบ A* (A* Search)

2.5 วิธีอื่น ๆ

2. เทคนิคการค้นหา

1. การค้นหาแบบบอด (Blind Search) โดยเริ่มการค้นหาในแนวกว้างก่อน แล้วตามด้วยการค้นในแนวลึก

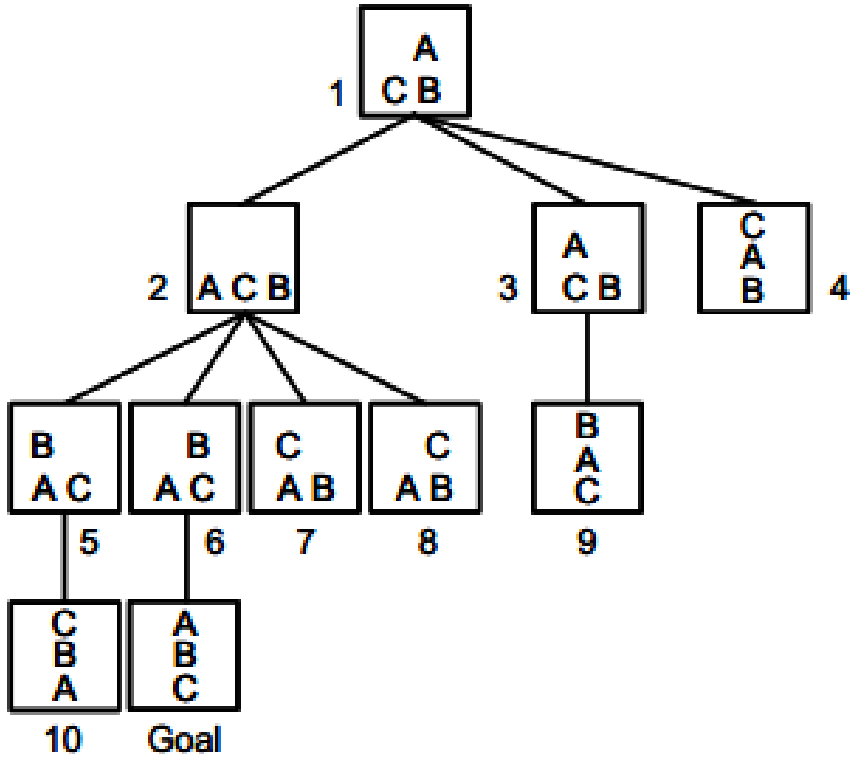
Ex. ปัญหาโลกของบล็อก (Block World Problem)



รูปที่ 2–4 ปัญหาโลกของบล็อก

2. เทคนิคการค้นหา

1.1.1 การค้นหาแนวกว้างก่อน (Breadth-First-Search)



รูปที่ 2–5 การค้นหาแบบแนวกว้างก่อนในปัญหาโลกของบล็อก

2. เทคนิคการค้นหา

ตารางที่ 2-1 อัลกอริทึมการค้นหาแนวกว้างก่อน

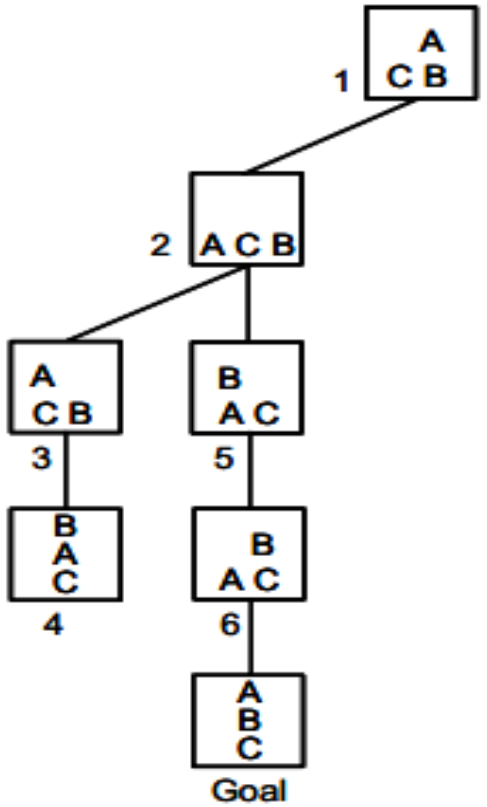
```
Algorithm: Breadth-First Search

1. Node-list := {initial state}
2. UNTIL a goal state is found or Node-list is empty DO
   2.1 Remove the first element from Node-list and call
       it E.
   2.2 For Each operator matching E DO
       2.2.1 Apply the operator to generate a new state.
       2.2.2 IF the new state is a goal state THEN quit
                 and return this state
             ELSE add the state to the end of Node-list.
```

หมายเหตุ อัลกอริทึมด้านบนนี้ไม่ได้ตรวจสอบสถานะซ้ำ

2. เทคนิคการค้นหา

1.1.2 การค้นหาแนวลึกก่อน (Depth-First-Search)



รูปที่ 2–6 การค้นหาแบบแนวลึกก่อนในปัญหาโลกของบล็อก

2. เทคนิคการค้นหา

ตารางที่ 2-1 อัลกอริทึมการค้นหาแนวลึกก่อน

```
Algorithm: Depth-First Search

1. IF initial state = goal state THEN quit and return
      success
   ELSE UNTIL success or failure DO
   1.1 Generate a successor, E, of the initial state
       IF there are no more successors
       THEN return failure
       ELSE call Depth-First Search with E as the
            initial state.
   1.2 IF success is return THEN return success
       ELSE continue in this loop.
```

หมายเหตุ อัลกอริทึมด้านบนนี้ไม่ได้ตรวจสอบสถานะซ้ำ

2. เทคนิคการค้นหา

ตารางที่ 2-2 เปรียบเทียบอัลกอริทึมการค้นหาแนวลึกก่อนและการค้นหาแบบแนวกว้างก่อน

การค้นหาแนวลึกก่อน	การค้นหาแนวกว้างก่อน
1. ใช้หน่วยความจำน้อยกว่าการค้นหาแนวกว้างก่อน เพราะว่าสถานะในเส้นทางค้นหาปัจจุบันเท่านั้นที่ถูกเก็บ (ในขณะใดๆ จะเก็บเส้นทางเดียว พอไปเส้นทางอื่น เส้นทางที่ผ่านมาก็ไม่จำเป็นต้องเก็บไว้อีก)	1. ใช้หน่วยความจำมากกว่า เพราะต้องเก็บสถานะไว้ทุกตัว เพื่อหาเส้นทางจากสถานะเริ่มต้นไปคำตอบ

2. เทคนิคการค้นหา

ตารางที่ 2-2 เปรียบเทียบอัลกอริทึมการค้นหาแนวลึกก่อนและการค้นหาแบบแนวกว้างก่อน

2. อาจติดเส้นทางที่ลึกมาก ๆ โดยไม่พบคำตอบ เช่นในกรณีที่เส้นทางนั้นไม่มีคำตอบ และเป็นเส้นทางที่ยาวไม่สิ้นสุด ซึ่งการค้นหาแบบนี้ จะไปเส้นทางอื่นไม่ได้ (เช่นกรณีของ Prolog ซึ่งใช้วิธีค้นหาแบบนี้ จะทำการค้นหาไปเรื่อยๆ จนกว่าสถานะที่สร้างขึ้น ใช้หน่วย-ความจำเกิน ซึ่งจะเกิดข้อผิดพลาดขึ้น วิธีแก้ไขที่ใช้ใน Prolog คือให้ผู้ใช้กำหนดความลึกในการค้นหาคำตอบจะได้ไม่เสียเวลา ในการค้นหานานเกินจำเป็น เช่นกำหนดให้ความลึกในการค้นหาในแต่ละเส้นทางไม่มากกว่า 100 ขั้นตอน เป็นต้น หรืออาจจะกำหนดที่ขนาดของหน่วยความจำก็ได้)	2. จะไม่ติดเส้นทางที่ลึกมากๆ โดยไม่พบคำตอบ

2. เทคนิคการค้นหา

ตารางที่ 2-2 เปรียบเทียบอัลกอริทึมการค้นหาแนวลึกก่อนและการค้นหาแบบแนวกว้างก่อน

3. ถ้าคำตอบอยู่ที่ระดับ n+1 สถานะอื่นทุกตัวที่อยู่ที่ระดับ 1 ถึงระดับ n ไม่จำเป็นต้องถูกกระจายจนหมด	3. ถ้าคำตอบอยู่ที่ระดับ n+1 สถานะทุกตัวที่ระดับ 1 ถึงระดับ n จะต้องถูกกระจายจนหมด ทำให้มีสถานะที่ไม่จำเป็นในเส้นทางที่จะไปสู่คำตอบถูกกระจายออกด้วย
4. เมื่อพบคำตอบ ไม่สามารถรับประกันได้ว่าเส้นทางที่ได้เป็นเส้นทางสั้นสุดหรือไม่	4. ถ้ามีคำตอบจะประกันได้ว่าจะพบคำตอบแน่ๆ และจะได้เส้นทางสั้นสุดด้วย (สมมติว่าระยะห่างหรือต้นทุนระหว่างสถานะ 2 ตัวใดๆ มีค่าเท่ากันหมด)

2. เทคนิคการค้นหา

ปัญหาการเดินทางของพนักงานขาย (Traveling Salesman Problem)

รูปที่ 2–7 ปัญหาการเดินทางของพนักงานขาย

2. เทคนิคการค้นหา

2. การค้นหาแบบฮิวริสติก (Heuristic Search) จะใช้รูปแบบ *ฮิวริสติก (ความรู้ที่ไม่สมบูรณ์หรือการคาดเดาอย่างมีเหตุผล)* มาช่วยในการค้นหาให้มีประสิทธิภาพมากขึ้น โดยวิธีนี้จะช่วยชี้แนะกระบวนการเลือกเส้นทางใดหรือสถานะใดเพื่อทำการค้นหาต่อไปให้ได้คำตอบอย่างมีประสิทธิภาพ พิจารณาจาก *ปัญหาการเดินทางของพนักงานขาย (Traveling Salesman Problem)*

