2

q = [(stan0, wiedza0)]

funkcja następnych stanow :: stan * wiedza -> [stan * wiedza]

3

> local beam search dla k = 1

wybieramy najlepszego lidera - **hill climbing**

> local beam search z jednym początkowym stanem, bez limitu następnych

zaczynamy od jednego, generujemy następników, brak limitu czyli nie odrzucamy żadnego; **BFS**

> symulowane wyżarzanie z T = 0

p = 0; nie wykonujemy psujących ruchów, czekamy aż wylosujemy lepszy; **fist choice hill climbing**

> symulowane wyżarzanie z T = inf

p = 1; zawsze wykonujemy jakiś ruch, nieważne czy dobry czy zły. **random walk**

> genetyk z populacją 1

albo wybieramy z nowej populacji lidera -> random walk

jeśli wybieramy lidera z nowej i starej -> first choice hill climbing

4

> Więzy (wyrażalne w SWI-Prologu) - obrazki logiczne

tuple_in([X1..Xn], lista wszystkich możliwych dobrych wierszy)

Niech **n** to długość wiersza, **c** to wektor wymagań (długości bloków), zmienne **Xi** są wynikowymi pikselami

Skonstruujmy zmienne **Si** (interpretacja: dodatkowe spacje przed **ci** w minimalnej konstrukcji (bloki oddzielone jedną spacją))

extra_spaces := n - sum c - (len c - 1)

Si in 0..extra_spaces

S0 + c0 + (1 + S1 + c1) + ... + (1 + Sk + ck) < n

Xi #<=> any(S0 + ... + Sj <= i < S0 + ... + Sj + cj)

Mając to dla jednego wiersza, uogólniamy dla zmiennych Xij, dla każdego wiersza i każdej kolumny, zmienne Xij są wspólne

5

> Uproszczone zadanie układania planu. Nie przejmujemy się dostępnością sal.
>
> - Mamy pewną liczbę zajęć do rozmieszczenia.
> - Zajęcia = klasa (tu: uczniowie) + nauczyciel.
> - każdemu zajęciu przypisać termin: 1..50, gdzie 1..10 poniedziałek, 11..20 wtorek, ...
> - żadna klasa nie może mieć okienka

zajęcia - zmienne: **[(Ci, Ti)]** `Class_i, Teacher_i`

Niech **ci** to nazwa i-tej klasy w liście zajęć, podobnie **ti**

`Ci = Ti` *w tym samym czasie* `ci = cj => Ci /= Cj` *te same klasy nie mogą mieć zajęc w tym samym czasie* `ti = tj => Ti /= Tj` *podobnie z nauczycielami* `???` *bez okienek*

7

> Zaproponuj sensowne kombinacje algorytmów a) alg ewolucyjne i hill climbing

ewoluujemy aż można poprawić, wykonujemy lokalnie najlepsze mutacje/krzyżowania

> b) A* i local beam search

kilka kroków A* przy generowaniu następników

> c) symulowane wyżarzanie i alg ewolucyjne

jeśli zmiana (krzyżowanie/mutacja) na lepsze to zrób, jeśli nie to z wyżarzanym prawdopodobieństwem zrób zmianę

> d) taboo search i alg ewolucyjne

nie zapominamy o historii, nie pozwalamy na cykle w ewolucji

> e) hill climbing i BFS

zachłanny BFS - kolejne kroki są coraz to lepsze (BFS bez rozważania kroków pogarszających)

8

> Hiperkrzyżówka Jak przedstawić jako problem więzowy? Podaj dwie reprezentacje

k^2 zmiennych, typu *litera*, każda kolumna/wiersz jako lista zmiennych jest słowem

(**tuples_in**)

2k zmiennych, typu *słowo*, pierwsze k słów są w dziedzinie wszystkich słów, drugie k zmiennych są tworzone przez wyciągnięcie odpowiednich części z pierwszych, a następnie że są słowami (**tuples_in**)

> Algorytm

backtracking

walk sat - wylosuj planszę, wybierz wiersz/kolumnę, której zmiena literek najlepiej poprawi sytuację na planszy, czasem psuj, resetuj jak długo nie znajdziesz

9

> Gra turowa. Zakładamy, że znamy algorytm (deterministyczny) przeciwnika. Dlaczego można potraktować zadanie znalezienia ruchu jako zadanie przeszukiwania?

Możemy wygenerować ruchy przed pierwszym ruchem. Wiemy, co przeciwnik zrobi w odpowiedzi na każdy nasz ruch.

Startowy stan. Mamy możliwości w postaci naszych ruchów. Każdy taki stan możemy odegrać używając algorytmu przeciwnika. Generuje to graf wszystkich możliwości. Daje możliwość znalezienia (najkrótszej) ścieżki do wygranej.

> a) Które algorytmy rozwiązywania zadań przeszukiwania można by zastosować do gier o złożoności szachów czy warcabów?

BFS, DFS, Dijkstra - odpadają bo wykładnicza pamięć

minimax kilka kroków ile mamy czasu, wybieramy aktualne liście

local beam search - trzymamy k najlepszych rozegrań, mamy nadzieję, że w końcu jakieś wygra

> b) Zaproponuj jakiś sposób użycia takiego algorytmu w prawdziwej grze

przy generowaniu następników nie wiemy co zrobi przeciwniki, ale możemy rozważyć wszystkie możliwe ruchy przeciwnika i założyć, że zrobi najbardziej bolący nas ruch.

do funkcji oceniającej dorzucamy następny, najbardziej pesymistyczny dla nas, ruch przeciwnika

10

> Wybierz jedną grę: lis i gęsi, breakthrough, pentago, skoczki.

breakthrough - same pionki, mogą chodzić do przodu i lekko w lewo, i lekko w prawo, bicia przez stawanie na, win gdy dojdzie do końca

> Zaproponuj heurystyczną funkcję oceniającą sytuację na planszy.

- (a) ilość pionków

- (b) odległość do końca (min?)
- (inf) pewna wygrana (niemożliwość zablokowania)