Sztuczna inteligencja. Dekompozycja więzów oraz lokalne szukanie rozwiązań

Paweł Rychlikowski

Instytut Informatyki UWr

30 marca 2018

Korzyści z reifikacji

Przypomnienie

Reifikacja – wprowadzanie więzów typu $X \Leftrightarrow c(Y_1, \ldots, Y_n)$.

- Możemy ponazywać sobie wszystkie więzy.
- A następnie kontrolować liczbę spełnionych więzów (dodając warunki: co najmniej 1, co najwyżej 5, więcej tych, niż tych, itd)

Uwaga

W niektórych systemach są na to osobne więzy globalne, alteast, almost, exactly

Struktura problemu

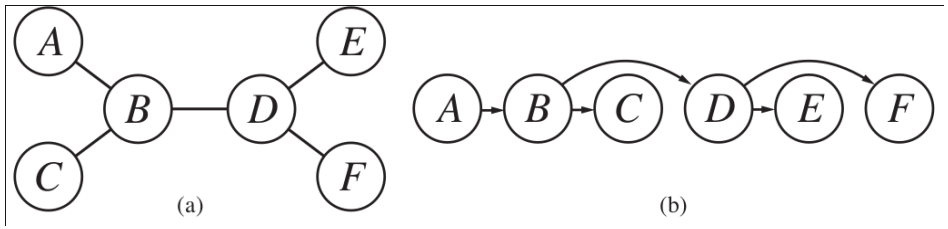
Uwaga

Patrząc na graf więzów, możemy zauważyć pewne właściwości. Na przykład podzielić więzy graf na spójne składowe i rozwiązać je osobno (Tasmania!).

Uwaga 2

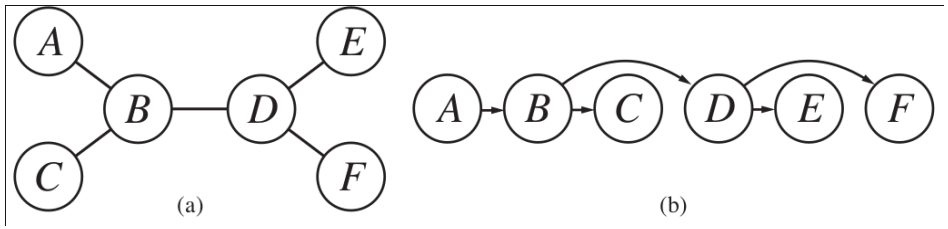
Inną ważną klasą grafów są drzewa. CSP o strukturze drzewiastej da się łatwo rozwiązać. Jak?

Drzewiaste CSP



1. Sortujemy topologicznie drzewo
2. Osiągamy spójność łukową (algorytm AC-3)
3. Rozwiązujemy szybko taką sieć więzów, zaczynając od korzenia.

Drzewiaste CSP



Algorytm

Trywialny: idziemy od lewej do prawej, wybieramy dowolne wartości z dostępnych w danym momencie.

Pamiętajmy o gwarancji, jaką daje AC-3!

Udrzewianie

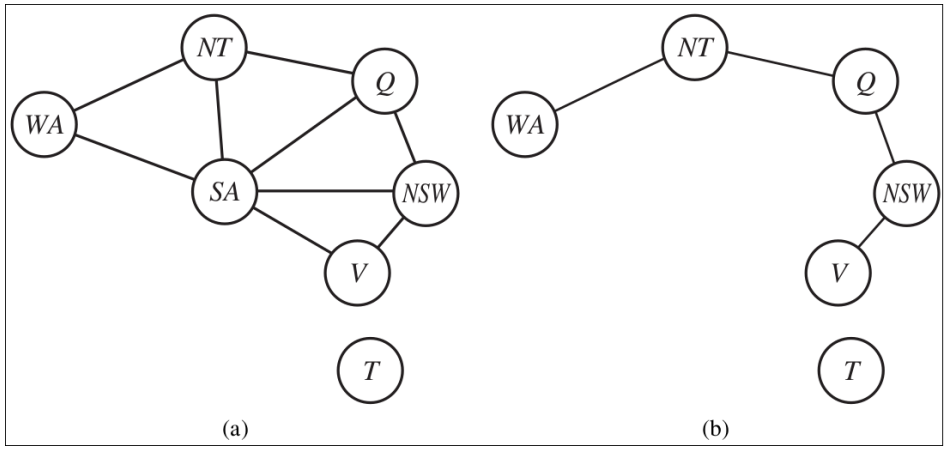
- Cieszymy się z algorytmu na drzewach, ale jak sprawić, by CSP stało się drzewem?
- I czy zawsze się da?