2. เทคนิคการค้นหา

ลักษณะเด่นของการค้นหาแบบฮิวริสติก

1. เป็นเทคนิคที่ใช้เพิ่มประสิทธิภาพของกระบวนการค้นหา โดยอาจจะต้องยอมให้ขาดความสมบูรณ์ไปบ้าง คือ ไม่พบคำตอบที่ถูกต้อง แม้ว่าปริภูมิสภานะจะมีคำตอบนี้อยู่

2. การนำฮิวริสติกมาใช้จะต้องนำมาใช้ในรูปแบบที่วัดค่าได้อย่างง่าย ซึ่งมักจะทำโดยนิยามฮิวริสติกให้อยู่ในรูปแบบของฟังก์ชัน เรียกว่า *ฮิวริสติกฟังก์ชัน (Heuristic function)* ซึ่งเป็นฟังก์ชันที่คำนวณค่าสถานะไปยังตัวเลขที่ชี้ว่าสถานะนั้นเข้าใกล้สถานะเป้าหมายมากเท่าไร (ยิ่งมากเท่าไร ยิ่งมีโอกาสที่จะเปลี่ยนสถานะเป้าหมายมากเท่านั้น) การค้นหาจึงมุ้งไปเส้นทางที่มีค่าฟังก์ชันฮิวริสติกที่ดีกว่า

2. เทคนิคการค้นหา

ลักษณะเด่นของการค้นหาแบบฮิวริสติก

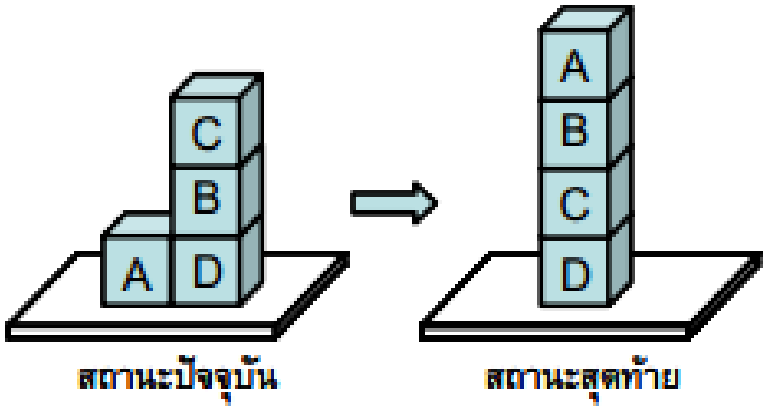
3. ฟังก์ชันฮิวริสติกนี้เป็นสิ่งที่ใช้ชี้แนะกระบวนการค้นหาว่าควรจะค้าหาไปในทิศทางใด ซึ่งกระบวนการค้นหาที่ใช้ฟังก์ชันฮิวริสติกสามารถถอบแบบได้หลายชนิด

4. ในบางกรณี สามารถนิยามฟังก์ชันฮิวริสติกได้อย่างสมบูรณ์แบบ การค้นหาก็จะมุ้งตรงไปสถานะเป้าหมายโดยไม่ผิดเส้นทางเลย แต่ถ้าฟังก์ชันฮิวริสติกไม่ดีก็อาจทำให้กระบวนการค้นหาหลงไปในทิศทางที่ผิดได้ ทำให้คำตอบที่ได้เมื่อใช้ฟังก์ชันฮิวริสติก ก็จะไม่ใช่คำตอบที่ดีที่สุด

2. เทคนิคการค้นหา

Ex. ฟังก์ชันฮิวริสติก h1 สำหรับปัญหาโลกของบล็อก

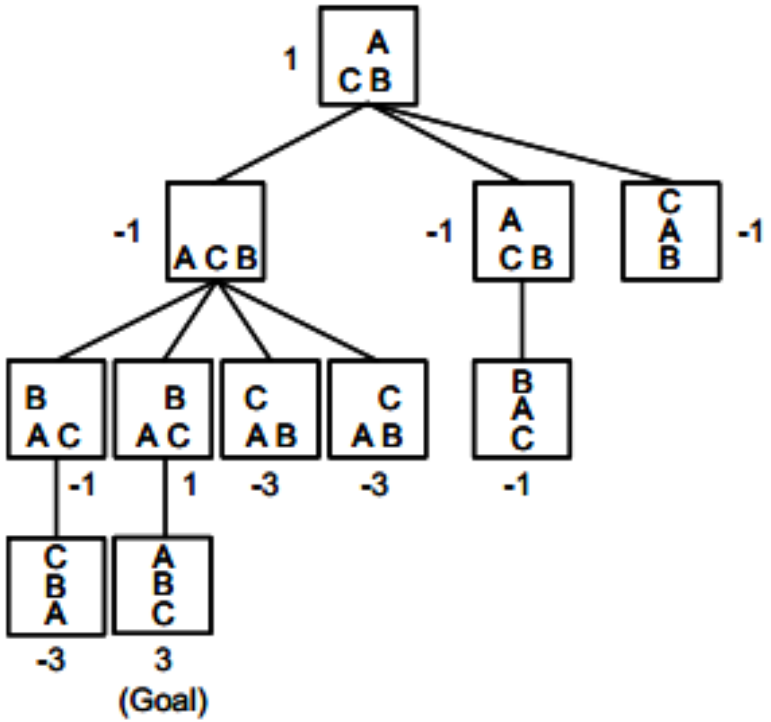
h1: บวกหนึ่งให้กับทุกบล็อกที่วางบนสิ่ง (บล็อกหรือโต๊ะ) ที่มันควรอยู่ และลบหนึ่งถ้าไม่ใช่



รูปที่ 2–8 ตัวอย่างฟังก์ชันฮิวริสติก h1

2. เทคนิคการค้นหา

Ex. แสดงค่าฮิวริสติกสำหรับสถานะต่าง ๆ

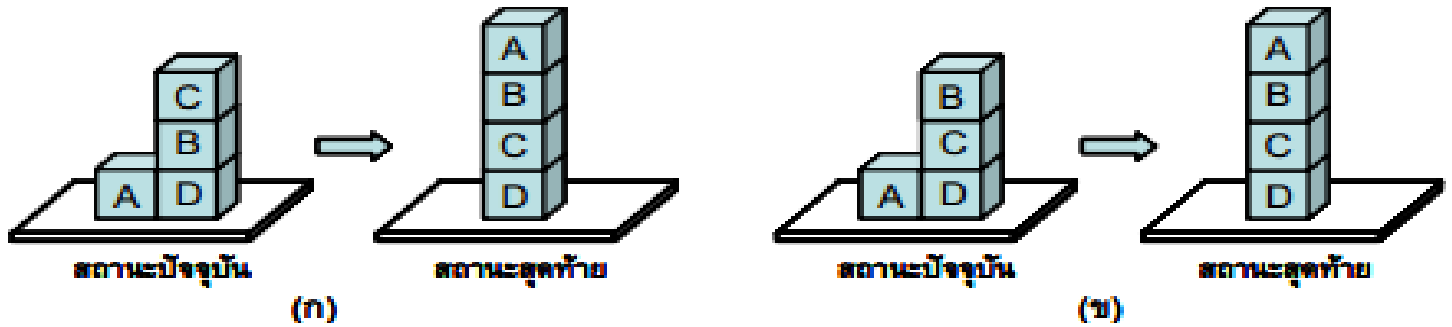


รูปที่ 2–9 ค่าฮิวริสติก h1 สำหรับสถานะต่างๆ ในรูปที่ 2–5

2. เทคนิคการค้นหา

Ex. ฟังก์ชันฮิวริสติก h2 สำหรับปัญหาโลกของบล็อก

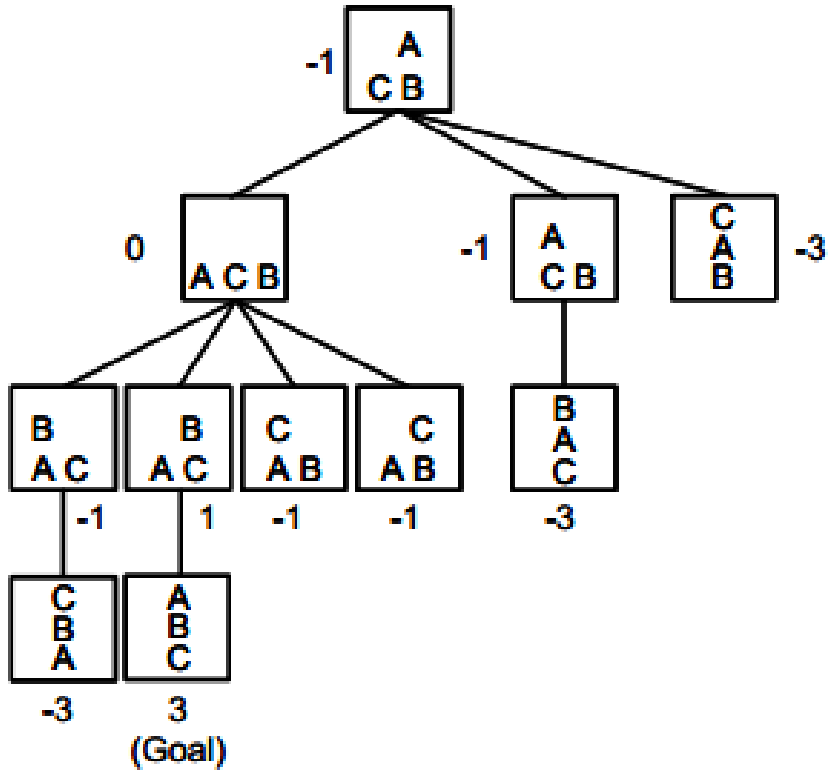
h2: สำหรับบล็อกแต่ละก้อนที่อยู่บนโครงสร้างที่ถูก บวก 1 แต้มให้กับบล็อกทุกก้อนที่อยู่ในโครงสร้างนั้นเพื่อเป็นคะแนนสำหรับบล็อกที่อยู่บนโครงสร้างที่ถูกนั้น และลบ 1 แต้มสำหรับบล็อกทุกก้อนที่อยู่บนโครงสร้างที่ผิด เพื่อเป็นคะแนนสำหรับบล็อกที่อยู่บนโครงสร้างที่ผิดนั้น ค่าฮิวริสติกสำหรับสถานะคือคะแนนรวมของบล็อกทุกก้อนที่พิจารณาโดยที่โครงสร้างคือบล็อกที่เรียงตัวต่อกันตามแนวตั้ง(ไม่นับโต๊ะ)



