Sztuczna Inteligencja

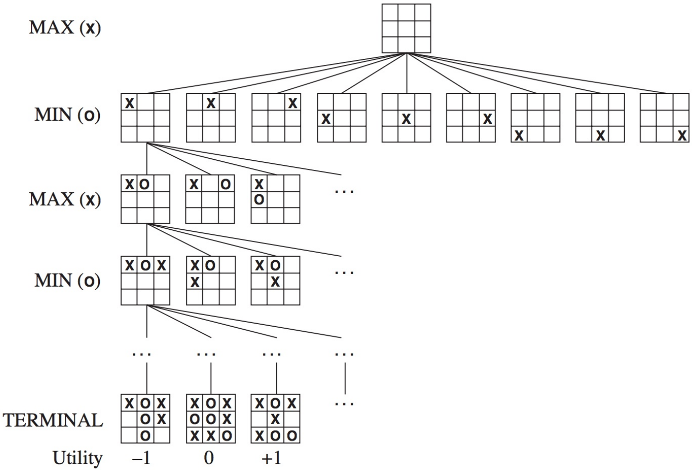
Soma Dutta

Wydział Matematyki i Informatyki, UWM w Olsztynie
soma.dutta@matman.uwm.edu.pl

Wykład - 4: Gry i Algorytmy spełniające więzy

Semestr letni 2022

- Aby sformułować grę jako problem przeszukiwania, potrzebujemy następujące pojęcia:
  - Initial State (Stan początkowy): określa początkową konfigurację gry
  - PLAYER(s): określa, który gracz ma ruch w stanie s
  - ACTION(s): Zwraca zbiór czynności poprawnych (ruchów) w stanie s
  - RESULT(s, a): Relacja przejścia, która określa wynik działania *a* lub ruchu w stanie s
  - TERMINAL-TEST(s): Test zwraca wartość prawda, gdy gra się skończy, a fałsz w przeciwnym razie. Stany, w których gra się kończy, nazywane są stanami końcowymi (terminal states).
  - UTILITY(s, p): Funkcja użyteczności (funkcja celu lub funkcja wypłaty) (objective function or payoff function) określa ostateczną wartość liczbową dla gry, która kończy się w stanie terminalnym s dla gracza p. (np. w szachach wyniki to wygrana (1), strata (0) lub remis ($\frac{1}{2}$).


MAX (x)
MIN (o)
MAX (x)
MIN (o)
TERMINAL
Utility
–1
0
+1

Utility function

- Funkcja użyteczności to odwzorowanie stanów świata na liczby rzeczywiste. Te liczby są interpretowane jako miary poziomu zadowolenia agenta w danym stanie. Kiedy agent nie jest pewien, z jakim stanem świata ma do czynienia, jego użyteczność jest definiowana jako oczekiwana wartość jego funkcji użyteczności w odniesieniu do odpowiedniego rozkładu prawdopodobieństwa stanów.
- Krotka (N, A, u) to gra postaci normalne (skończona, n-osobowa) gdzie:
  - N jest skończonym zbiórem n graczy, indeksowanych według i;
  - $A = A_1 \times A_2 \times \ldots \times A_n$, gdzie A_i to skończony zbiór działań dostępnych dla gracza i. Każdy wektor $a = (a_1, a_2, \ldots, a_n)$ $\in A$ jest nazywany profilem działania.
  - $u = (u_1, u_2, \ldots, u_n)$ gdzie każdy $u_i : A \mapsto R$ to funkcja użyteczności (lub funkcja wypłaty) dla każdego gracza i.
- W przypadku gry dwuosobowej $N = 2$.