Uwaga

Pamiętajmy, że CSP jest NP-zupełne, a nie spodziewamy się wielomianowego algorytmu dla takich problemów!

Udrzewianie (2)

Można usunąć jakiś węzeł (węzły)



Popatrzmy na Australię:

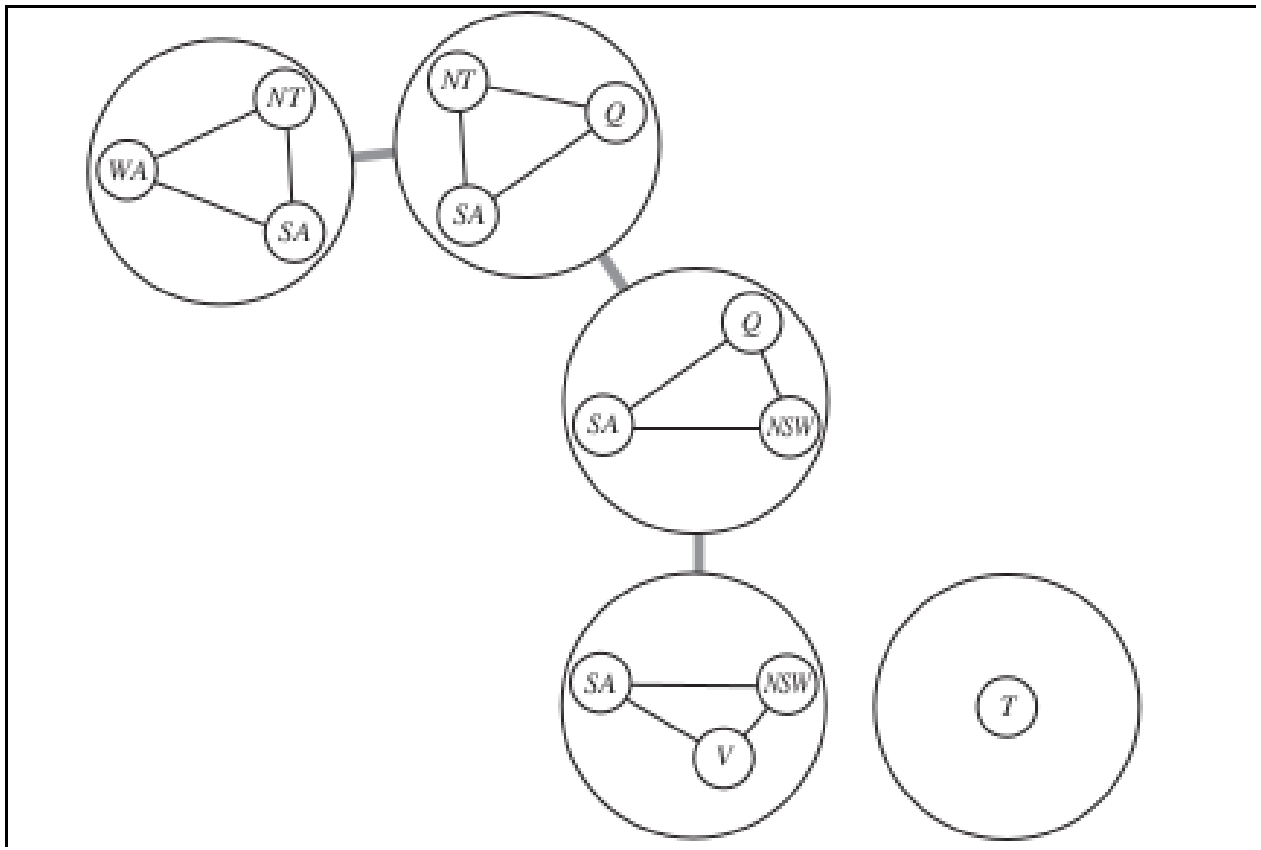
- Usunięcie węzła to przypisanie wartości zmiennej (i sprawdzenie innych wartości w kolejnych nawrotach)
- Problem cycle cutset
- Używane również w przetwarzaniu sieci bayesowkich

Cutset conditioning

- Cutset conditioning jest metodą wyznaczania zbioru udrzewiającego (od którego zaczniemy przypisywanie)
- Powiedzmy, że szukamy zbioru o wielkości c. Metoda jest następująca:
  1. Wybieramy zbiór zmiennych, które udrzewią problem (czyli po usunięciu graf jest drzewem).
  2. Dla każdego podstawienia, spełniającego więzy z wybranymi zmiennymi usuń w pozostałych dziedzinach wartości niezgodne z wybramym podstawieniem
  3. Rozwiąż pozostały problem (drzewiasty)
- Daje to wielomianowy algorytm dla całego zbioru więzów (przy założeniu stałego c).

