ปัญญาประดิษฐ์ SCI193611

การบรรยายที่ 3: เกมและการค้นหาแบบประชันกัน (Games and Adversarial search)

อิทธิพล ฟองแก้ว
[ittipon@g.sut.ac.th]

Nash Equilibrium| HD | With Subtitles|

วันนี้

- จะทำอย่างไรเพื่อทำงานอย่างมีเหตุผลในสภาพแวดล้อม multi-agent?
- จะคาดการณ์และตอบสนองต่อ พฤติกรรมที่ไม่แน่นอน ของ agent อื่นๆ ได้อย่างไร?
- การค้นหาแบบประชันกัน (Adversarial search)
  - Minimax
  - $\alpha - \beta$ pruning
  - H-Minimax
  - Expectiminimax
  - Monte Carlo Tree Search
- สมมติฐานการสร้างแบบจำลอง
- Agent ที่ทันสมัย

Minimax

เกม (Games)

- เกม เป็นสภาพแวดล้อม multi-agent ที่ agent อาจมีผลประโยชน์ที่ ขัดแย้งกัน หรือ ร่วมกัน
- ฝ่ายตรงข้ามอาจทำงานได้ อย่างไรก็ได้ แม้ว่าเราจะสมมติสภาพแวดล้อมที่กำหนดได้และสังเกตได้อย่างสมบูรณ์
    - วิธีการแก้ปัญหาเกมคือ กลยุทธ์ ที่ระบุการเคลื่อนไหวสำหรับทุกการตอบสนองของฝ่ายตรงข้ามที่เป็นไปได้
    - นี่แตกต่างจากการค้นหาที่วิธีการแก้ปัญหาคือ ลำดับการกระทำที่คงที่
- เวลามักจะ จำกัด

ประเภทของเกม

- กำหนดได้ (Deterministic) หรือ สุ่ม (stochastic)?
- ข้อมูล สมบูรณ์ (Perfect) หรือ ไม่สมบูรณ์ (imperfect)?
- สองผู้เล่น หรือ หลายผู้เล่น?

คำนิยามแบบเป็นทางการ

เกม มีคำนิยามอย่างเป็นทางการเป็นปัญหาการค้นหาชนิดหนึ่งที่มีองค์ประกอบดังนี้:

- การแสดงของ สถานะ ของ agent และสภาพแวดล้อม
- สถานะเริ่มต้น s_0 ของเกม
- ฟังก์ชัน $\text{player}(s)$ ที่กำหนดว่า ผู้เล่น $p \in \{1, ..., N\}$ คนไหนเป็นผู้เล่นในสถานะ s
- คำอธิบายของ การกระทำ (หรือ การเคลื่อนไหว) ที่ถูกต้องที่มีในสถานะ s แสดงด้วย $\text{actions}(s)$
- โมเดลการเปลี่ยนแปลง ที่ส่งกลับสถานะ $s' = \text{result}(s, a)$ ที่เป็นผลมาจากการกระทำ a ในสถานะ s
- การทดสอบสถานะสิ้นสุด ที่กำหนดว่าเกมจบหรือไม่

- ฟังก์ชัน utility $\mathrm{utility}(s, p)$ (หรือ payoff) ที่กำหนดค่าตัวเลขสุดท้ายสำหรับเกมที่จบในสถานะ s สำหรับผู้เล่น p
  - เช่น $1, 0$ หรือ $\frac{1}{2}$ หากผลลัพธ์คือชนะ แพ้ หรือเสมอ
- เมื่อรวมกัน สถานะเริ่มต้น ฟังก์ชัน $\mathrm{actions}(s)$ และฟังก์ชัน $\mathrm{result}(s, a)$ จะกำหนด ต้นไม้เกม
  - โหนดคือสถานะของเกม
  - ขอบคือการกระทำ

เกม Zero-sum

- ในเกม zero-sum ผลตอบแทนรวมให้กับผู้เล่นทั้งหมดจะ คงที่ สำหรับทุกเกม
    - เช่น ในหมากรุก: $0 + 1, 1 + 0$ หรือ $\frac{1}{2} + \frac{1}{2}$
- สำหรับเกมสองผู้เล่น agent มี ฟังก์ชัน utility เดียวกัน แต่ฝ่ายหนึ่งต้องการ เพิ่ม ขณะที่อีกฝ่ายต้องการ ลด
    - MAX เพิ่มฟังก์ชัน utility ของเกม
    - MIN ลดฟังก์ชัน utility ของเกม
- การแข่งขันที่เข้มงวด
    - หากฝ่ายหนึ่งชนะ อีกฝ่ายก็แพ้ และในทางกลับกัน

—

Credits: CS188, UC Berkeley.

