1 AI101-04: Local Search and Adversarial Search

Diese Einheit behandelt Suchstrategien für zwei spezielle Problemklassen:

- **Lokale Suche:** Optimierungsprobleme, bei denen der Pfad zur Lösung irrelevant ist und nur der Endzustand zählt (z.B. N-Damen-Problem, Chip-Design).
- **Adversarial Search:** Probleme, bei denen ein Agent gegen einen Gegner agiert (Spiele wie Schach oder Go).

1.1 Lokale Suche (Local Search)

Bei vielen Optimierungsproblemen ist der Zustandsraum riesig oder unendlich, aber der Weg zum Ziel ist unwichtig. Lokale Suchalgorithmen operieren auf einem einzelnen aktuellen Zustand (oder einer kleinen Menge) und bewegen sich nur zu dessen Nachbarn.

Eigenschaften der Lokalen Suche

- **Speichereffizienz:** Verbraucht meist konstanten Speicher ($O(1)$ oder $O(k)$).
- **Anwendung:** Geeignet für riesige oder kontinuierliche Zustandsräume.
- **Ziel:** Finden des globalen Optimums einer **Zielfunktion** (Objective Function).

1.1.1 Hill Climbing (Bergsteigen)

Hill Climbing ist der einfachste lokale Suchalgorithmus ("Gierige lokale Suche"). Er versucht kontinuierlich, den aktuellen Zustand zu verbessern, indem er zum besten Nachbarn wechselt.

Algorithmus:

1. Starte mit einem zufälligen Zustand.
2. Generiere alle Nachbarn des aktuellen Zustands.
3. Wähle den Nachbarn mit der besten Bewertung (höchster Wert der Zielfunktion).
4. Wenn der beste Nachbar besser ist als der aktuelle Zustand: Gehe dorthin.
5. Sonst: Terminiere (Gipfel erreicht).

Metapher: „Das Erklimmen des Mount Everest bei dichtem Nebel und Amnesie."

Probleme des Hill Climbing: Der Algorithmus garantiert nicht das Finden des globalen Optimums, da er in lokalen Optima stecken bleiben kann.

- **Lokales Maximum**: Ein Gipfel, der höher ist als alle direkten Nachbarn, aber niedriger als das globale Maximum.
- **Plateau**: Ein flacher Bereich, in dem alle Nachbarn den gleichen Wert haben. Der Algorithmus hat keine Richtungsinformation (Random Walk notwendig).
  - **Flat Local Maximum:** Ein Plateau, das ein lokales Maximum ist.
  - **Shoulder:** Ein Plateau, von dem aus es noch weiter bergauf gehen könnte.
- **Ridge (Grat)**: Eine schmale Erhebung, die ansteigt, bei der aber alle direkten Nachbarn (z.B. Norden, Süden, Osten, Westen) bergab führen. Der Anstieg verläuft oft diagonal, was für einfache Bewegungsoperatoren schwer zu erkennen ist.

Varianten zur Verbesserung:

- **Stochastic Hill Climbing:** Wählt zufällig einen der besseren Nachbarn aus (nicht zwingend den besten). Dies erhöht die Chance, lokale Maxima zu umgehen oder Plateaus zu überwinden.

- **Random-Restart Hill Climbing:** Führt den Algorithmus mehrfach mit unterschiedlichen zufälligen Startzuständen aus.
  - Wenn die Anzahl der Versuche gegen unendlich geht, nähert sich die Wahrscheinlichkeit, das globale Optimum zu finden, 1 an (asymptotisch vollständig).

1.1.2 Simulated Annealing

Inspiriert vom physikalischen Prozess des Ausglühens in der Metallurgie (Erhitzen und langsames Abkühlen, um stabile Kristallstrukturen zu bilden). Ziel ist es, lokale Maxima zu verlassen, indem man *schlechte* Züge mit einer gewissen Wahrscheinlichkeit zulässt.