รูปที่ 2–10 ตัวอย่างฟังก์ชันฮิวริสติก h2

2. เทคนิคการค้นหา

Ex. แสดงค่าฮิวริสติกสำหรับสถานะต่าง ๆ



รูปที่ 2–11 ค่าฮิวริสติก h2 สำหรับสถานะต่างๆ ในรูปที่ 2–5

2. เทคนิคการค้นหา

Ex. ฟังก์ชันฮิวริสติกสำหรับปัญหา 8-Puzzle

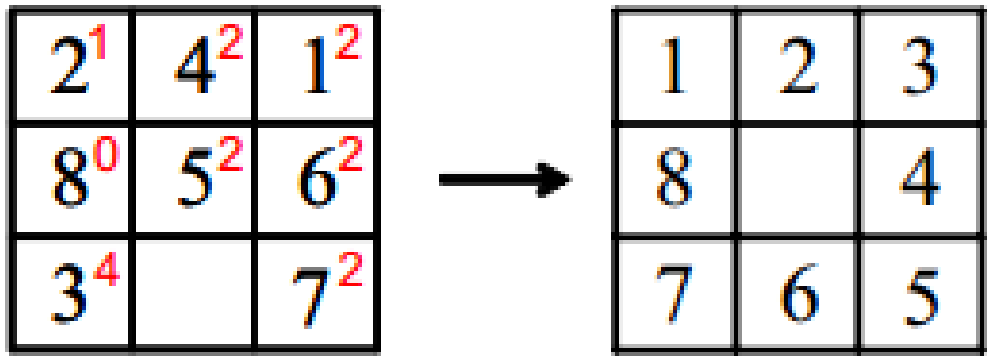
ตัวอย่างของฟังก์ชันฮิวริสติกสำหรับปัญหา 8-Puzzle แสดงในสมการด้านล่างนี้

$$h_{Man} = \sum_{i=1}^{8} d_x(c_i, g_i) + \sum_{i=1}^{8} d_y(c_i, g_i) \qquad (2.1)$$

โดยที่ c_i, g_i, d_x, d_y คือพิกัดของแผ่นป้าย i ที่สถานะปัจจุบัน พิกัดของแผ่นป้าย i ที่สถานะเป้าหมาย ระยะห่างระหว่าง c_i กับ g_i ตามแกน x และระยะห่างระหว่าง c_i กับ g_i ตามแกน y ตามลำดับ

2. เทคนิคการค้นหา

Ex. ตัวเลขที่อยู่มุมขวาบนแผ่นป้ายแต่ละแผ่นแสดงจำนวนครั้งที่ต้องเลื่อนไปยังตำแหน่งที่มันควรอยู่เมื่อเทียบกับคำตอบ

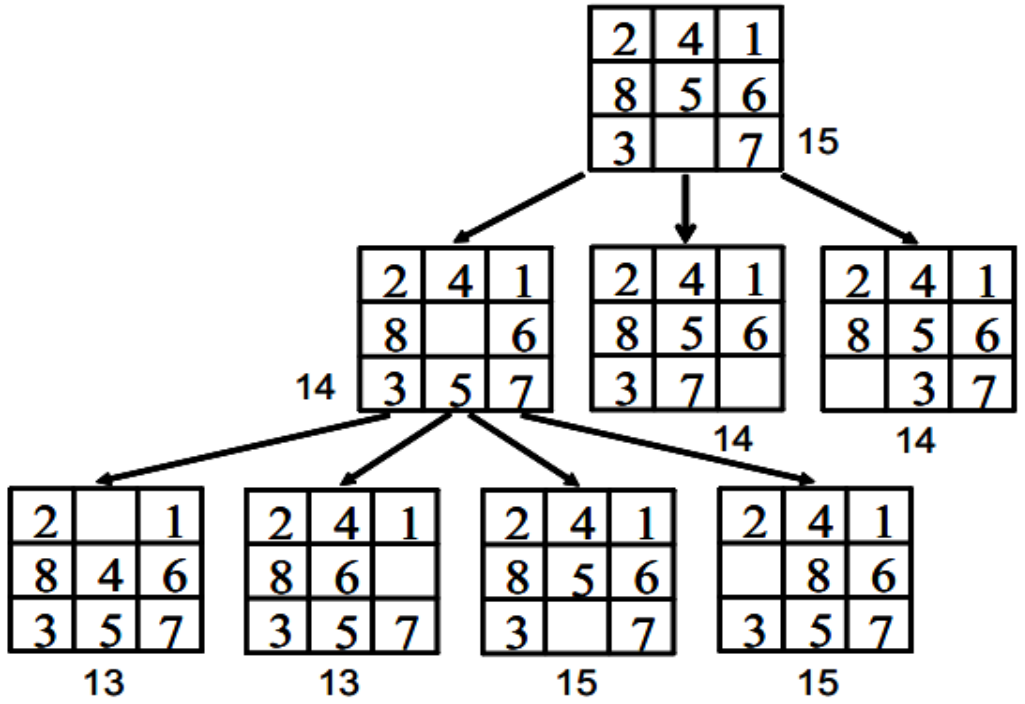


รูปที่ 2–12 ฟังก์ชันแมนฮัตตันสำหรับปัญหา 8-Puzzle

ดังนั้น ค่าฮิวริสติกของสถานะปัจจุบัน = 2+1+4+2+2+2+2+0 = 15 หน่วย

2. เทคนิคการค้นหา

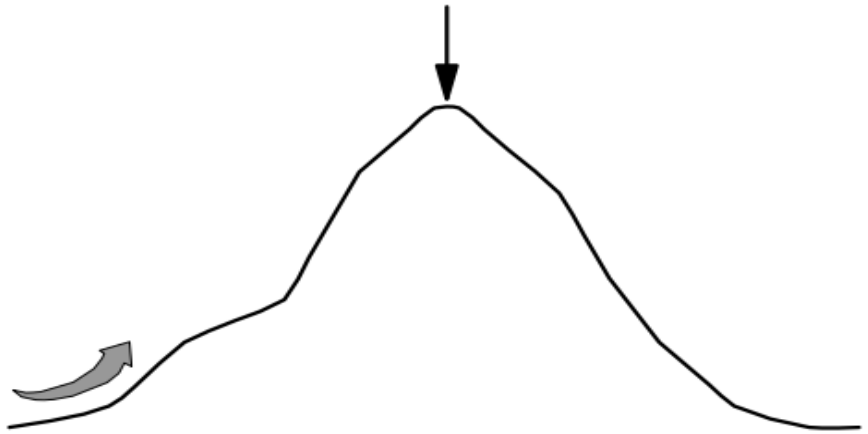
Ex. เมื่อนำฟังก์ชัน h_{Man} ไปวัดค่าฮิวริสติกให้กับสถานะต่างๆ ในรูป 2-2 จะได้ผลดังรูปที่ 2-13 ต่อไปนี้



รูปที่ 2–13 ค่าฟังก์ชันแมนฮัตตันสำหรับสถานะต่างๆ ในรูปที่ 2–2

2. เทคนิคการค้นหา

3. อัลกอริทึมปีนเขา (Hill-climbing Algorithm)



รูปที่ 2–14 อัลกอริทึมปีนเขา

การค้นหาแบบนี้เปรียบเสมือนการปีนไปสู่ยอดเขาดังแสดงในรูปที่ 2–14 ซึ่งอัลกอริทึมจะขึ้นในแนวดิ่งตลอด ถ้าเจอทางแยกเราก็จะไปแยกที่ตรงดิ่งขึ้นไปจนกระทั่งถึงยอดเขา ความสูงจากฐานภูเขาจนถึงตำแหน่งที่อยู่ในปัจจุบันก็จะเปรียบเหมือนค่าฮิวริสติก (ในกรณีนี้เราต้องการหาค่าสูงสุด) ซึ่งในที่นี้ยิ่งมากยิ่งดี อัลกอริทึมแสดงในตารางที่ 2–4