ต้นไม้เกม Tic-Tac-Toe

กลยุทธ์ที่เหมาะสมที่สุด (หรือการเล่นที่สมบูรณ์) คืออะไร? เราจะหาได้อย่างไร?

สมมติฐาน

- เราสมมติว่าเป็นเกม zero-sum สำหรับ สองผู้เล่น แบบ กำหนดได้ ผลัดกันเล่น ที่มีข้อมูล สมบูรณ์
  - เช่น Tic-Tac-Toe, หมากรุก, หมากฮอส, โกะ เป็นต้น
- เราจะเรียกผู้เล่นสองคนว่า MAX และ MIN โดย MAX เคลื่อนไหวก่อน

—

Credits: CS188, UC Berkeley.

การค้นหาแบบประชันกัน

- ในปัญหาการค้นหา วิธีแก้ปัญหาที่เหมาะสมที่สุดคือลำดับการกระทำที่นำไปสู่สถานะเป้าหมาย
  - กล่าวคือ สถานะสิ้นสุดที่ MAX ชนะ
- ในเกม ฝ่ายตรงข้าม (MIN) อาจตอบสนองต่อการเคลื่อนไหวได้ อย่างไรก็ได้
- ดังนั้น ผู้เล่น (MAX) จะต้องกำหนด กลยุทธ์ ที่เป็นเงื่อนไขซึ่งระบุ
  - การเคลื่อนไหวในสถานะเริ่มต้น
  - การเคลื่อนไหวในสถานะที่เป็นผลมาจากการตอบสนองทุกอย่างที่เป็นไปได้โดย MIN
  - การเคลื่อนไหวในสถานะที่เป็นผลมาจากการตอบสนองทุกอย่างที่เป็นไปได้โดย MIN ในสถานะเหล่านั้น...

—

Credits: CS188, UC Berkeley.

Minimax

ค่า minimax $\text{minimax}(s)$ เป็นผลตอบแทนที่ใหญ่ที่สุดที่สามารถบรรลุได้ (สำหรับ MAX) จากสถานะ s โดยสมมติว่ามีฝ่ายตรงข้ามที่เหมาะสม (MIN)

การเคลื่อนไหวครั้งต่อไปที่ เหมาะสม (สำหรับ MAX) คือการกระทำที่เพิ่มค่า minimax ในสถานะที่เป็นผลลัพธ์ให้มากที่สุด

- โดยสมมติว่า MIN เป็นฝ่ายตรงข้ามที่เหมาะสมที่เพิ่มผลลัพธ์ กรณีเลวร้ายที่สุด สำหรับ MAX ให้มากที่สุด
- สิ่งนี้เทียบเท่ากับการไม่ทำสมมติฐานเกี่ยวกับความแข็งแกร่งของฝ่ายตรงข้าม

คุณสมบัติของ Minimax

- ความสมบูรณ์ (Completeness):
  - ใช่ หากต้นไม้มีจำกัด
- ความเหมาะสม (Optimality):
  - ใช่ หาก MIN เป็นฝ่ายตรงข้ามที่เหมาะสม
  - จะเป็นอย่างไรหาก MIN ไม่เหมาะสม?
    - แสดงว่า MAX จะทำได้ดีกว่า
  - จะเป็นอย่างไรหาก MIN ไม่เหมาะสมและคาดการณ์ได้?
    - กลยุทธ์อื่นๆ อาจทำได้ดีกว่า Minimax แต่อาจทำได้แย่กว่าฝ่ายตรงข้ามที่เหมาะสม