Dekompozycja drzewa

- Dzielimy problem na podproblemy, rozwiązywane osobno
- Trywialny przykład: Australia i Tasmania
- Mniej trywialna dekompozycja:



Warunki dekompozycji drzewa

Dzielimy problem na osobne (potencjalnie powiązane) podproblemy.

Warunki

1. Każda zmienna jest w co najmniej jednym podproblemie.
2. Nie gubimy więzów: jeżeli dwie zmienne występują w jednym więzie, wówczas muszą znajdować się (z tym więzem) w jakimś podproblemie
3. Jeżeli zmienna występuje w dwóch podproblemach (A i B), to występuje w każdym pobproblemie na ścieżce pomiędzy.

Dekompozycja drzewa (2)

- Oczywiście interesuje nas możliwie najmniejsza wielkość pobproblemów.
- Można testować różnego rodzaju algorytmy zachłanne, czy heurystyczne.

Pętla: obrazki logiczne

- Jedno z pierwszych zadań na naszej pracowni to były obrazki logiczne (inspirowane algorytmem WalkSat)
- Spróbujemy uogólnić sobie te idee na dowolne CSP.

Przeszukiwanie lokalne dla CSP

- Przeszukiwanie lokalne nie próbuje systematycznie przeglądać przestrzeni rozwiązań (ogólniej: przestrzeni stanów)
- Zamiast tego pamięta jeden stan (lub niewielką, stałą liczbę stanów)
- Dla CSP stanem będzie kompletne przypisanie (niekoniecznie spełniające więzy).

Problemy optymalizacyjne

- W tych problemach szukamy stanu, który maksymalizuje wartość pewnej funkcji (jakość planu).
- Często problemy z twardymi warunkami da się zamienić na problemy optymalizacyjne. Jak?

Można policzyć liczbę złych wierszy (kolumn) w obrazkach logicznych, albo liczbę szachów w hetmanach, albo....

MinConflicts

Uwaga

Możemy myśleć o spełnianiu CSP jako o zadaniu maksymalizacji liczby spełnionych więzów.

Możemy zatem stworzyć algorytm, w którym:

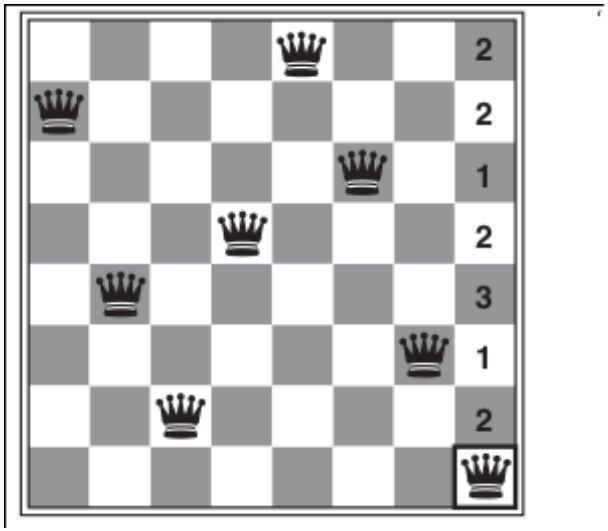
- Zmieniamy tę zmienną, która powoduje niespełnienie największej liczby więzów.
- Wybieramy dla niej wartość, która owocuje najmniejszą liczbą konfliktów.