2. เทคนิคการค้นหา

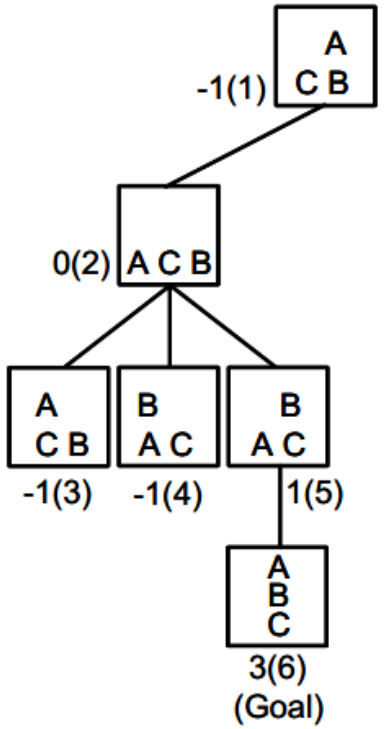
ตารางที่ 2-4 อัลกอริทึมแบบปีนเขาอย่างง่าย

```
Algorithm: Simple Hill-Climbing Search

1. Evaluate the initial state.
2. IF the initial state=goal state THEN
       return the initial state and quit
   ELSE current state := initial state.
3. UNTIL a goal state is found or there are no new
   operators left to be applied in the current state DO
     3.1  Select an operator that has not yet been
          applied to the current state and apply it to
          produce a new state.
     3.2  Evaluate the new state.
          IF  new state=goal state THEN
              return the new state and quit
          ELSE IF the new state is better than the
                  current state THEN
                  current state := new state
          ELSE IF the new state is not better than the
                  current state THEN
                  continue in this loop.
```

2. เทคนิคการค้นหา

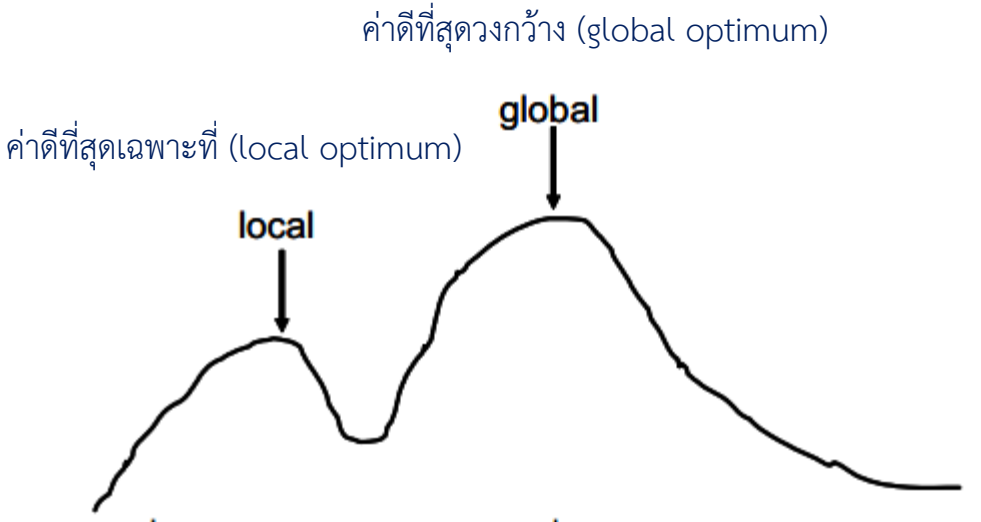
Ex. การใช้ฟังก์ชันฮิวริสติก h2 แสดงอัลกอริทึมแบบปีนเขาอย่างง่าย



รูปที่ 2–15 ตัวอย่างอัลกอริทึมปีนเขากับปัญหาโลกของบล็อก

2. เทคนิคการค้นหา

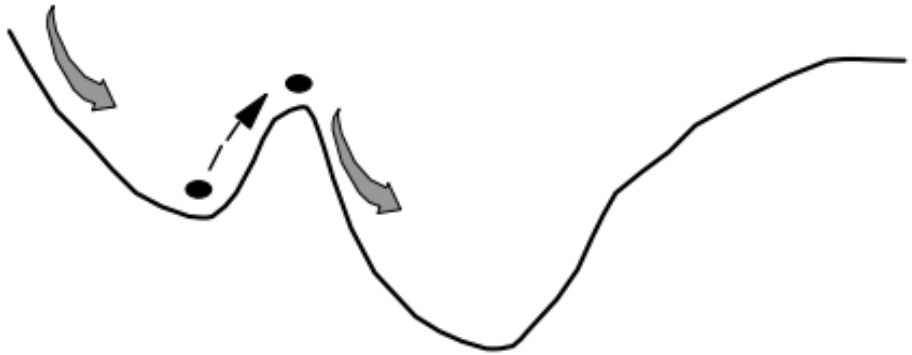
ปัญหาของอัลกอริทึมปีนเขา (Hill-climbing Algorithm)



รูปที่ 2–16 ตัวอย่างปัญหาที่เกิดกับอัลกอริทึมปีนเขา

2. เทคนิคการค้นหา

4. อัลกอริทึมการอบเหนียวจำลอง (Simulated Annealing Algorithm) ถูกออกแบบมาเพื่อให้สามารถหลุดออกได้จากสถานะดีที่สุดเฉพาะที่ โดยใช้แนวคิดของอุณหพลศาสตร์ของกระบวนการอบเหนียว ซึ่งเป็นการลดอุณหภูมิลงอย่างช้าๆ ระหว่างการหลอม เพื่อให้ได้โลหะที่อยู่ในสภาวะเหมาะสมที่สุด เป็นโลหะเหนียว ไม่เปราะ



รูปที่ 2–17 อัลกอริทึมการอบเหนียวจำลอง

2. เทคนิคการค้นหา

ตารางที่ 2-5 อัลกอริทึมแบบอบเหนียวจำลอง

```
Algorithm: Simulated Annealing Search

1. Evaluate the initial state.
2. IF initial state=goal state THEN
      return the initial state and quit
   ELSE current state := initial state.
3. BEST-SO-FAR := current state
4. T := constant
5. UNTIL a goal state is found or there are no new
   operators left to be applied in the current state DO
   5.1 Select an operator that has not yet been applied
       to the current state and apply it to produce a
       new state.
   5.2 Evaluate the new state.
       IF new state=goal state THEN
          return new state and quit
       ELSE IF the new state is better than the current
               state THEN {
               current state := new state
               IF the new state is better than BEST-SO-FAR
               THEN BEST-SO-FAR := new state }
       ELSE IF the new state is not better than the
               current state THEN {
               ΔE := |(value of the current state)-
                      (value of the new state)|
               IF e^(-ΔE/T) > random(0,1) THEN
                  current state := new state }
   5.3 Revise T as necessary.
6. Return BEST-SO-FAR as the answer.
```

2. เทคนิคการค้นหา

4. อัลกอริทึมดีที่สุดก่อน (Best-First Search) จะเก็บสถานะทุกตัวโดยไม่ตัดทิ้งไป โดยการที่ไม่ตัดทิ้งไปจึงทำให้อัลกอริทึมนี้ไม่พลาดเส้นทางที่นำไปสู่คำตอบ โดยแต่ละขั้นตอนจะเลือกสถานะที่มีค่าฮิวริสติกดีสุด โดยพิจารณาทุกตัวที่ยังไม่ถูกกระจาย (สถานะที่ยังไม่ได้สร้างสถานะลูก)

2. เทคนิคการค้นหา

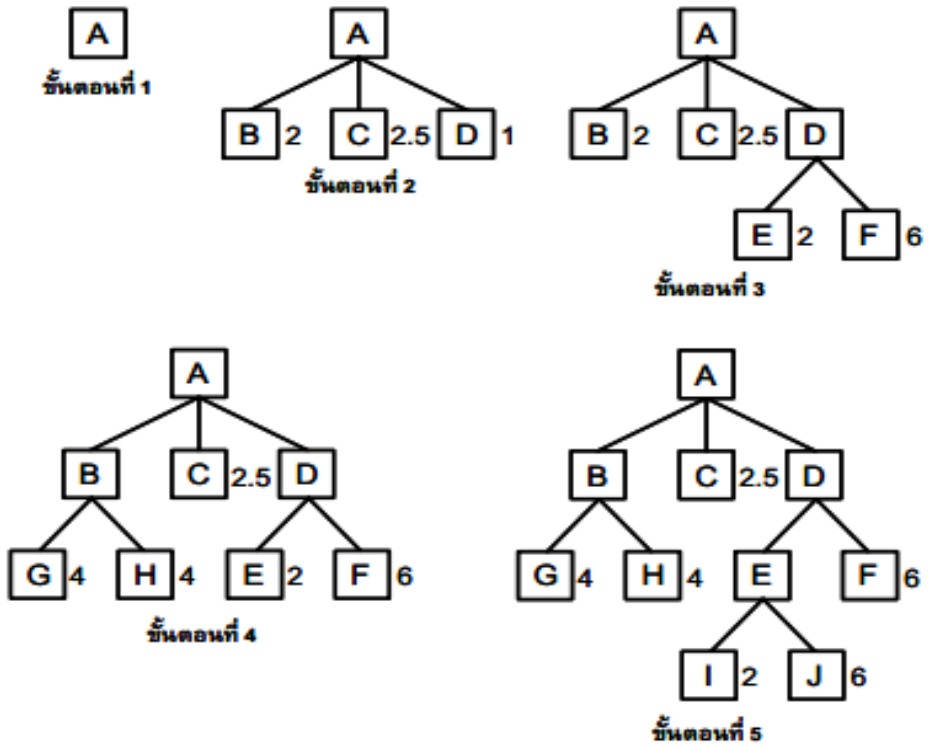
ตารางที่ 2-5 อัลกอริทึมดีที่สุดก่อน (Best-First Search)

```
Algorithm: Best-First Search

1. OPEN := {initial state}
2. UNTIL a goal state is found or there are no states
   left on OPEN DO
   2.1 Pick the best node on OPEN.
   2.2 Generate its successors.
   2.3 FOR EACH successor DO
          IF the successor has not been generated THEN
             evaluate it, add it to OPEN, and record it
             parent.
          IF the successor has been generated before
          THEN change the parent if this new path is
               better than the previous one.
```

2. เทคนิคการค้นหา

Ex. การค้นหาแบบอัลกอริทึมดีที่สุดก่อน (Best-First Search)