ประสิทธิภาพของ Minimax

- สมมติว่า $\mathrm{minimax}(s)$ ถูกใช้งานโดยใช้คำนิยามแบบเวียนเกิด
- Minimax มี ประสิทธิภาพ อย่างไร?
    - ความซับซ้อนของเวลา: เหมือนกับ DFS กล่าวคือ $O(b^m)$
    - ความซับซ้อนของพื้นที่:
        - $O(bm)$ หากการกระทำทั้งหมดถูกสร้างขึ้นในครั้งเดียว หรือ
        - $O(m)$ หากการกระทำถูกสร้างขึ้นครั้งละหนึ่งรายการ

เราจำเป็นต้องสำรวจต้นไม้เกมทั้งหมดหรือไม่?

การตัดแต่ง (Pruning)

ดังนั้น จึงเป็นไปได้ที่จะคำนวณการตัดสินใจ minimax ที่ ถูกต้อง โดยไม่ต้องดูทุกโหนด ในต้นไม้

เราต้องการคำนวณ $v = \mathrm{minimax}(n)$ สำหรับ $\mathrm{player}(n)$=MIN

- เราวนลูปผ่านลูกของ n
- ค่า minimax กำลังถูกคำนวณทีละค่าและ v ได้รับการปรับปรุงแบบซ้ำ
- ให้ α เป็นค่าที่ดีที่สุด (กล่าวคือ สูงสุด) ที่จุดเลือกใดๆ ตามเส้นทางสำหรับ MAX
- หาก v กลายเป็นต่ำกว่า α แล้ว n จะไม่ถูกเข้าถึง ในการเล่นจริง
- ดังนั้น เราสามารถ หยุดการวนซ้ำ ผ่านลูกคนอื่นๆ ที่เหลือของ n

เช่นเดียวกัน β ถูกกำหนดให้เป็นค่าที่ดีที่สุด (กล่าวคือ ต่ำสุด) ที่จุดเลือกใดๆ ตามเส้นทางสำหรับ MIN เราสามารถหยุดการขยายของโหนด MAX ทันทีที่ v กลายเป็นใหญ่กว่า β

การตัดแต่ง α-β

- ปรับปรุงค่าของ α และ β เมื่อเส้นทางถูกขยาย
- ตัดแต่งกิ่งที่เหลือ (กล่าวคือ ยุติการเรียกแบบเวียนเกิด) ทันทีที่ค่าของโหนดปัจจุบันเป็นที่ทราบว่าแย่กว่าค่า α หรือ β ปัจจุบันสำหรับ MAX หรือ MIN ตามลำดับ

การค้นหา α-β

คุณสมบัติของการค้นหา α-β

- การตัดแต่ง ไม่มีผลกระทบ ต่อค่า minimax ดังนั้น ความสมบูรณ์ และ ความเหมาะสม จึงถูกรักษาไว้จาก Minimax
- ความซับซ้อนของเวลา:
  - ประสิทธิภาพขึ้นอยู่กับลำดับที่สถานะถูกตรวจสอบ
  - หากสถานะสามารถตรวจสอบใน ลำดับที่สมบูรณ์ แล้วการค้นหา $\alpha - \beta$ จะตรวจสอบเพียง $O(b^{m/2})$ โหนดเพื่อเลือกการเคลื่อนไหวที่ดีที่สุด เทียบกับ $O(b^m)$ สำหรับ minimax
    - $\alpha - \beta$ สามารถแก้ปัญหาต้นไม้ที่ลึกกว่าสองเท่าของ minimax ได้ในระยะเวลาเดียวกัน
    - เทียบเท่ากับ branching factor ที่มีประสิทธิภาพ $\sqrt{b}$
- ความซับซ้อนของพื้นที่: $O(m)$ เช่นเดียวกับ Minimax

ขนาดของต้นไม้เกม

หมากรุก:

- $b \approx 35$ (branching factor เฉลี่ยโดยประมาณ)
- $d \approx 100$ (ความลึกของต้นไม้เกมสำหรับเกมทั่วไป)
- $b^d \approx 35^{100} \approx 10^{154}$
- สำหรับการค้นหา $\alpha - \beta$ และการจัดลำดับที่สมบูรณ์ เราได้ $b^{d/2} \approx 35^{50} = 10^{77}$

