Platz sparen

bessere Definition von Konfiguration in “Würmer versenken”:
aktuelle Aufstellung aller Würmer (+ Zeitlimits, etc.)

Position der Hindernisse ändert sich nicht, etc.

weiter reduzierbar (auf Kosten der Vollständigkeit):

- betrachte nur die ersten beiden Segmente der anderen Würmer
- ignoriere Zeitlimits

allgemein: erwarte, dass Gegner bestmöglich spielt

Kodieren

Datenstrukturen möglichst gering halten

Bsp.: `int` braucht evtl. weniger Platz als `bool[32]`

bei Konfigurationen?

Graphalgorithmen

Distanz zum Loch

sei p Position des Kopfes eines Wurms, l Position des Lochs

$dist(p, l)$ wegen Hindernissen nicht nur aus p und l berechenbar

stattdessen aus Distanz zwischen Nachbarn von p und l berechnen

$$dist(p,l) \quad = \quad \begin{cases} 0 & \text{falls } p = l \\ 1 + \min\{dist(p', l) \mid p' \text{ Nachbar von } p\} & \text{sonst} \end{cases}$$

liefert rekursiven Algorithmus

worst-case Laufzeit ∞ oder $O(5^{O(n)})$

Distanz zum Loch

3 Beobachtungen:

- $dist(p, l) = dist(l, p)$
- Distanz ändert sich für Würmer, aber nicht für Spielfelder
- sehr viele gleiche Teilprobleme

Dynamisches Programmieren

allgemeiner Begriff für Klasse von Algorithmen

- berechne Lösung bottom-up
- vermeide Mehrfachberechnung durch Abspeichern von Ergebnissen für Teilprobleme

s.a. Fibonaccizahlen

hier: Distanzen für alle Felder nur einmal von innen nach aussen berechnen

Kürzeste Wege

hilft um kürzesten Weg zum Loch zu finden

hilft nicht um kürzesten Weg von a zu b zu finden
$\leadsto$ Breitensuche auf Graphen

Idee:

1. halte in einer Queue alle Knoten, von denen aus gesucht werden soll; zu Begin b
2. nimm ersten Knoten c aus Queue,
3. markiere c als besucht
4. füge dessen unbesuchte Nachbarn in Queue ein
5. merke zu jedem dieser Knoten c als Vorgänger
6. halte an, wenn a gefunden wird

kürzester Weg von a nach b über Vorgängerrelation gegeben

Verbesserung

starte Suche in zwei Threads von a und b aus

Suche beendet, falls ein Thread Feld findet, welches von anderem markiert wurde

Wieso Verbesserung?

Übung: wie Algorithmus ändern, um kürzeste Wege von einem a zu **jedem** b zu finden?

Kürzeste Wege zwischen allen Knoten

wieder dynamisches Programmieren

gegeben n (freie) Felder, berechne $dist(a,b)$ für alle a, b

$$dist^{(i)}(a,b) = \text{Distanz zwischen } a \text{ und } b \text{ über } \leq i \text{ Schritte}$$

$$dist^{(0)}(a,b) = \begin{cases} 0 \ , & \text{falls } a = b \\ \infty \ , & \text{sonst} \end{cases}$$

$$dist^{(i+1)}(a,b) = \min\{ dist^{(i)}(c,b) + 1 \mid c \text{ Nachbar von } a \}$$

$$dist(a,b) = dist^{(n)}(a,b)$$

Übung: finde bessere Schranke für $dist$ als $dist^{(n)}$

Maximaler Fluss in einem Netzwerk

Gegeben Graph G, zwei Knoten s, t, “wieviel kann gleichzeitig von s nach t fliessen”?

Ford-Fulkerson-Methode (vereinfacht)

```
int f=0;
Graph g = G;
while (es gibt Pfad π in g von s nach t)
   entferne π aus g;
   f = f+1;
return f;
```

Annahme hier: jede Kante hat Kapazität 1

klar: maximaler Fluss von s nach t immer höchstens 1, daher besser Start- und Zielgebiete S und T

Entwurfsprinzipien für Heuristiken

Überblick

bereits erwähnt / angedeutet:

- Greedy-Algorithmen
- dynamisches Programmieren
- parallele Algorithmen

weitere:

- branch-and-bound
- divide-and-conquer
- best-first
- simulated annealing
- genetische Algorithmen

Branch-and-bound

Idee: schätze Kosten für eine Gesamtlösung nach oben hin ab

hier z.B.:
$k =$ Summe aller Distanzen zwischen eigenen Würmern und Loch $\cdot(1 + \epsilon)$

betrachte nicht solche Züge, die zu einer Konfiguration führen mit Bewertung $> k$