รูปที่ 2–18 อัลกอริทึมดีสุดก่อน

2. เทคนิคการค้นหา

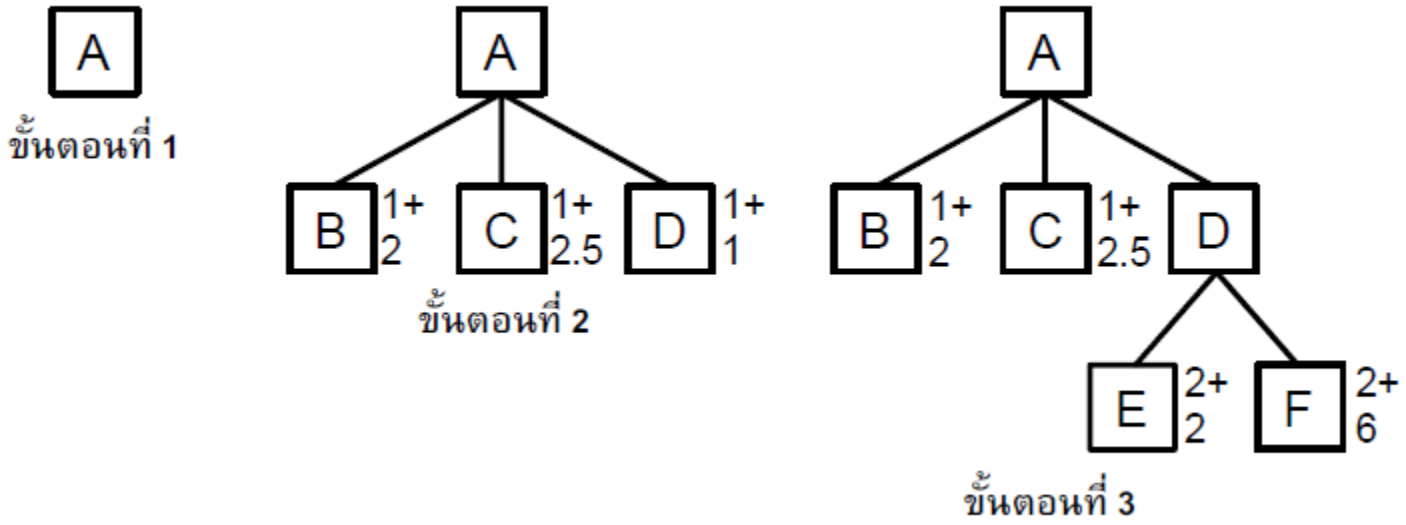
5. อัลกอริทึม A* เป็นการขยายอัลกอริทึมดีที่สุดก่อนโดยพิจารณาเพิ่มถึงต้นทุนจากสถานะเริ่มต้นมายังสถานะปัจจุบันเพื่อใช้คำนวณค่าฮิวริสติกด้วย ในกรณีของอัลกอริทึม A* ต้องหาค่าต่ำสุดของฟังก์ชัน f’(x) ของสถานะ s นิยามดังนี้

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

โดยที่ g คือฟังก์ชันที่คำนวณต้นทุนจากสถานะเริ่มต้นมายังสถานะปัจจุบัน h' คือฟังก์ชันที่ประมาณต้นทุนจากสถานะปัจจุบันไปยังคำตอบ ดังนั้น f' จึงเป็นฟังก์ชันที่ประมาณต้นทุนจากสถานะเริ่มต้นไปยังคำตอบ (ยิ่งน้อยยิ่งดี)

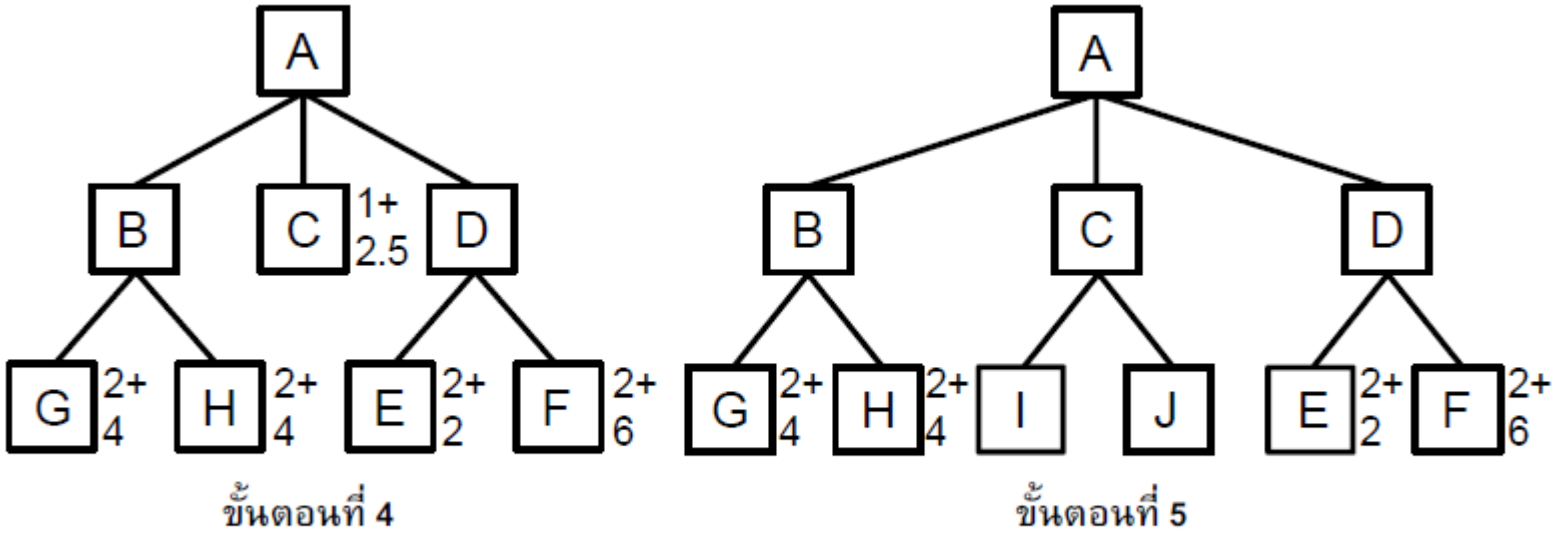
2. เทคนิคการค้นหา

Ex. แสดงการค้นหาด้วยอัลกอริทึมที่ดีที่สุดก่อน



2. เทคนิคการค้นหา

Ex. แสดงการค้นหาด้วยอัลกอริทึม A*



รูปที่ 2–19 อัลกอริทึม A*

2. เทคนิคการค้นหา

6. การค้นหาตาบู ตาบู (tabu, ta boo) แปลว่า ต้องห้าม (เช่น สิ่งของที่เป็นของศักดิ์สิทธิ์ จึงห้ามแตะต้อง) โดยจะทำเครื่องหมายบางเส้นทางที่ไม่สนใจจะค้นหา การทำเครื่องหมายนี้อาจทำในระดับของตัวกระทำการ หรือ หน่วยย่อยของตัวกระทำการก็ได้ หน่วยย่อยใดถูกทำเครื่องหมายไว้จะเปลี่ยนเป็นสถานะภาพต้องห้าม (tabu status) ให้อยู่ในภาวะต้องห้าม (tabu active) กล่าวคือ หน่วยย่อยนี้จะไม่ถูกนำมาใช้เพื่อสร้างเส้นทางในการค้นหา อาจเป็นเพราะเส้นทางนี้คงนำไปสู่คำตอบหรืออาจเป็นเส้นทางที่เคยค้นหามาแล้ว เป็นต้น

2. เทคนิคการค้นหา

แนวคิดการค้นหาที่ฉลาดจะต้องพิจารณาถึงสิ่งเหล่านี้ คือ

1. หน่วยความจำปรับตัว (adaptive memory) ตามสถานะภาพการค้นหา ณ ตำแหน่งนั้นๆ

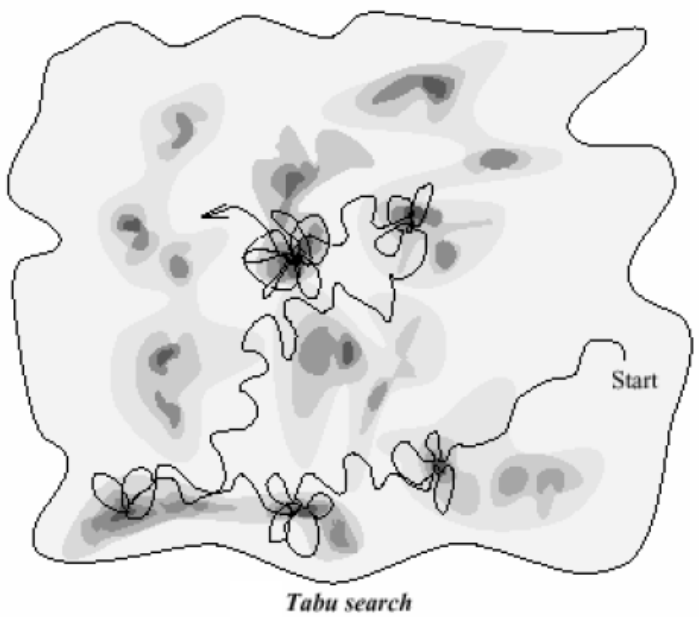
2. การสำรวจแบบสอบถาม (responsive exploration)

    2.1 หน่วยความจำตามเวลา (recency-based memory) จะปรับเปลี่ยนไปตามขั้นตอนการค้าหา

    2.2 หน่วยความจำตามความถี่ (frequency-based memory) ใช้สำหรับจำว่าตัวกระทำการตัวไหนใช้งานบ่อย ๆ