การหาวิธีแก้ปัญหาที่แน่นอนด้วย Minimax ยังคง ไม่สามารถแก้ได้

ตาราง Transposition

- สถานะที่ซ้ำกันเกิดขึ้นบ่อยครั้งเนื่องจาก transposition: การเรียงสับเปลี่ยนที่แตกต่างกันของลำดับการเคลื่อนไหวจบลงในตำแหน่งเดียวกัน
- คล้ายกับเซต `closed` ใน `Graph-Search` (บทบรรยายที่ 2) คุ้มค่าที่จะเก็บการประเมินของสถานะเพื่อให้การปรากฏขึ้นอีกครั้งของสถานะไม่ต้องคำนวณใหม่
- ควรใช้โครงสร้างข้อมูลแบบใดเพื่อเก็บและค้นหาค่าของตำแหน่งอย่างมีประสิทธิภาพ?

การตัดสินใจแบบเรียลไทม์ที่ไม่สมบูรณ์

- ภายใต้ ข้อจำกัดของเวลา การค้นหาวิธีแก้ปัญหาที่แน่นอนไม่สามารถทำได้ในเกมจริงส่วนใหญ่
- วิธีแก้ปัญหา: ตัดการค้นหาให้เร็วขึ้น
  - แทนที่ฟังก์ชัน $\mathrm{utility}(s)$ ด้วย ฟังก์ชันการประเมิน heuristic $\mathrm{eval}(s)$ ที่ประมาณค่า utility ของสถานะ
  - แทนที่การทดสอบสถานะสิ้นสุดด้วย การทดสอบการตัด ที่ตัดสินใจว่าเมื่อใดจะหยุดการขยายสถานะ

การค้นหา $\alpha - \beta$ สามารถปรับให้ใช้งาน H-Minimax ได้หรือไม่?

ฟังก์ชันการประเมิน

- ฟังก์ชันการประเมิน $\mathrm{eval}(s)$ ส่งกลับ การประมาณ ของ utility ที่คาดหวังของเกมจากตำแหน่ง s ที่กำหนด
- การคำนวณ ต้องสั้น (นั่นคือจุดทั้งหมดที่จะค้นหาให้เร็วขึ้น)
- ในอุดมคติ การประเมินควร จัดลำดับ สถานะในลักษณะเดียวกับใน Minimax
    - ค่าการประเมินอาจแตกต่างจากค่า minimax ที่แท้จริง ตราบใดที่ลำดับถูกรักษาไว้
- ในสถานะที่ไม่ใช่สถานะสิ้นสุด ฟังก์ชันการประเมินควรมี ความสัมพันธ์ อย่างแรงกับโอกาสที่แท้จริงในการชนะ

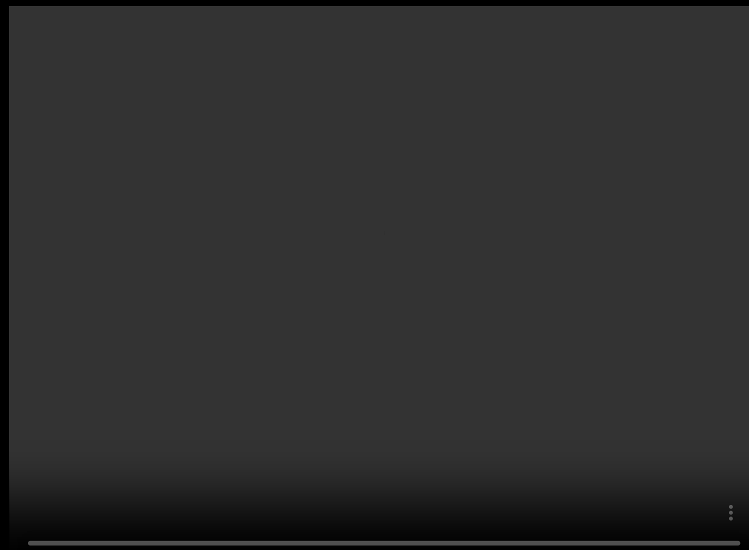
ความสงบ (Quiescence)