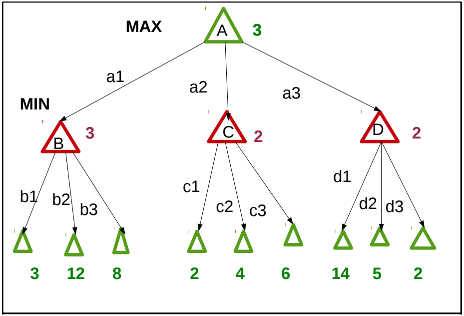
Strategia

- Biorąc pod uwagę zbiór dostępnych działań agenta MAX (MIN), jedną ze strategii może być polegać na wybieraniu deterministycznie jednej akcji. Taka strategia nazywa się czystą strategią (pure strategy). Wybór czystej strategii nazywamy dla każdego agenta profilem czystej strategii (pure strategy profile).
- Agenci mogą również przypisywać prawdopodobieństwa (na podstawie swoich preferencji) akcji ze zbioru dostępnych akcji. Tak więc każda strategia agenta jest przypisaniem z pewnym prawdopodobieństwem. Taka strategia nazywa się mieszaną strategią (mixed strategy).
- W przypadku czystych strategii mówiliśmy o funkcji użyteczności (utility function), a w kontekście mieszanych strategii mówimy o oczekiwanej użyteczności (expected utility function).

Przykład i MINIMAX z ostatniego wykładu

- Wartość minmax węzła n, oznaczona przez $MINMAX(n)$, jest użytecznością (dla MAX) bycia w n.
- Ponieważ ten algorytm zawsze reprezentuje użyteczność dla MAX, MAX preferuje stan o maksymalnej wartości, a MIN próbuje obniżyć użyteczność MAX i dlatego wybiera stan o minimalnej wartości.

$$
\begin{aligned}
\mathrm{MINIMAX(s)} &= \mathrm{UTILITY(s)} && \text{jeśli TERMINAL-TEST(s),}\\
&= max_{a \in ACTION(s)} MINIMAX(RESULT(s,a)) && \text{jeśli PLAYER(s) = MAX,}\\
&= min_{a \in ACTION(s)} MINIMAX(RESULT(s,a)) && \text{jeśli PLAYER(s) = MIN,}
\end{aligned}
$$

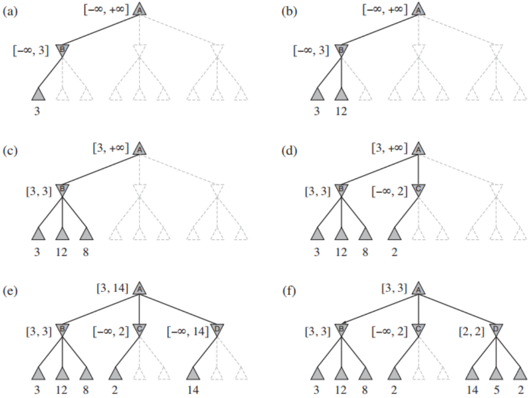


MINIMAX: Pomiar wydajności

- Wykorzystuje proste rekurencyjne obliczenie wartości minimax każdego następnika (potomnika) węzła. Rekurencja sięga aż do liści.
- Na przykład, w pokazanym drzewie gry najpierw pojawia się w lewym dolnym węźle z wartościami użytecznymi odpowiednio 3, 12 i 8. Następnie ten algorytm wybiera minimalną wartość 3, a wartość ta jest zapisana dla MAX jako najgorsza możliwa wartość użyteczna po wybraniu akcji a_1 (zobacz B). Po podobnym procesie algorytm przechowuje 2 jako wartości użyteczności dla obu akcji a_2, a_3 (zobacz C, D). Na koniec algorytm wybiera 3 jako maksimum najgorszych możliwych wartości użyteczności branych pod uwagę we wszystkich działaniach a_1, a_2, a_3.
- Algorytm wykonuje przeszukiwanie w głąb (depth-first search) drzewa gry. Tak więc, jeśli maksymalna głębokość wynosi m, i jest najwyżej b dopuszczalnych ruchów w każdym węźle, wówczas złożoność czasowa wynosi $O(b^m)$, a złożoność pamięciowa wynosi $O(bm)$.

