

Grundlagen der Künstlichen Intelligenz

17. Klassische Suche: IDA*

Malte Helmert

Universität Basel

27. März 2015

Grundlagen der Künstlichen Intelligenz

27. März 2015 — 17. Klassische Suche: IDA*

17.1 IDA*: Idee

17.2 IDA*: Algorithmus

17.3 IDA*: Eigenschaften

17.4 Zusammenfassung

Klassische Suche: Überblick

Kapitelüberblick klassische Suche:

- ▶ 5.–7. Grundlagen
- ▶ 8.–12. Basisalgorithmen
- ▶ 13.–20. heuristische Algorithmen
 - ▶ 13. Heuristiken
 - ▶ 14. Analyse von Heuristiken
 - ▶ 15. Bestensuche als Graphensuche
 - ▶ 16. Gierige Bestensuche, A*, Weighted A*
 - ▶ 17. IDA*
 - ▶ 18. A*: Optimalität, Teil I
 - ▶ 19. A*: Optimalität, Teil II
 - ▶ 20. A*: Vollständigkeit und Komplexität

17.1 IDA*: Idee

IDA*

Der Hauptnachteil der vorgestellten Bestensuchalgorithmen ist ihr Speicheraufwand.

Idee: verfolge einen ähnlichen Ansatz wie bei der iterativen Tiefensuche

- ▶ beschränkte Tiefensuche mit steigenden Schranken
- ▶ statt der **Tiefe** beschränken wir **f** (in diesem Kapitel $f(n) := g(n) + h(n.state)$, wie bei A*)
- ↪ **IDA*** (**iterative-deepening A***)
- ▶ ist eine **Baumsuche**, anders als die vorigen Bestensuchalgorithmen

17.2 IDA*: Algorithmus

Anmerkungen zum Algorithmus (1)

- ▶ Wir beschreiben eine IDA*-Implementierung mit **expliziten Suchknoten**.
- ▶ Effizientere Implementierungen lassen Suchknoten **implizit**, so wie bei der Tiefensuche in Kapitel 12.

Anmerkungen zum Algorithmus (2)

- ▶ Unsere rekursiven Aufrufe liefern zwei Werte:
 - ▶ **f_limit** , die nächste sinnvolle f -Schranke für den Teilbaum, der in dem Aufruf betrachtet wurde (oder **none**, wenn eine Lösung gefunden wurde)
 - ▶ **solution**, die gefundene Lösung (bzw. **none**)
- ▶ Effizientere Implementierungen speichern diese Werte in Instanzvariablen oder einer Closure, um Zeit für Werteübergabe zu sparen

IDA*: Pseudo-Code (Hauptprozedur)

IDA*: Hauptprozedur

```

 $n_0 := \text{make\_root\_node}()$ 
 $f\_limit := 0$ 
while  $f\_limit \neq \infty$ :
     $\langle f\_limit, solution \rangle := \text{recursive\_search}(n_0, f\_limit)$ 
    if  $solution \neq \text{none}$ :
        return  $solution$ 
return unsolvable

```

IDA*: Pseudo-Code (Tiefensuche)

function recursive_search(n, f_limit):

```

if  $f(n) > f\_limit$ :
    return  $\langle f(n), \text{none} \rangle$ 
if  $\text{is\_goal}(n.\text{state})$ :
    return  $\langle \text{none}, \text{extract\_path}(n) \rangle$ 
 $\text{next\_limit} := \infty$ 
for each  $\langle a, s' \rangle \in \text{succ}(n.\text{state})$ :
    if  $h(s') < \infty$ :
         $n' := \text{make\_node}(n, a, s')$ 
         $\langle \text{rec\_limit}, solution \rangle := \text{recursive\_search}(n', f\_limit)$ 
        if  $solution \neq \text{none}$ :
            return  $\langle \text{none}, solution \rangle$ 
         $\text{next\_limit} := \min(\text{next\_limit}, \text{rec\_limit})$ 
return  $\langle \text{next\_limit}, \text{none} \rangle$ 

```

17.3 IDA*: Eigenschaften

IDA*: Eigenschaften

Erbt wichtige Eigenschaften von A* und iterativer Tiefensuche:

- ▶ **semi-vollständig**, wenn h sicher und $\text{cost}(a) > 0$ für alle Aktionen a
- ▶ **optimal**, wenn h zulässig
- ▶ **Speicheraufwand** $O(\ell b)$, wobei
 - ▶ ℓ : Länge des längsten erzeugten Pfads (bei Einheitskosten: beschränkt durch optimale Lösungskosten)
 - ▶ b : Verzweigungsgrad

⇒ Beweise?

IDA*: Diskussion

- ▶ im Vergleich zu A* potenziell erheblicher Overhead, da keine **Duplikate** erkannt werden
 - ↪ in vielen Zustandsräumen exponentiell langsamer
 - ↪ oft mit teilweiser Duplikateliminierung kombiniert (Zykelerkennung, Transpositionstabellen)
- ▶ Overhead durch **iteratives Anheben** der f -Schranke ist **oft vernachlässigbar**, aber **nicht immer**
 - ▶ besonders problematisch, wenn Aktionskosten stark variieren: dann kann es leicht passieren, dass jede neue f -Schranke nur wenige neue Suchknoten erreicht

17.4 Zusammenfassung

Zusammenfassung

- ▶ **IDA*** ist eine Baumsuchvariante von A* basierend auf iterativer Tiefensuche
- ▶ Hauptvorteil: **niedriger Speicheraufwand**
- ▶ Nachteil: **wiederholte Arbeit** kann signifikant sein
- ▶ am nützlichsten, wenn es **wenige Duplikate** gibt (Duplikateliminierung würde Speichervorteil verlieren)