Przykład: 8 hetmanów

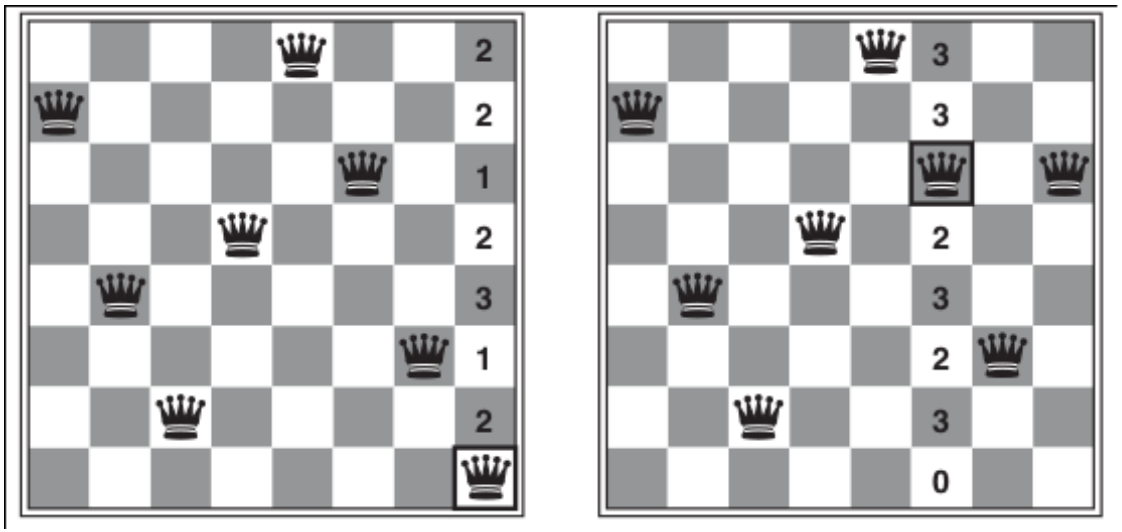
- Jak wybrać stan? (Wskazówka: powinniśmy umieć łatwo przejść ze stanu do stanu)
- Stan: w każdej kolumnie 1 hetman, Ruch: przesunięcie hetmana w górę lub w dół

Popatrzmy, jak działa min-conflicts dla hetmanów.

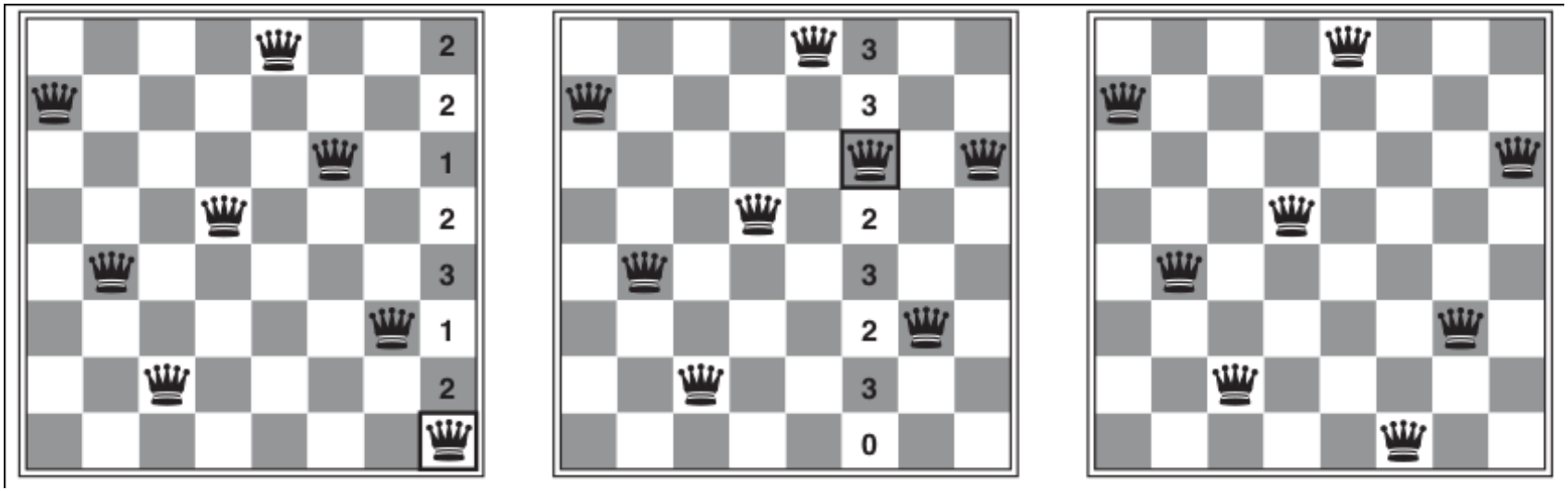
Min-conflicts dla hetmanów



Min-conflicts dla hetmanów



Min-conflicts dla hetmanów



Hetmany. Wyniki

- Dla planszy 8×8 osiąga sukces w 14% przypadków.
- Niby niezbyt dużo, ale możemy go uruchomić na przykład 20 razy, wówczas p-stwo sukcesu to ponad 95%.
- Można dopuszczać pewną liczbę ruchów w bok (czyli, że nie możemy poprawić, ale możemy nie pogorszyć, jak na obrazkach).
- Jak dopuścimy ruchy w bok , to wówczas mamy sukces w 94%