Funktionsweise

Kombiniert Hill Climbing (Effizienz) mit Random Walk (Exploration).
- Es gibt einen Temperaturparameter T, der gemäß einem Abkühlungsplan (*schedule*) sinkt.
- Ein zufälliger Nachbar wird gewählt.
- Ist der Nachbar besser ($\Delta E > 0$): Der Zug wird immer akzeptiert.
- Ist der Nachbar schlechter ($\Delta E < 0$): Der Zug wird mit Wahrscheinlichkeit P akzeptiert:

$$P = e^{\frac{\Delta E}{T}}$$

Interpretation:

- Hohes T: Hohe Wahrscheinlichkeit, schlechte Züge zu akzeptieren (ähnlich Random Walk).
- $T \to 0$: Wahrscheinlichkeit sinkt gegen 0 (ähnlich Hill Climbing).
- Wird T langsam genug gesenkt, findet der Algorithmus garantiert das globale Optimum.

1.1.3 Local Beam Search

Statt nur einen Zustand zu betrachten, verfolgt dieser Algorithmus k Zustände parallel.

- Starte mit k zufälligen Zuständen.
- Generiere in jedem Schritt alle Nachfolger aller k Zustände.
- Wenn einer davon das Ziel ist: Stopp.
- Sonst: Wähle die k besten Nachfolger aus der *gesamten* Menge aller Nachfolger aus.

Unterschied zu k mal Random-Restart: Die k Zustände sind nicht unabhängig. Informationen werden geteilt, da sich die Suche auf vielversprechende Regionen des Zustandsraums konzentriert („Wo ein guter Zustand ist, sind oft auch andere“).

1.2 Suche in kontinuierlichen Räumen

Viele reale Probleme (z.B. Training neuronaler Netze) haben kontinuierliche Zustandsräume.

1.2.1 Gradient Descent (Gradientenabstieg)

Wenn die Zielfunktion $f(\mathbf{x})$ differenzierbar ist, nutzen wir den Gradienten ∇f, um die Richtung des steilsten Anstiegs/Abstiegs zu finden.

Update-Regel

Um eine Kostenfunktion $L(\theta)$ zu minimieren, aktualisieren wir die Parameter θ iterativ:

$$\theta \leftarrow \theta - \alpha \nabla L(\theta)$$

Dabei ist α die **Lernrate** (Step Size).

Wahl der Lernrate α:

- **Zu klein:** Konvergenz ist sehr langsam, viele Iterationen nötig.
- **Zu groß:** Der Algorithmus kann über das Ziel hinausschießen, oszillieren oder sogar divergieren.

1.3 Adversarial Search (Spiele)

In Multi-Agenten-Umgebungen beeinflussen die Aktionen anderer Agenten das Ergebnis. Wir betrachten **Nullsummenspiele** (Zero-Sum Games) mit perfekter Information (z.B. Schach, Tic-Tac-Toe).

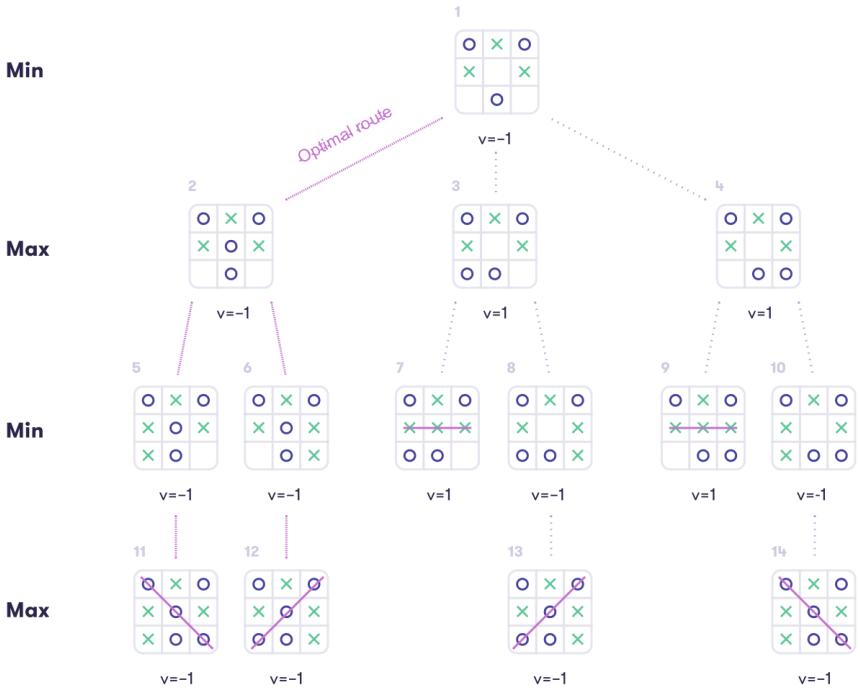
- **MAX:** Unser Agent, möchte den Nutzen (Utility) maximieren.
- **MIN:** Der Gegner, möchte den Nutzen minimieren (bzw. seinen eigenen maximieren).