2. เทคนิคการค้นหา

Ex. แสดงการค้นหาแบบตาบู

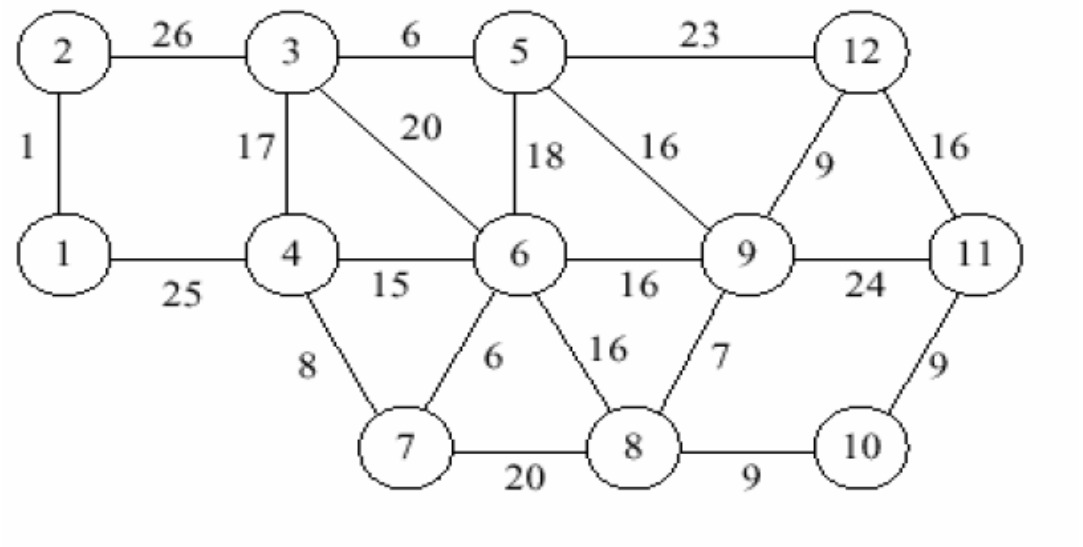


รูปที่ 2–20 การค้นหาตาบู

2. เทคนิคการค้นหา

Ex. ปัญหาแบบต้นไม้ k กิ่งน้อยที่สุด

ปัญหาต้นไม้ k กิ่งน้อยสุด (minimum k-tree problem) คือ การหาต้นไม้ที่มี k กิ่งจากกราฟโดยให้ผลรวมของน้ำหนักของกิ่งน้อยที่สุด (ในที่นี้ให้ k=4)



รูปที่ 2–21 ตัวอย่างปัญหาต้นไม้ k กิ่งน้อยสุด

2. เทคนิคการค้นหา

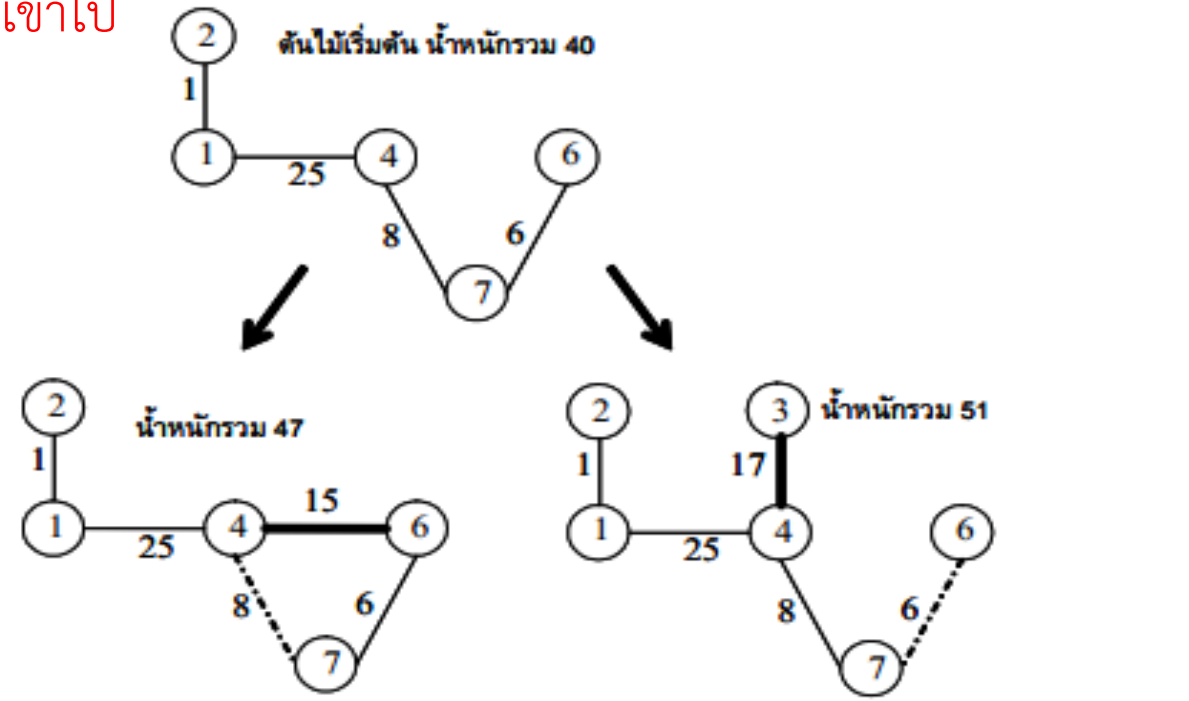
จากกราฟข้างบน เราจะสร้างต้นไม้ที่มี 4 กิ่งโดยใช้*อัลกอริทึมตะกราม (greedy algorithm)* ซึ่งเริ่มจากการหากิ่งแรกในกราฟที่มีน้ำหนักน้อยสุด (กิ่ง (1,2)) จากนั้นหากิ่งที่เชื่อมกับกิ่งนี้ที่มีน้ำหนักน้อยสุด ทำเช่นนี้ไปจนครบ 4 กิ่ง จะได้ผลดังตารางข้างล่างนี้

ตารางที่ 2–7 การสร้างผลเฉลยเริ่มต้นด้วยอัลกอริทึมตะกราม

ขั้นตอนที่	ตัวเลือก	กิ่งที่เลือก	น้ำหนักรวม
1	(1,2)	(1,2)	1
2	(1,4), (2,3)	(1,4)	26
3	(2,3), (3,4), (4,6), (4,7)	(4,7)	34
4	(2,3), (3,4), (4,6), (6,7),(7,8)	(6,7)	40

2. เทคนิคการค้นหา

Ex. แสดงการสร้างผลเฉลยใหม่ 2 ตัวโดยการลบกิ่ง 1 กิ่ง (แสดงด้วยเส้นปะในรูป) ออกจากต้นไม้เดิมและเพิ่มกิ่งอีก 1 กิ่ง (แสดงด้วยเส้นทึบในรูป) เข้าไป



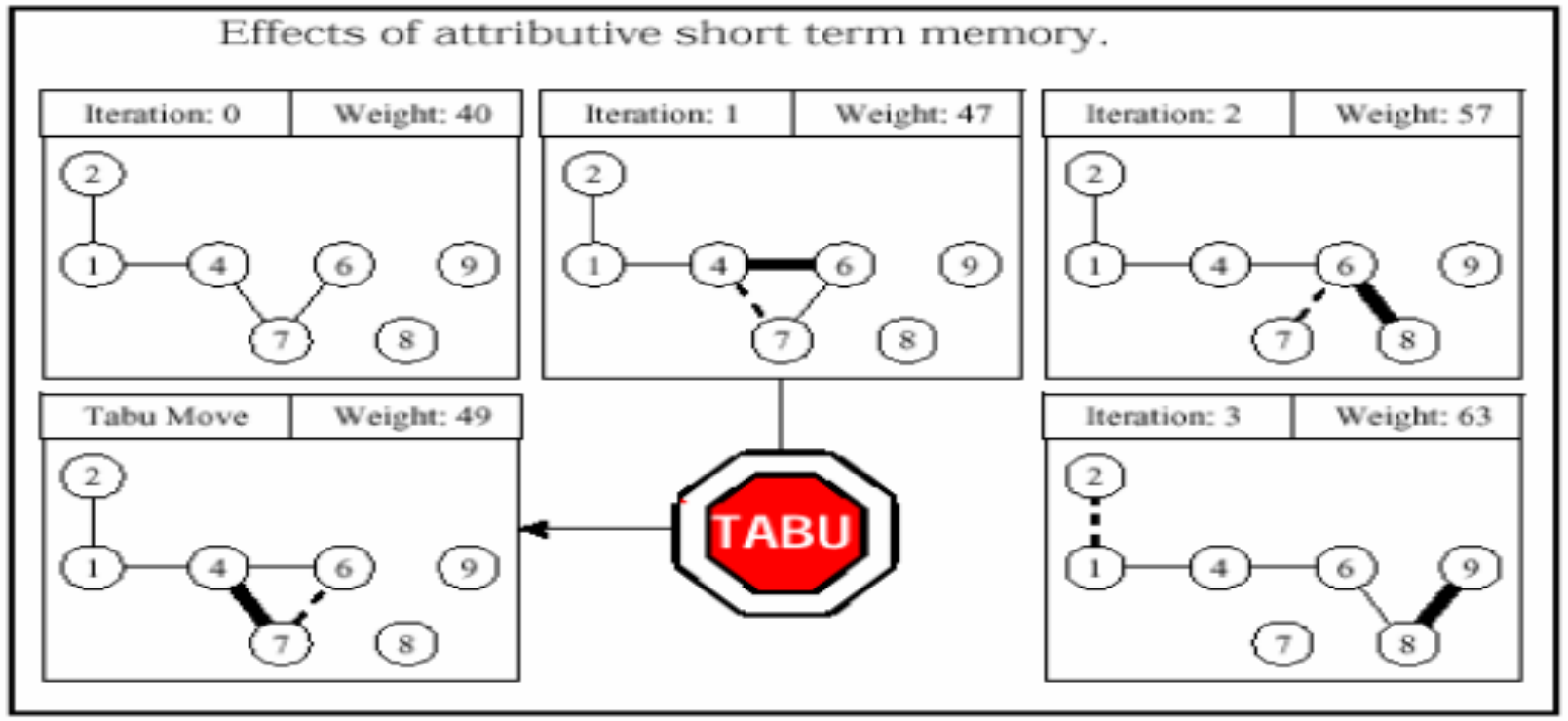
รูปที่ 2–22 การสร้างผลเฉลยข้างเคียงโดยลบกิ่งหนึ่งกิ่งและเพิ่มกิ่งหนึ่งกิ่ง