Ważenie więzów

- Każdy więz ma wagę, początkowo wszystkie równe na przykład 1
- Waga więzów niespełnionych cały czas troszkę rośnie.
- Chcemy naprawiać nie zbiór więzów o liczności n, ale raczej zbiór więzów o największej sumarycznej wadze

Więzy trudne, rzadko spełniane będą miały coraz większy priorytet.

Więzy on-line

- Wyobraźmy sobie, że mamy problem, który się zmienia (ale w niewielkim stopniu)
- Przykład: obsługa linii lotniczych – bo zamykają się lotniska, pilot może złapać grypę, ...

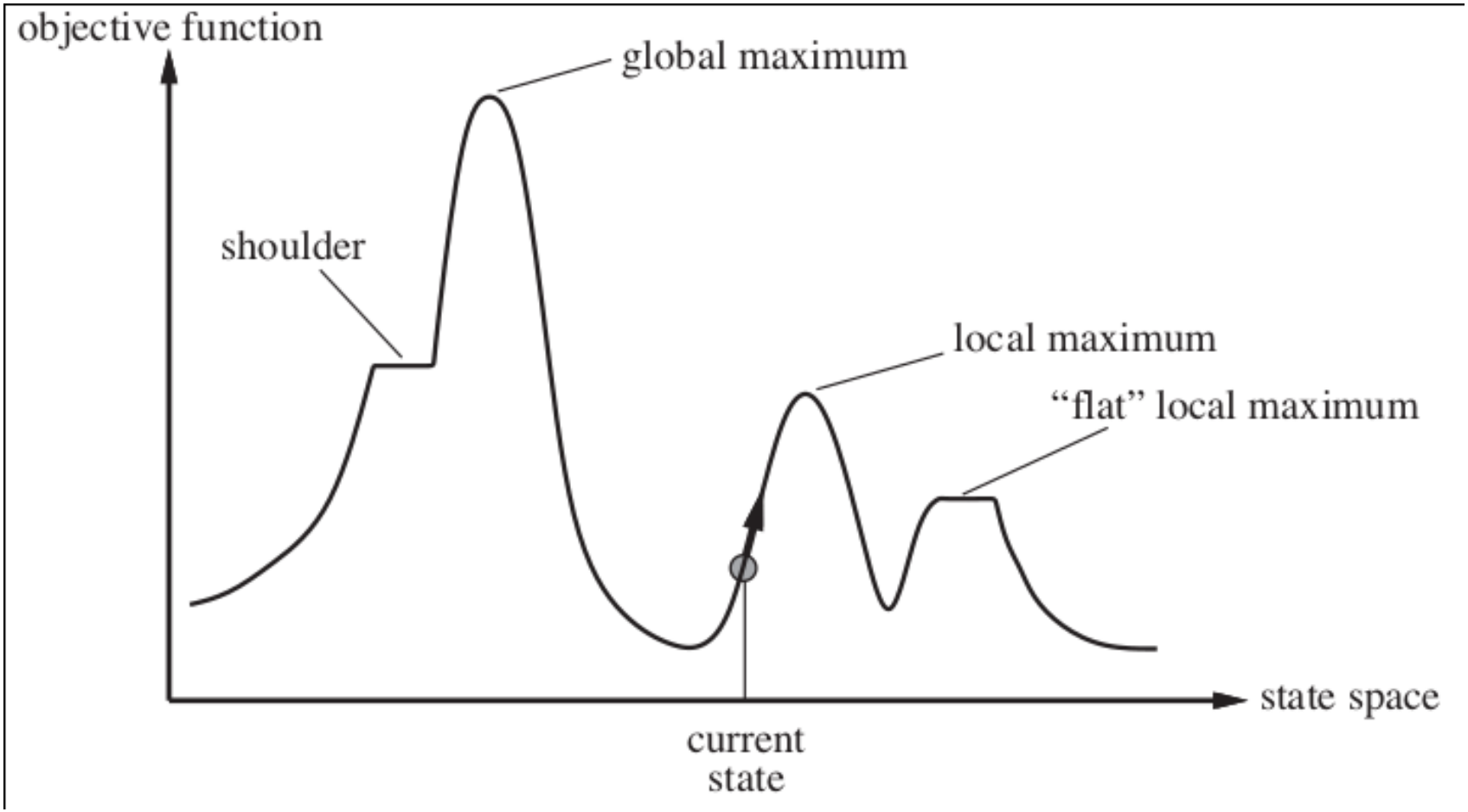
On-line CSP

Min-conflicts umożliwia rozwiązywanie tego typu zadań: stan początkowy to ostatnie dobre przypisanie.