- สถานะเหล่านี้แตกต่างกันเพียงในตำแหน่งของรุกที่มุมล่างขวา
- อย่างไรก็ตาม ฝ่ายดำมีข้อได้เปรียบใน (a) แต่ไม่ใช่ใน (b)
- หากการค้นหาหยุดใน (b) ฝ่ายดำจะไม่เห็นว่าการเคลื่อนไหวครั้งต่อไปของฝ่ายขาวคือการจับควีนของมัน ทำให้ได้เปรียบ
- การตัดควรใช้เฉพาะกับตำแหน่งที่เป็น quiescent
    - กล่าวคือ สถานะที่ไม่น่าจะแสดงการเปลี่ยนแปลงค่าอย่างรุนแรงในอนาคตอันใกล้

ผลกระทบขอบฟ้า (Horizon effect)

ฟังก์ชันการประเมิน ไม่สมบูรณ์เสมอ

- หากไม่มองลึกพอ การเคลื่อนไหวที่ไม่ดี อาจปรากฏเป็น การเคลื่อนไหวที่ดี (ตามที่ประมาณโดยฟังก์ชันการประเมิน) เพราะผลที่ตามมาถูกซ่อนไว้เกินขอบเขตการค้นหา
    - และในทางกลับกัน!
- บ่อยครั้ง ยิ่งฟังก์ชันการประเมินถูกฝังลึกลงไปในต้นไม้มากเท่าใด คุณภาพของฟังก์ชันการประเมินก็ยิ่งสำคัญน้อยลง



ตัดที่ความลึก 2, การประเมิน = ยิ่งใกล้จุดยิ่งดี

Credits: CS188, UC Berkeley.

ตัดที่ความลึก 10, การประเมิน = ยิ่งใกล้จุดยิ่งดี

เกมหลายผู้เล่น

- จะเป็นอย่างไรถ้าเกมไม่ใช่ zero-sum หรือมี หลายผู้เล่น?
- การขยายผลของ Minimax:
  - สถานะสิ้นสุดถูกติดป้ายด้วย tuple ของ utility (1 ค่าต่อผู้เล่น)
  - สถานะกลางก็ถูกติดป้ายด้วย tuple ของ utility
  - ผู้เล่นแต่ละคนเพิ่มองค์ประกอบของตนเองให้มากที่สุด
  - อาจให้ผลเป็นความร่วมมือและการแข่งขันแบบไดนามิก

—

Credits: CS188, UC Berkeley.

เกมสุ่ม (Stochastic games)

เกมสุ่ม

- ในชีวิตจริง เหตุการณ์ภายนอกที่คาดเดาไม่ได้มากมายสามารถทำให้เราตกอยู่ในสถานการณ์ที่ไม่คาดคิดได้
- เกมที่สะท้อนความไม่แน่นอนนี้เรียกว่า เกมสุ่ม รวมถึงองค์ประกอบสุ่ม เช่น:
  - ความสุ่มที่ชัดเจน: การทอยลูกเต๋า
  - การกระทำอาจล้มเหลว: เมื่อขับหุ่นยนต์ ล้ออาจลื่น

- ในต้นไม้เกม องค์ประกอบสุ่มนี้สามารถ สร้างแบบจำลอง ด้วย โหนดโอกาส ที่แมปคู่สถานะ-การกระทำไปยังชุดของผลลัพธ์ที่เป็นไปได้ พร้อมด้วย ความน่าจะเป็น ของแต่ละรายการ
- สิ่งนี้เทียบเท่ากับการพิจารณาสภาพแวดล้อมเป็น ผู้เล่นสุ่ม พิเศษที่เคลื่อนไหวหลังจากผู้เล่นอื่นๆ แต่ละคน

—

Credits: CS188, UC Berkeley.