2. เทคนิคการค้นหา

ตารางที่ 2–8 ผลเฉลยที่ได้เมื่อผ่านไป 3 รอบ

รอบที่	ระยะเวลาต้องห้าม		กิ่งเพิ่มเข้า	กิ่งลบออก	น้ำหนัก
	1	2			
1			(4,6)	(4,7)	47
2	(4,6)	(4,7)	(6,8)	(6,7)	57
3	(6,8), (4,7)	(6,7)	(8,9)	(1,2)	63

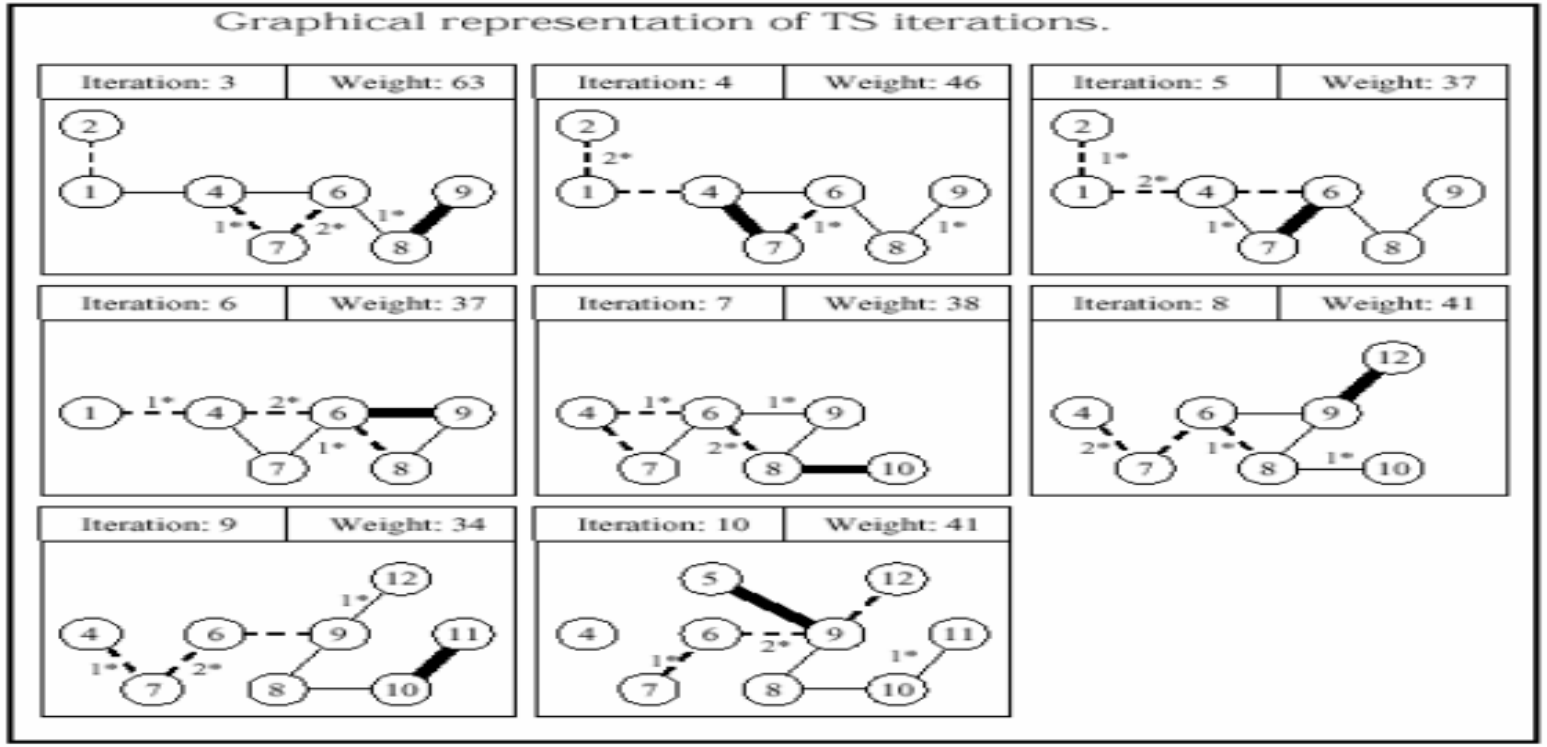
2. เทคนิคการค้นหา



รูปที่ 2–23 ผลของการใช้หน่วยความจำระยะสั้น

2. เทคนิคการค้นหา

การค้นหาตาบูจะทำการค้นหาไปจนกระทั่งจำนวนรอบเกินกว่าค่าขีดแบ่งที่กำหนดไว้ (ในตัวอย่างด้านบนกำหนดให้เป็น 10 รอบ) กระบวนการค้นหาแสดงในรูปที่ 2–24 และ ตารางที่ 2–9 ซึ่งจะพบว่าการใช้ภาวะต้องห้ามจะทำให้ประหยัดเวลาในการไปในเส้นทางต่าง ๆ ได้มากพอควร ตัวเลข 1* และ 2* ในรูปที่ 2–24 แสดงระยะเวลาต้องห้ามของกิ่ง



รูปที่ 2–24 ต้นไม้ที่ได้ในรอบที่ 3 ถึงรอบที่ 10 ของการค้นหาตาบู

2. เทคนิคการค้นหา

ตารางที่ 2–9 ผลเฉลยที่ได้เมื่อผ่านไป 10 รอบ

รอบที่	ระยะเวลาต้องห้าม		กิ่งเพิ่มเข้า	กิ่งลบออก	น้ำหนัก
	1	2			
0					40
1			(4,6)	(4,7)	47
2	(4,6)	(4,7)	(6,8)	(6,7)	57
3	(6,8), (4,7)	(6,7)	(8,9)	(1,2)	63
4	(6,7), (8,9)	(1,2)	(4,7)	(1,4)	46
5	(1,2), (4,7)	(1,4)	(6,7)	(4,6)	37
6	(1,4), (6,7)	(4,6)	(6,9)	(6,8)	37
7	(4,6), (6,9)	(6,8)	(8,10)	(4,7)	38
8	(6,8), (8,10)	(4,7)	(9,12)	(6,7)	41
9	(4,7), (9,12)	(6,7)	(10,11)	(6,9)	34
10	(6,7), (10,11)	(6,9)	(5,9)	(9,2)	41

2. เทคนิคการค้นหา

หน่วยความจำระยะยาว

หน่วยความจำเหตุการณ์วิกฤต (critical event memory) เพื่อจดจำเหตุการณ์สำคัญที่ผ่านมาแล้วนำมาใช้เป็นข้อมูลสำหรับการสร้างสถานภาพต้องห้ามสำหรับจุดใหม่ที่จะใช้เป็นจุดเริ่มต้นของการค้นหาครั้งใหม่ และนอกจากนั้นหน่วยความจำเหตุการณ์วิกฤตนี้จะใช้กำหนดความหลากหลายอีกด้วย

สำหรับปัญหานี้กำหนดให้เหตุการณ์สำคัญคือ

- จุดเริ่มต้นของการค้นหาแต่ละครั้ง (รอบที่ 0 ในกรณีตัวอย่าง)
- จุดให้ค่าต่ำสุดเฉพาะที่ซึ่งเกิดขึ้นในการค้นหาแต่ละครั้งที่ให้ค่า f(x) น้อยกว่าหรือเท่ากับจุดก่อนหน้าและจุดด้านหลัง (รอบที่ 5,6,9 ในกรณีของตัวอย่าง)
- ในตัวอย่างรอบที่ 9 คือรอบที่ให้ค่าต่ำสุด ดังนั้นเราต้องการเริ่มการค้นหาครั้งใหม่ก่อนรอบนี้โดยไม่นำรอบที่ 9 นี้มาพิจารณา

2. เทคนิคการค้นหา

ในกรณีนี้เหตุการณ์สำคัญคือเหตุการณ์ที่เกิดในรอบที่ 0, 5, 6 เราจะรวบรวมข้อมูลทั้งหมดของทั้งสามรอบไว้ในหน่วยความจำระยะยาว ซึ่งก็คือกิ่ง (1,2), (1,4), (4,7), (6,7), (6,8), (8,9) และ (6,9) และในกรณีที่ใช้หน่วยความจำตามความถี่ (frequency-based memory) เป็นหน่วยความจำระยะยาว เราจะใช้จำนวนครั้งเพื่อกำหนดความสำคัญของแต่ละกิ่งด้วย ในกรณีนี้สมมติว่าไม่นำความถี่มาพิจารณาจะได้ดังนี้

รอบที่ 0 ประกอบด้วย (1,2), (1,4), (4,7), (6,7)

รอบที่ 5 ประกอบด้วย (4,7), (6,7), (8,9), (6,8)

รอบที่ 6 ประกอบด้วย (4,7), (6,7), (8,9), (6,9)