Alpha-Beta Pruning

- Liczba węzłów, które należy sprawdzić w MINIMAX, jest wciąż wykładnicza względem głębokości drzewa gry. ‘Alpha-beta prunning’ wprowadza przeszukiwanie tylko efektywnych węzłów i odcina część drzewa.



(a) Na początku zakres wartości w węźle głównym A jest przechowywany jako $[-\infty, \infty]$. Aby sprawdzić wartość $MINIMAX(A)$, przeszukiwanie rozpoczyna się od lewej skrajnej gałęzi dla MAX.

(b, c) Następnie od lewej do prawej sprawdzane są wszystkie możliwe wartości dla działań b_1, b_2, b_3. Najgorsza możliwa wartość użyteczna dla MAX, po tym, jak MIN wybierze własne działanie, jest przechowywana w węźle B jako 3. Na tym etapie optymalny zakres wartości dla węzła A jest przechowywany jako $[3, \infty]$ a dla B to $[3, 3]$.

(d) Następnie rozpoczyna się przeszukiwanie od następnego węzła C, sprawdzanie od lewej skrajnej gałęzi c_1. Ponieważ wartość użyteczności 2 jest już mniejsza niż optymalna wartość użyteczności (3), dla MAX w węźle B, algorytm nie sprawdza pozostałych gałęzi C. Zatem w węźle C zakres optymalnych wartości użyteczności zostanie ustawiony jako $[-\infty, 2]$.

(e, f) Teraz przeszukiwanie przechodzi do D i rozpoczyna sprawdzanie od lewej do prawej wszystkich możliwych działań d_1, d_2, d_3. Tak jak poprzednio, wybiera najgorszą możliwą wartość, która wynosi 2, dla MAX. Dlatego optymalny zakres wartości użyteczności dla MAX przy D jest ustawiony jako $[2, 2]$.

- Wreszcie po sprawdzeniu wszystkich gałęzi a_1, a_2, a_3 w A maksymalny optymalny zakres wartości użyteczności dla MAX jest ustawiony jako $[3, 3]$. Więc $MINIMAX(A) = 3$.
- główny pomysł: Jeśli gracz ma lepszy wybór w którymkolwiek z nadrzędnych węzłów x lub wyższych, gałąź sięgająca x może zostać przycięta.
- α = najlepsza wartość (tj. najwyższa wartość) znaleziona do tej pory na ścieżce MAX
  β = najlepsza wartość (tj. najmniejsza wartość) znaleziona do tej pory na ścieżce MIN.
- Przeszukiwanie aktualizuje wartość $[\alpha, \beta]$ w każdym węźle, gdy porusza się wzdłuż ścieżek drzewa, i przycina pozostałe gałęzie w węźle, gdy tylko wiadomo, że wartość bieżącego węzła jest gorsza niż bieżące wartości α lub β.

Gry z niedoskonałą decyzją w czasie rzeczywistym: funkcja heurystyczna

- ‘Alpha-beta prunning’ wciąż musi sprawdzać węzły liści, a czasami ta głębokość stwarza problem, gdy ruchy muszą być wykonywane bardzo szybko.
- Claude Shannon (1950): ‘Programming a computer for playing chess’

> W tym artykule Shannon zasugerował, że zamiast szukać aż do węzłów końcowych, programy powinny zatrzymać się na wcześniejszym poziomie i zastosować funkcję oceny heurystycznej do stanów, aby sprawdzić skuteczność działania.