ต้นไม้เกมสุ่ม

Expectiminimax

- เนื่องจากความไม่แน่นอนในผลลัพธ์ของการกระทำ สถานะจึงไม่มีค่า minimax ที่ แน่นอน อีกต่อไป
- อย่างไรก็ตาม เราสามารถคำนวณค่า คาดหวัง ของสถานะภายใต้การเล่นที่เหมาะสมโดยฝ่ายตรงข้ามได้
  - กล่าวคือ ค่าเฉลี่ยของผลลัพธ์ที่เป็นไปได้ทั้งหมดของโหนดโอกาส
  - ค่า minimax สอดคล้องกับผลลัพธ์กรณีเลวร้ายที่สุด

การเคลื่อนไหวแบบเหตุผลหมายความว่า agent จะประสบความสำเร็จหรือไม่?

ฟังก์ชันการประเมิน

- เช่นเดียวกับ $\mathrm{minimax}(n)$ ค่าของ $\mathrm{expectiminimax}(n)$ อาจถูกประมาณโดยการหยุดการเรียกซ้ำเร็วและใช้ฟังก์ชันการประเมิน
- อย่างไรก็ตาม เพื่อให้ได้การเคลื่อนไหวที่ถูกต้อง ฟังก์ชันการประเมินควรเป็น การเปลี่ยนแปลงเชิงเส้นในเชิงบวก ของ utility ที่คาดหวังของสถานะ
    - ไม่เพียงพอที่ฟังก์ชันการประเมินจะรักษาลำดับไว้เท่านั้น
- หากเราสมมติขอบเขตของฟังก์ชัน utility การค้นหา $\alpha - \beta$ สามารถปรับให้เข้ากับเกมสุ่มได้

การเปลี่ยนแปลงที่รักษาลำดับบนค่าใบไม้เปลี่ยนการเคลื่อนไหวที่ดีที่สุด

Monte Carlo Tree Search

การประเมิน Random playout

- เพื่อประเมินสถานะ ให้อัลกอริทึม เล่นกับตัวเอง โดยใช้ การเคลื่อนไหวสุ่ม หลายพันครั้ง
- ลำดับของการเคลื่อนไหวสุ่มเรียกว่า random playout
- ใช้สัดส่วนของการชนะเป็นการประเมินสถานะ
- กลยุทธ์นี้ ไม่ต้องการความรู้ในโดเมน!
    - เพียงแค่ต้องมีเอนจิ้นเกมเท่านั้น

Monte Carlo Tree Search

จุดสำคัญของ MCTS คือการวิเคราะห์การเคลื่อนไหวที่มีแนวโน้มดีที่สุด โดยประเมินแบบเพิ่มหน่วยด้วย random playout

แต่ละโหนด n ในต้นไม้การค้นหาปัจจุบันเก็บค่าสองค่า:

- จำนวนการชนะ $Q(n, p)$ ของผู้เล่น p สำหรับ playout ทั้งหมดที่ผ่าน n
- จำนวน $N(n)$ ของครั้งที่ n ถูกเยี่ยมชม

อัลกอริทึมค้นหาต้นไม้เกมดังนี้:

1. การเลือก (Selection): เริ่มจากรูท เลือกโหนดลูกต่อเนื่องลงไปจนถึงโหนด n ที่ยังขยายไม่สมบูรณ์
2. การขยาย (Expansion): เว้นแต่ว่า n เป็นสถานะสิ้นสุด ให้สร้างโหนดลูกใหม่ n'
3. การจำลอง (Simulation): เล่น random playout จาก n'
4. การส่งกลับ (Backpropagation): ใช้ผลลัพธ์ของ playout เพื่อปรับปรุงข้อมูลในโหนดบนเส้นทางจาก n' ไปยังรูท

ทำซ้ำ 1-4 ตราบใดที่งบประมาณเวลาอนุญาต เลือกการเคลื่อนไหวโดยตรงที่ดีที่สุด

การสำรวจและการใช้ประโยชน์

เมื่อมีงบประมาณจำกัดของ random playout ประสิทธิภาพของ MCTS ขึ้นอยู่กับการเลือกโหนดที่ถูกเลือกในขั้นตอน 1 อย่างมาก

ระหว่างการสำรวจกิ่งในขั้นตอนการเลือก นโยบาย UCB1 เลือกโหนดลูก n' ของ n ที่ทำให้มากที่สุด $\frac{Q(n',p)}{N(n')} + c\sqrt{\frac{\log N(n)}{N(n')}}$.