จากนั้นจะให้กิ่งเหล่านี้มีสถานภาพเป็นภาวะต้องห้าม เพื่อใช้เป็นตัวป้องกันการสร้างจุดเริ่มต้นใหม่ที่มีกิ่งเหมือนกับกิ่งในหน่วยความจำนี้ อย่างไรก็ดีในแต่ละขั้นตอนของการสร้างจุดเริ่มต้นใหม่นั้น (ในกรณีของตัวอย่างเราใช้อัลกอริทึมตะกราม ซึ่งจะทำทั้งหมด 4 ขั้นตอน – มี 4 กิ่ง) จะทำการป้องกันมากถ้าเป็นขั้นตอนต้นๆ และป้องกันน้อยถ้าเป็นขั้นตอนท้ายๆ ของการสร้างจุดเริ่มต้นใหม่ (เพราะหากป้องกันมากไปอาจทำให้ไม่สามารถสร้างจุดใหม่ได้เลย)

2. เทคนิคการค้นหา

ในกรณีของตัวอย่าง กำหนดให้การป้องกันเป็นดังต่อไปนี้

- ใน 2 ขั้นตอนแรก ห้ามมีกิ่งที่เราเก็บไว้ในหน่วยความจำเลย (คือห้ามมีกิ่งที่เราจำไว้ข้างบนของรอบ 0, 5, 6 ซึ่งก็คือกิ่ง (1,2), (1,4), (4,7), (6,7), (6,8), (8,9) และ (6,9) ดังนั้นเราจะเริ่มจาก (3,5) เป็นกิ่งแรก (ดูตารางที่ 2–10 ประกอบ)
- หลังจากนั้นยอมให้มีกิ่งในหน่วยความจำได้

ตารางที่ 2–10 กระบวนการหาจุดเริ่มต้นใหม่โดยใช้หน่วยความจำระยะยาว

ขั้นตอนที่	ตัวเลือก	กิ่งที่เลือก	น้ำหนักรวม
1	(3,5)	(3,5)	6
2	(2,3), (3,4), (3,6), (5,6), (5,9), (5,12)	(5,9)	22
3	(2,3), (3,4), (3,6), (5,6), (5,12), (6,9), (8,9), (9,12)	(8,9)	29
4	(2,3), (3,4), (3,6), (5,6), (5,12), (6,8), (6,9), (7,8), (8,10), (9,12)	(8,10)	38

ในขั้นตอนที่ 2 เลือก (5,9) ก็เพราะว่ามีน้ำหนักน้อยที่สุดในทางเลือกทั้งหมดที่เป็นไปได้ ส่วนขั้นตอนที่ 3 และ 4 ก็เช่นเดียวกัน เราจะใช้ต้นไม้ที่ได้ (ในรูปที่ 2–25) เป็นจุดเริ่มต้น

2. เทคนิคการค้นหา

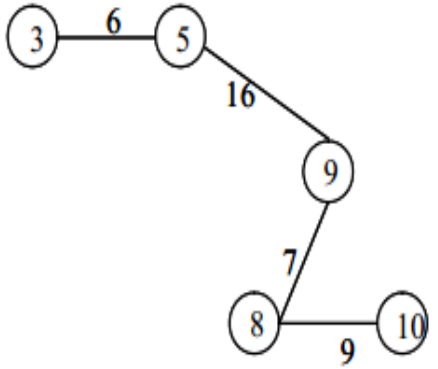
ในกรณีของตัวอย่าง กำหนดให้การป้องกันเป็นดังต่อไปนี้

- ใน 2 ขั้นตอนแรก ห้ามมีกิ่งที่เราเก็บไว้ในหน่วยความจำเลย (คือห้ามมีกิ่งที่เราจำไว้ข้างบนของรอบ 0, 5, 6 ซึ่งก็คือกิ่ง (1,2), (1,4), (4,7), (6,7), (6,8), (8,9) และ (6,9) ดังนั้นเราจะเริ่มจาก (3,5) เป็นกิ่งแรก (ดูตารางที่ 2–10 ประกอบ)
- หลังจากนั้นยอมให้มีกิ่งในหน่วยความจำได้

ตารางที่ 2–10 กระบวนการหาจุดเริ่มต้นใหม่โดยใช้หน่วยความจำระยะยาว

ขั้นตอนที่	ตัวเลือก	กิ่งที่เลือก	น้ำหนักรวม
1	(3,5)	(3,5)	6
2	(2,3), (3,4), (3,6), (5,6), (5,9), (5,12)	(5,9)	22
3	(2,3), (3,4), (3,6), (5,6), (5,12), (6,9), (8,9), (9,12)	(8,9)	29
4	(2,3), (3,4), (3,6), (5,6), (5,12), (6,8), (6,9), (7,8), (8,10), (9,12)	(8,10)	38

ในขั้นตอนที่ 2 เลือก (5,9) ก็เพราะว่ามีน้ำหนักน้อยที่สุดในทางเลือกทั้งหมดที่เป็นไปได้ ส่วนขั้นตอนที่ 3 และ 4 ก็เช่นเดียวกัน เราจะใช้ต้นไม้ที่ได้ (ในรูปที่ 2–25) เป็นจุดเริ่มต้นใหม่ และใช้กระบวนการค้นหาด้วยหน่วยความจำระยะสั้นเช่นเดียวกับที่ผ่ามมา ซึ่งจะได้ผลดังตารางที่ 2–11 และรูปที่ 2–26



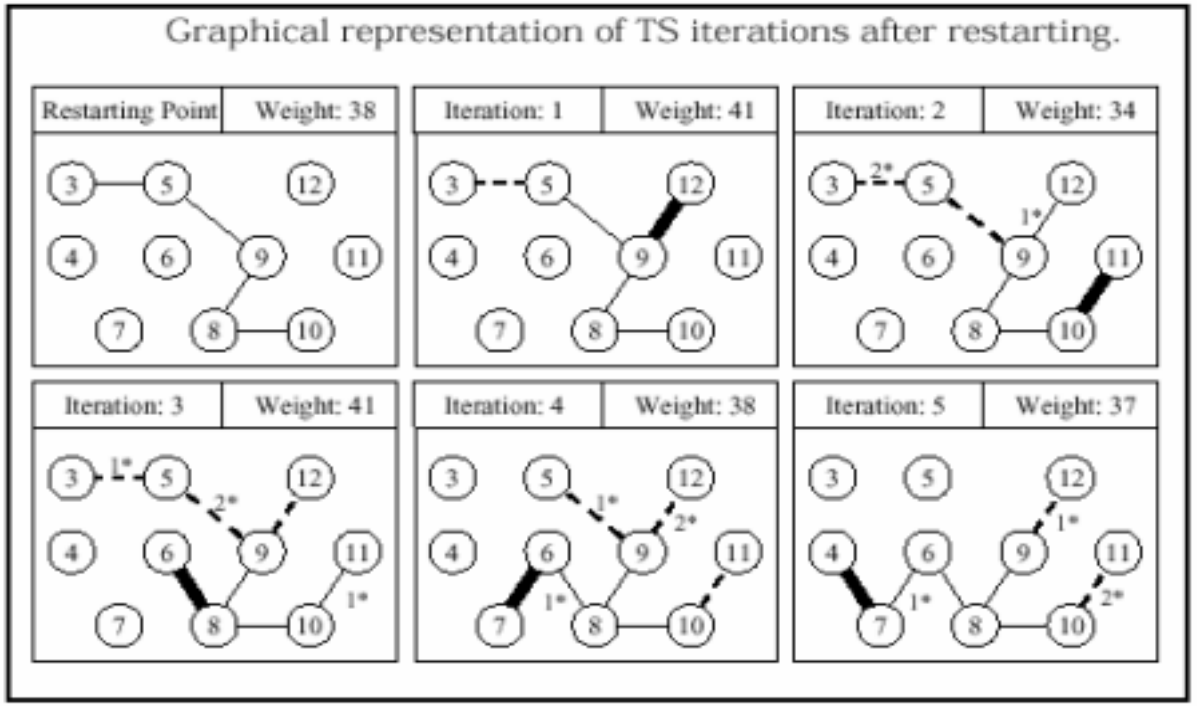
รูปที่ 2–25 ต้นไม้ที่ได้สำหรับเริ่มการค้นหาใหม่

2. เทคนิคการค้นหา

ตารางที่ 2–11 กระบวนการค้นหาตาบูเมื่อเริ่มจากผลเฉลยตัวใหม่

รอบที่	ระยะเวลาต้องห้าม		กิ่งเพิ่มเข้า	กิ่งลบออก	ค่าที่เปลี่ยน	น้ำหนัก
	1	2				
1			(9,12)	(3,5)	3	41
2	(9,12)	(3,5)	(10,11)	(5,9)	-7	34
3	(3,5), (10,11)	(5,9)	(6,8)	(9,12)	7	41
4	(5,9), (6,8)	(9,12)	(6,7)	(10,11)	-3	38
5	(9,12), (6,7)	(10,11)	(4,7)	(8,10)	-1	37

2. เทคนิคการค้นหา



รูปที่ 2–26 ต้นไม้ที่ได้ในแต่ละรอบของกระบวนการค้นหาตาบูเมื่อเริ่มจากผลเฉลยตัวใหม่

2. เทคนิคการค้นหา

ตารางที่ 2–12 อัลกอริทึมการค้นหาตาบู

```
Algorithm: Tabu Search

1. Choose an initial (possibly random) solution x ∈ X
2. x* := x, k := 1
3. Initialize tabu short-term memory and long-term
   memory
4. WHILE the stopping condition is not met DO {
          k := k+1
          Generate a candidate set N*(x) including x'
           with tabu-active which satisfies aspiration
           criteria.
          Find a best x' ∈ N*(x)
          IF local optimum reached THEN {
             IF no improvement made over a period THEN {
                Apply long term memory to restart the
                  process, and find a new solution x'.
             }
          }
          x := x'
          Update tabu memory and adjust search parameters.
          IF f(x') < f(x*) THEN x* := x'
   }
```

THANK YOU
FOR
YOUR ATTENTION

จบบทที่ 2