Aby nieco zmienić MINIMAX lub 'Alapha-Beta prunning'

- Zastąpienie funkcji Utility heurystyczną funkcją oceny EVAL, która szacuje użyteczność stanu.
- Zastąpienie TERMINAL-TEST testem odcięcia (CUT-OFF TEST), który decyduje, kiedy zastosować EVAL
- To daje nam następującą heurystyczną funkcję minimax dla stanu s i maksymalnej głębokości d.
- Za pomocą EVAL oszacujemy oczekiwaną użyteczność gry z danego stanu. Pomysł jest podobny do użycia funkcji heurystycznej do oszacowania odległości węzła docelowego w algorytmach przeszukiwania.

$$
\text{H-MINIMAX(s, d)} \begin{cases} = \text{EVAL(s)} & \text{jeśli CUTOFF-TEST(s, d),} \\ = max_{a \in ACTION(s)} H\text{-}MINIMAX(RESULT(s, a), d + 1) & \text{jeśli PLAYER(s) = MAX,} \\ = min_{a \in ACTION(s)} H\text{-}MINIMAX(RESULT(s, a), d + 1) & \text{jeśli PLAYER(s) = MIN,} \end{cases}
$$

Przykład heurystycznej funkcji oceny dla szachów

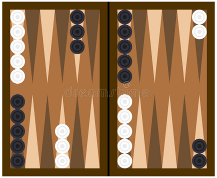
- Ponieważ przeszukiwanie musi zostać przerwane w stanie nieterminalnym, algorytmy muszą zgadywać o wyniku EVAL dla stanu końcowego. EVAL jest zaprojektowany w oparciu o różne funkcje gry.
- Na przykład, w książkach wprowadzających do gry w szachy mamy pewne wartości konkretne dla każdego rodzaju figur: Takie jak pion jest wart 1, skoczek lub goniec wart jest 3, jednego gońca wart jest 5, a królowa 9. Każdy typ odpowiada jednej kategorii. Zatem EVAL może być postaci:
  $EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  gdzie f_i to kategoria odpowiadająca każdemu typowi i w_i to liczba sztuk tego typu.
- Na przykład, jeśli w stanie mamy pięć pionków, jednego skoczka, jedną wieżę, jedenego gońca i jedną królową, wtedy
  $EVAL(s) = 5 \times 1 + 1 \times 3 + 1 \times 3 + 1 \times 5 + 1 \times 9 = 25$.

Głębokość na CUTOFF-TEST(s, x)

- Głębokość (d) odcięcia przeszukiwania można określić na podstawie średniej liczby ruchów, które można wybrać w określonym czasie. Czas ten można z góry ustalić na podstawie tego, ile czasu pozwalamy agentowi oprogramowania na wybór ruchu.
- Wobec tego CUTOFF-TEST(s, x) zwraca wartość prawda, jeśli głębokość x jest poniżej d, a zatem EVAL(s) należy obliczyć dla *s*.

Gry stochastyczne

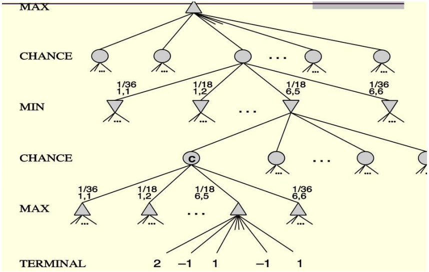
Backgammon to typowa gra, która łączy szansę i umiejętności.



- W grze gra kolejno dwóch graczy. W każdej turze gracz musi rzucić dwiema kostkami.
- Każdy gracz ma 15 warcabów (zwykle białe i czarne). Liczby pojawiające się w dwóch kostkach określają liczbę prawidłowych ruchów dla każdego gracza.
- Załóżmy, że białe wyrzucają 6-5 i mają cztery ruchy. Ale białe nie wiedzą, jakie liczby wyrzucają czarne, a zatem jakie będą dopuszczalne ruchy czarnych.
- Dlatego nie możemy zbudować standardowego drzewa gry jak dla gry w kółko i krzyżyk.

Drzewo gry dla blackgammona

- Drzewo gry dla gry Blackgammon zawiera losowe węzły oprócz węzłów dla MAX i MIN. Gałęzie z każdego węzła losowego oznaczają możliwe rzuty kostkami i są oznaczone rzutami i ich prawdopodobieństwami.
- Z 36 równie prawdopodobnych rzutów 21 jest różnych, ponieważ zasady dla 6-5 są takie same jak dla 5-6.
- 6 podwójnych ma prawdopodobieństwo $\frac{1}{36}$, a dla każdego z pozostałych 15 rzutów prawdopodobieństwo wynosi $\frac{2}{36}$ (tzn. $\frac{1}{18}$).