1.3.1 Minimax-Algorithmus

Der Minimax-Algorithmus berechnet den optimalen Zug durch rekursive Suche im Spielbaum.

Minimax-Wert eines Knotens:

- **Terminal-Knoten (Blatt):** Utility-Wert des Zustands (z.B. +1 Sieg, -1 Niederlage, 0 Unentschieden).
- **MAX-Knoten:** max(Werte der Nachfolger).
- **MIN-Knoten:** min(Werte der Nachfolger).



Eigenschaften:

- **Vollständig:** Ja, bei endlichem Baum.
- **Optimal:** Ja, gegen einen optimalen Gegner.
- **Zeitkomplexität:** $O(b^m)$ (exponentiell), mit Verzweigungsfaktor b und Tiefe m.
- **Platzkomplexität:** $O(b \cdot m)$ (bei Tiefensuche).

1.3.2 Alpha-Beta Pruning

Alpha-Beta Pruning ist eine Optimierung des Minimax-Algorithmus, die irrelevante Zweige des Suchbaums ignoriert („abschneidet“), ohne das Ergebnis zu verändern.

Idee: Wenn wir bereits einen Zug gefunden haben, der uns einen gewissen Wert garantiert, und wir in einem anderen Zweig sehen, dass der Gegner uns dort auf einen schlechteren Wert zwingen kann, müssen wir diesen Zweig nicht weiter untersuchen.

Parameter α und β

Wir führen zwei Werte durch die Rekursion mit:

- α **(Alpha)**: Der Wert der besten Alternative für MAX, die bisher auf dem Pfad gefunden wurde (untere Schranke für MAX). Initial: $-\infty$.
- β **(Beta)**: Der Wert der besten Alternative für MIN, die bisher auf dem Pfad gefunden wurde (obere Schranke für MIN). Initial: $+\infty$.

Pruning-Regeln:

1. **MIN-Knoten (Update β):**
   - Berechne Wert v des Kindknotens.
   - Wenn $v \leq \alpha$: **Pruning!** (MAX wird diesen Ast nie wählen, da er α bereits sicher hat).
   - Sonst: $\beta = \min(\beta, v)$.
2. **MAX-Knoten (Update α):**
   - Berechne Wert v des Kindknotens.
   - Wenn $v \geq \beta$: **Pruning!** (MIN wird diesen Ast nie zulassen, da er β bereits sicher hat).
   - Sonst: $\alpha = \max(\alpha, v)$.

Effizienz:

- Im besten Fall (perfekte Sortierung der Züge): Komplexität reduziert sich auf $O(b^{m/2})$. Das bedeutet effektiv eine Verdopplung der durchsuchbaren Tiefe.
- Im schlechtesten Fall (schlechteste Sortierung): Keine Verbesserung, $O(b^m)$.
- *Move Ordering* ist daher entscheidend (z.B. Schlagen von Figuren zuerst prüfen).

1.3.3 Umgang mit Ressourcenbeschränkungen

Da komplette Suchbäume für komplexe Spiele (Schach: 10^{40} Zustände, Go: 10^{170}) zu groß sind, wird die Suche oft bei einer Tiefe d abgebrochen.

- **Heuristische Evaluierungsfunktion (H-Minimax):** Schätzt den Wert einer Position an den Blättern der begrenzten Suche (z.B. Materialwert im Schach: Bauer=1, Dame=9).
- **Horizon Effect:** Ein unvermeidbarer negativer Event (z.B. Verlust der Dame) wird durch „Verzögerungstaktiken“ aus dem Suchhorizont geschoben, sodass die Evaluierung fälschlicherweise positiv wirkt.