ϵ muss nicht fest sein, sondern kann sich im Laufe des Spiels ändern

Divide-and-conquer

Zerlege Problem in Teilprobleme, löse Teilprobleme, baue Teillösungen zu Gesamtlösung zusammen

hier vermutlich nicht sehr hilfreich wegen

- dynamischer Änderung der Spielsituation
- Gesamtlösung = Strategie lässt sich nur schwer zerlegen

Best-first

Strategie zum Durchsuchen des Graphen der Konfigurationen

funktioniert so wie depth-first, aber wählt den Nachfolger mit bester Bewertung zuerst

Lokale Optima

Problem mit den meisten Methoden:
Suche kann in lokales Optimum laufen, d.h. alle weiteren Züge liefern Konfiguration mit schlechterer Bewertung

Was tun gegen lokale Optima?

- wähle ab und zu zufällig schlechten Zug
  kann noch schlechter sein
- simulated annealing

Simulated Annealing

in Anlehnung an Züchten von Kristallen oder Herstellen von Metalllegierungen

Suche im Graph der Konfigurationen wird gesteuert durch Parameter T (“Temperatur”)

T liefert Wahrscheinlichkeit P, mit der Zug von Konfiguration c_1 nach c_2 gemacht / untersucht wird

Bsp.: sei b Bewertungsfunktion, Aufgabe Minimierung

$$P = \frac{1}{1 + e^{\frac{b(c_2) - b(c_1)}{T}}}$$

Genetische Algorithmen

fundamentaler Unterschied: untersuche gleichzeitig mehrere Lösungen

Analogie zur Natur:

- Lösungen = Chromosomen
- schlechte Lösungen sterben aus
- gute Lösungen pflanzen sich fort
- ab und zu mutieren Lösungen

Genetische Algorithmen

hier: Lösungen = Strategien, viel zu gross zum Abspeichern

aber Resultat eines Spiels eine Partie mit n Runden

letztendlich nur gemacht: n Züge

Chromosom = geordnete Folge aus n Zügen

benötigt:
Bewertung b der Güte eines Chromosoms (Maximierung), z.B.
Güte der Konfiguration, in die diese Zugfolge führt,
unter (Nicht-)Berücksichtigung der Züge des Gegners

Operationen auf Chromosomen

neue Mengen von Chromosomen c werden iterativ berechnet
in jedem Zyklus

- c stirbt aus mit W'keit, z.B. $P = 1 - e^{k \cdot b(c)}$
  evtl. auch Alter der Lösung einbeziehen

- c mutiert mit W'keit, z.B. $P = 10^{-6}$
  zufälliges Ersetzen eines Zuges

- c_1 und c_2 paaren sich mit W'keit, z.B. $P = e^{k \cdot (b(c_1)+b(c_2))}$
  bringt neue Lösung hervor, alte existieren weiterhin

Paarung

mehrere Möglichkeiten denkbar, z.B.

- 2 neue Lösungen durch Mischen erzeugen
- Reissverschlussverfahren, danach wieder auf Länge n trimmen
- Zerlegen in Blöcke, Güte einzelner Blöcke bestimmen, zusammensetzen von Blöcken
- ...

Unsicheres Wissen

Züge des Gegners

können wegen Platzkomplexität nicht vollständig analysiert werden

betrachte **alle** möglichen Züge des Gegners in einem Schritt

Funktion f: Spielfelder $\rightarrow$ `bool`,
$f(x) =$ `true` gdw. auf x ein gegnerischer Wurm stehen könnte

Verbesserung:
Annahme, dass alle Züge des Gegners gleich wahrscheinlich sind
speichere Wahrscheinlichkeit, mit der ein gegnerischer Wurm auf einem Feld steht

Problem: Werte müssen nach jedem Zug aktualisiert werden

Wahrscheinlichkeiten errechnen

a Spielfeld, $P_i(a) \in [0,1]$ Wahrscheinlichkeit, dass nach i Zügen auf a gegnerischer Wurm steht

$$P_0(a) = \begin{cases} 1 \ , & \text{falls momentan einer dort steht} \\ 0 \ , & \text{sonst} \end{cases}$$

$$P_{i+1}(a) = \sum_{\substack{b \text{ Nachbar von } a \\ \text{kein Hindernis}}} \frac{1}{5 \cdot n} \cdot P_i(b)$$

wobei $n =$ Anzahl gegnerischer Würmer

nur Überapproximation an echte Wahrscheinlichkeit

Verbesserungen

- berücksichtige mehr als nur Köpfe, für Würmer der Länge l:

$$P'_i(a) = \max\{P_i(a), \ldots, P_{i-l+1}(a)\}$$

- Züge des Gegners nicht gleich verteilt, bessere Züge mit grösserer Wahrscheinlichkeit

- Hindernisse in Rechnung miteinbeziehen