Zamiast MINIMAX mamy EXPECTMINIMAX

- W przypadku gier stochastycznych zamiast dokładnej wartości minimax wprowadzamy oczekiwaną wartość minimax.

EXPECTMINIMAX(s)

$$
\begin{aligned}
&= \text{UTILITY(s)} && \text{jeśli TERMINAL-TEST(s),}\\
&= max_{a \in ACTION(s)} EXPECTMINIMAX(RESULT(s, a)) && \text{jeśli PLAYER(s) = MAX,}\\
&= min_{a \in ACTION(s)} EXPECTMINIMAX(RESULT(s, a)) && \text{jeśli PLAYER(s) = MIN,}\\
&= \Sigma_r P(r) EXPECTMINIMAX(RESULT(s, r)) && \text{jeśli PLAYER(s) = CHANCE}
\end{aligned}
$$

gdzie r = możliwy rzut kostką (lub inne zdarzenie losowe) i $RESULT(s, r) = s$

Kilka słów o innych kryteriach optymalizacji

Przypomnijmy definicję gry omówioną w trzecim wykładzie

- Krotka (N, A, u) to gra w normalnej formie (skończona, n-osobowa) gdzie:
  N jest skończonym zbiórem graczy n, indeksowanych przez i;
  $A = A_1 \times A_2 \times \ldots \times A_n$, gdzie A_i to skończony zbiór działań dostępnych dla gracza i. Każdy wektor $a = (a_1, a_2, \ldots, a_n) \in A$ jest nazywany profilem działania.
  $u = (u_1, u_2, \ldots, u_n)$ gdzie każdy $u_i : A \mapsto R$ to funkcja użyteczności (lub funkcja wypłaty) dla każdego gracza i.
- W przypadku gry dwuosobowej $N = 2$.

Rozważmy tę macierz wypłat dla dwóch graczy, którzy mają dwie możliwe akcje C i D. Wiersz odpowiada graczowi 1, a kolumna odpowiada graczowi 2.

	C	D
C	(-1, -1)	(-4, 0)
D	(0, -4)	(-3, -3)

Liczba ujemna -k oznacza stratę k jednostek.

Optymalna strategia Pareto i najlepsza odpowiedź

(i) Pareto Optimal: Kryterium optymalności Pareto stanowi porządek określony na profilach użyteczności agentów. Na przykład, można powiedzieć, że wypłata (-1, -1) jest lepsza niż wypłata (-3, -3) dla obu agentów. Tak więc profil strategii (C, C) dominuje (D, D) w oparciu o kryterium optymalności pareto i Wtedy optymalna strategia to (C, C).

(ii) Best response: Gdy gracz 1 wybiera C, dla gracza 2 akcja D jest optymalna, a gdy gracz 1 wybiera D, akcja D jest ponownie optymalna dla gracza 2. Tak więc dla gracza 2 najlepszą odpowiedzią jest zawsze D. Za pomocą podobnego argumentu możemy sprawdzić, dla gracza 1 najlepszą odpowiedzią jest również D. Tak więc strategia (D, D) jest optymalna, biorąc pod uwagę najlepsze odpowiedzi obu graczy.