- พจน์แรกสนับสนุนการ ใช้ประโยชน์ จากโหนดที่ให้ผลตอบแทนสูงกว่า
- พจน์ที่สอง สนับสนุนการ สำรวจ โหนดที่ถูกเยี่ยมชมน้อยกว่า
- ค่าคงที่ $c > 0$ ควบคุมการแลกเปลี่ยนระหว่างการใช้ประโยชน์และการสำรวจ

สมมติฐานการสร้างแบบจำลอง

จะเป็นอย่างไรถ้าสมมติฐานของเราไม่ถูกต้อง?

—

Credits: CS188, UC Berkeley.

การตั้งค่า

- P_1: Pacman ใช้การค้นหาความลึก 4 ด้วยฟังก์ชันการประเมินที่หลีกเลี่ยงปัญหา โดยสมมติว่าผีทำตาม P_2
- P_2: ผีใช้การค้นหาความลึก 2 ด้วยฟังก์ชันการประเมินที่แสวงหา Pacman โดยสมมติว่า Pacman ทำตาม P_1
- P_3: Pacman ใช้การค้นหาความลึก 4 ด้วยฟังก์ชันการประเมินที่หลีกเลี่ยงปัญหา โดยสมมติว่าผีทำตาม P_4
- P_4: ผีเคลื่อนไหวแบบสุ่ม

Minimax Pacman (P_1) vs. Adversarial ghost (P_2)

—

Credits: CS188, UC Berkeley.

Minimax Pacman (P_1) vs. Random ghost (P_4)

Credits: CS188, UC Berkeley.

Expectiminimax Pacman ($\mathbf{P}_3$) vs. Random ghost ($\mathbf{P}_4$)

Expectiminimax Pacman (P_3) vs. Adversarial ghost (P_2)

โปรแกรมเกมที่ทันสมัย

หมากฮอส (Checkers)

1951

ผู้เล่นคอมพิวเตอร์คนแรกโดย Christopher Strachey

1994

โปรแกรมคอมพิวเตอร์ Chinook ยุติการครองราชย์ 40 ปีของแชมป์เปียนมนุษย์ Marion Tinsley

- ไลบรารีของการเคลื่อนไหวเปิดเกมจากแกรนด์มาสเตอร์
- อัลกอริทึมการค้นหาลึก
- ฟังก์ชันการประเมินการเคลื่อนไหวที่ดี (ตาม linear model)
- ฐานข้อมูลสำหรับตำแหน่งทั้งหมดที่มีแปดตัวหรือน้อยกว่า

2007

หมากฮอส ถูกแก้ไขแล้ว วิธีแก้ปัญหาอ่อนถูกพิสูจน์ทางคอมพิวเตอร์

- จำนวนการคำนวณที่เกี่ยวข้องคือ 10^{14} ในช่วงเวลา 18 ปี
- การเสมอได้รับการรับประกันเสมอหากไม่มีผู้เล่นคนใดทำผิดพลาด

Schaeffer, Jonathan, et al. "Checkers is solved." science 317.5844 (2007): 1518-1522.

หมากรุก (Chess)

1997

- Deep Blue เอาชนะแชมป์เปียนมนุษย์ Gary Kasparov
  - การประเมินตำแหน่ง 200000000 ครั้งต่อวินาที
  - ฟังก์ชันการประเมินที่ซับซ้อนมาก
  - วิธีการที่ไม่เปิดเผยสำหรับการขยายเส้นการค้นหาบางเส้นถึง 40 ply
- โปรแกรมสมัยใหม่ (เช่น Stockfish หรือ AlphaZero) ดีกว่า แม้จะมีประวัติศาสตร์น้อยกว่า
- หมากรุกยังคง ยังไม่ถูกแก้ไข เนื่องจากความซับซ้อนของเกม

โกะ (Go)