	C	*D*
C	(-1, -1)	(-4, 0)
D	(0, -4)	(-3, -3)

Problemy spełniania więzów (Constraint Satisfaction Problems (CSP))

- Podczas przeszukiwania algorytmów i gier rozważaliśmy atomową reprezentację każdego stanu.
- W CSP reprezentujemy każdy stan za pomocą zbioru zmiennych, z których każda ma wartość. Jak omówiliśmy w pierwszym wykładzie, jest to zatem kategoria rozproszonej reprezentacji środowiska zadań.
- CSP: CSP składa się z trzech komponentów X, D i C, gdzie
  - $X = \{X_1, X_2, \ldots, X_n\}$ jest zbiorem zmiennych,
  - $D = \{D_1, D_2, \ldots, D_n\}$ jest zbiorem składającym się z dziedzin dla każdej zmiennej, a
  - C jest zbiorem więzów (ograniczeń), które określają dopuszczalne kombinacje wartości dla zmiennych.
- Każda domena $D_i = \{v_{i1}, v_{i2}, \ldots, v_{ik}\}$ jest zbiorem wartości dla X_i.
- Każdy z więzów (ograniczenie) C_j może być reprezentowany jako para $\langle zakres, rel \rangle$ gdzie zakres jest krotką zmiennych, które dotyczą ograniczenia C_j, a rel jest relacją, która definiuje wartości które te zmienne mogą przyjąć.

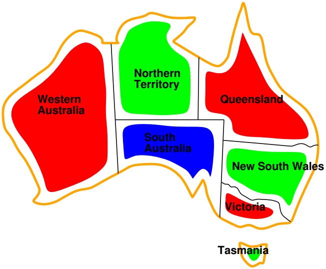
Rozwiązanie w CSP

- Przykład-1: Niech $X = \{X_1, X_2\}$, $D_1 = D_2 = \{A, B\}$, a wybrany z więzów (ograniczenie) to $X_1 \neq X_2$. To ograniczenie można przedstawić jako $\langle (X_1, X_2), \{(A, B), (B, A)\} \rangle$ lub $\langle (X_1, X_2), X_1 \neq X_2 \rangle$.
- Każdy stan w CSP jest definiowany przez przypisanie (assignment) wartości do niektórych lub wszystkich zmiennych, np. $\{X_i = v_i, X_j = v_j, \ldots, X_k = v_k\}$ gdzie $\{X_i, X_j, \ldots, X_k\} \subseteq X$.
- Przypisanie, które nie narusza żadnego ograniczenia, nazywane jest przypisaniem spójnym (consistent assignment).
- Pełne przypisanie (complete assignment) to takie, które przypisuje wartość każdej zmiennej.
- Rozwiązanie (solution) CSP to spójne, pełne przypisanie.
- Oba (A, B), (B, A) są rozwiązaniami dla przykładu-1.

Przykład-2: problem kolorowania mapy

- Rozważmy problem kolorowania mapy Australii. Australia ma następujące stany:
  Western Australia (WA), Northern Terriotory (NT), South Australia (SA), Queensland (Q), New South Wales (NSW), Victoria (V), Tasmania (T)
- Zadaniem jest pokolorowanie każdego regionu kolorym czerwonym, zielonym albo niebieskim w taki sposób, aby żadne sąsiednie regiony nie miały tego samego koloru.
- $X = \{WA, NT, SA, Q, NSW, V, T\}$,
  $D_1 = \ldots = D_7 = \{red, green, blue\}$ i
  $C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$.
- Ograniczenie $SA \neq WA$ można przedstawić jako $\{(red, green), (red, blue), (green, red), \ldots, (blue, green)\}$.

Jedno z rozwiązań



Rozwiązania są wartościowaniami spełniającymi wszystkie więzy, np.
$\{\mathrm{WA} = \mathrm{red}, \mathrm{NT} = \mathrm{green}, \mathrm{Q} = \mathrm{red}, \mathrm{NSW} = \mathrm{green}, \mathrm{V} = \mathrm{red}, \mathrm{SA} = \mathrm{blue}, \mathrm{T} = \mathrm{green}\}$

Rodzaje więzów

- Więzy unarne dotyczą pojedynczych zmiennych
  np. $SA \neq green$
- Więzy binarne dotyczą dwu zmiennych
  np. $SA \neq WA$
- Więzy wyższego rzędu dotyczą 3 lub więcej zmiennych,
  np. Sudoku (Zwykle uzywamy więź Alldiff, który określa, że wszystkie zmienne związane z ograniczeniem muszą mieć różne wartości.
- Preferencje (więzy nieostre)
  np. red jest lepszy niż green (często reprezentowane przez funkcję kosztu przypisania wartości do zmiennej)
  Rozważenie rozwiązań problemu spełniającego ograniczenia, a także preferencje należą do innej gałęzi, zwanej problemami optymalizacji ograniczeń (Constraint Optimization Problems)

Sudoku

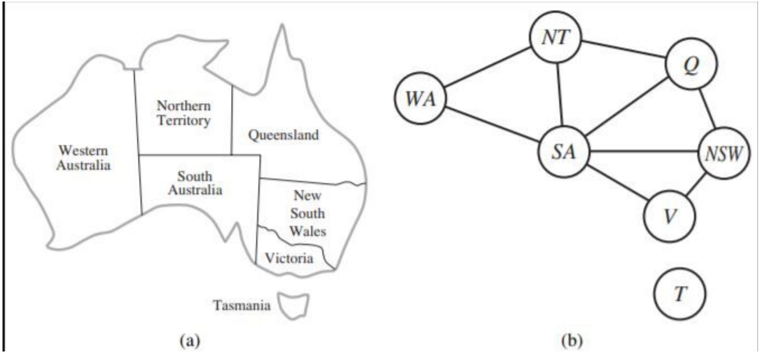
X_{11}	X_{12}	X_{13}		$\dots$			$\dots$	X_{19}
X_{21}	X_{22}	X_{23}		$\dots$			$\dots$	X_{29}
X_{31}	X_{32}	X_{33}		$\dots$			$\dots$	X_{19}
X_{71}	X_{72}	X_{73}		$\dots$			$\dots$	X_{79}
X_{81}	X_{82}	X_{83}		$\dots$			$\dots$	X_{89}
X_{91}	X_{92}	X_{93}		$\dots$			$\dots$	X_{99}

		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7								8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

- $X = \{X_{11}, X_{12}, \dots X_{99}\}$ i $D_i = \{1, 2, \dots, 9\}$
- Więzy:
  - W wierszu (rzędzie) wszystkie cyfry są różne
    $Alldiff(X_{11}, X_{12}, X_{13}, \dots, X_{19})$
    $\vdots$
  - W kolumnie wszystkie cyfry są różne
    $Alldiff(X_{11}, X_{21}, X_{31}, \dots, X_{91})$
    $\vdots$
  - W kwadracie 3×3 wszystkie cyfry są różne
    $Alldiff(X_{11}, X_{12}, X_{13}, X_{21}, X_{22}, X_{23}, X_{31}, X_{32}, X_{33})$
    $\vdots$

CSP-Graf: mapa Australii

- Jak możemy znaleźć rozwiązanie, biorąc pod uwagę CSP przy użyciu agenta oprogramowania?
- Pierwszym krokiem jest to, że możemy przedstawić CSP jako graf.



- Węzły grafu odpowiadają zmiennym CSP
- Krawędź łączy dwie zmienne, jeśli uczestniczą w ograniczeniu

Sprawdzanie lokalnej spójności

- Aby stworzyć algorytm przeszukiwania rozwiązania dla CSP, oprócz grafu reprezentującego przestrzeń stanu problemu musimy dodać metodę sprawdzania spójności lokalnej.
- W algorytmach CSP występują dwie części;
  - (i) jedna polega na poszukiwaniu przypisania z kilku możliwych przypisań dla zmiennych lub
  - (ii) za pomocą propagacji więzów algorytm ustala, które wartości zmiennej (połączonej z bieżącą zmienną) należy uznać za dopuszczalne (tj. które wartości należy wykluczyć z rozważania)
- Tak więc potrzebne jest sprawdzanie lokalnej spójności. Rozpoznając CSP jako graf, metoda sprawdzania lokalnej spójności wymusza eliminację niespójnych wartości na rozpatrywanej części grafu.

Dziękuję za uwagę