เป็นเวลานาน โกะถูกมองว่าเป็นจอกศักดิ์สิทธิ์ของ AI เนื่องจากขนาดของต้นไม้เกม

- บนกระดาน 19x19 จำนวนตำแหน่งที่ถูกกฎหมายคือ $\pm 2 \times 10^{170}$
- ส่งผลให้เกิด $\pm 10^{800}$ เกม เมื่อพิจารณาความยาว 400 หรือน้อยกว่า

2010-2014

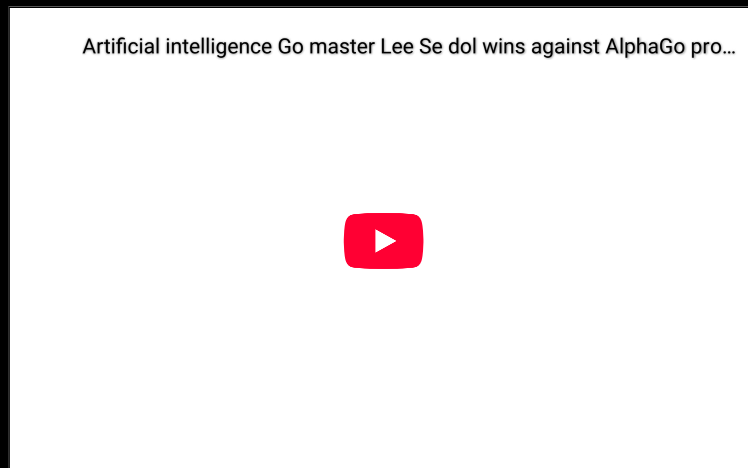
การใช้ Monte Carlo tree search และ machine learning ผู้เล่นคอมพิวเตอร์มาถึงระดับ dan ต่ำ

2015-2017

Google Deepmind ประดิษฐ์ AlphaGo

- 2015: AlphaGo เอาชนะ Fan Hui แชมป์เปี้ยนโกะยุโรป
- 2016: AlphaGo เอาชนะ Lee Sedol (4-1) แกรนด์มาสเตอร์ 9-dan
- 2017: AlphaGo เอาชนะ Ke Jie ผู้เล่นมนุษย์อันดับ 1 ของโลก

AlphaGo ผรวม Monte Carlo tree search และ deep learning ด้วยการฝึกอบรมอย่างกว้างขวาง ทั้งจากการเล่นของมนุษย์และคอมพิวเตอร์



การรายงานข่าวสำหรับชั่วโมงแห่งชัยชนะของ AlphaGo เหนือ Lee Sedol

2017

AlphaGo Zero ผสม Monte Carlo tree search และ deep learning ด้วยการฝึกแบบ reinforcement ด้วยการเล่นด้วยตัวเองเพียงอย่างเดียว

—

Credits: AlphaGo Zero: Learning from scratch

สรุป

- เกมหลายผู้เล่นเป็นรูปแบบต่างๆ ของปัญหาการค้นหา
- ความยากคือการพิจารณาข้อเท็จจริงที่ว่าฝ่ายตรงข้ามอาจทำงานได้อย่างไรก็ได้
  - วิธีแก้ปัญหาที่เหมาะสมคือ กลยุทธ์ ไม่ใช่ลำดับการกระทำที่คงที่
- Minimax เป็นอัลกอริทึมที่เหมาะสมสำหรับเกม zero-sum สองผู้เล่นแบบกำหนดได้ ผลัดกันเล่น ที่มีข้อมูลสมบูรณ์
  - เนื่องจากข้อจำกัดของเวลาในทางปฏิบัติ การสำรวจต้นไม้เกมทั้งหมดมักจะ ไม่สามารถทำได้
  - การประมาณสามารถทำได้ด้วย heuristic ลดเวลาการคำนวณ
  - Minimax สามารถปรับให้เข้ากับเกมสุ่มได้
  - Minimax สามารถปรับให้เข้ากับเกมที่มีผู้เล่นมากกว่า 2 คนได้
- พฤติกรรมที่เหมาะสม สัมพันธ์กัน และขึ้นอยู่กับสมมติฐานที่เราทำเกี่ยวกับโลก

จบ