Przeszukiwania lokalne (ogólnie)

- Powiemy sobie jeszcze o paru ideach związanych z przeszukiwaniem lokalnym.
- Można je wykorzystywać w zadaniach więzowych, ale nie tylko.

Krajobraz przeszukiwania lokalnego



Hill climbing

Hill climbing jest chyba najbardziej naturalnym algorytmem inspirowanym poprzednim rysunkiem.

- Dla stanu znajdujemy wszystkie następniki i wybieramy ten, który ma największą wartość.
- Powtarzamy aż do momentu, w którym nie możemy nic poprawić

Problem

Oczywiście możemy utknąć w lokalnym maksimum.

Hill climbing z losowymi restartami

Uwaga

Możemy podjąć dwa działania, oba testowaliśmy w obrazkach logicznych:

1. Dorzucać ruchy niekoniecznie poprawiające (losowe, ruchy w bok)
2. Gdy nie osiągamy rozwiązania przez dłuższy czas rozpoczynamy od początku.

Hill climbing + random restarts (w trywialny sposób) jest algorytmem zupełnym z p-stwem 1 (bo kiedyś wylosujemy układ startowy)

Inne warianty Hill climbing

a) Stochastic hill climbing – wybieramy losowo ruchy w górę (p-stwo stałe, albo zależne od wielkości skoku).
b) First choice hill climbing – losujemy następnika tak długo, aż będzie on ruchem w górę
   - dobre, jeżeli następników jest bardzo dużo

Uwaga

Idee z tego i kolejnych algorytmów można dowolnie mieszać – na pewno coś wyjdzie!

Symulowane wyżarzanie

- Motywacja fizyczna: ustalanie struktury krystalicznej metalu.
- Jeżeli będziemy ochładzać powoli, to metal będzie silniejszy (bliżej globalnego minimum energetycznego).
- **Symulowane** wyżarzanie – próba oddania tej idei w algorytmie.

Algorytm

Symulujemy opadającą temperaturę, prawdopodobieństwo ruchu chaotycznego zależy malejąco od temperatury.

Symulowane wyżarzanie (2)

- Przykładowa implementacja bazuje na first choice hill climbing.
- Jak wylosowany jest lepszy $\Delta F > 0$, to do niego przechodzimy (maksymalizacja F).
- W przeciwnym przypadku wykonujemy ruch z p-stwem $p = e^{\frac{\Delta F}{T}}$
- Pilnujemy, żeby T zmniejszało się w trakcie działania (i było cały czas dodatnie)

Komentarze do wzoru

- $\Delta F \leq 0$, $T > 0$, czyli $0 \leq p \leq 1$.
- Im większe pogorszenie, tym mniejsze p-stwo
- Im większa temperatura, tym większe p-stwo.

Taboo search

Problem

Być może płaskie maksimum lokalne.

Rozwiązanie

Dodajemy pamięć algorytmowi, zabraniamy powtarzania ostatnio odwiedzanych stanów.

Local beam search

- Zamiast pamiętać pojedynczy stan, pamiętamy ich k (wiązkę).
- Generujemy następniki dla każdego z k stanów.
- Pozostawiamy k liderów.

Uwaga 1

To nie to samo co k równoległych wątków hill-climbing (bo uwaga algorytmu może przerzucać się do bardziej obiecujących kawałków przestrzeni)

Uwaga 2

Beam search jest bardzo popularnym algorytmem w różnych zadaniach wykorzystujących sieci neuronowe do modelowania sekwencji (np. tłumaczenie maszynowe).

Algorytmy ewolucyjne

- Zarządzamy populacją osobników (czyli np. pseudorozwiązań jakiegoś problemu więzowego).
- Mamy dwa rodzaje operatorów:
  a) Mutacja, która z jednego osobnika robi innego, podobnego.
  b) Krzyżowanie, która z dwóch osobników robi jednego, w jakiś sposób podobnego do „rodziców".
- Nowe osobniki oceniane są ze względu na wartość funkcji przystosowania
- Przeżywa k najlepszych.

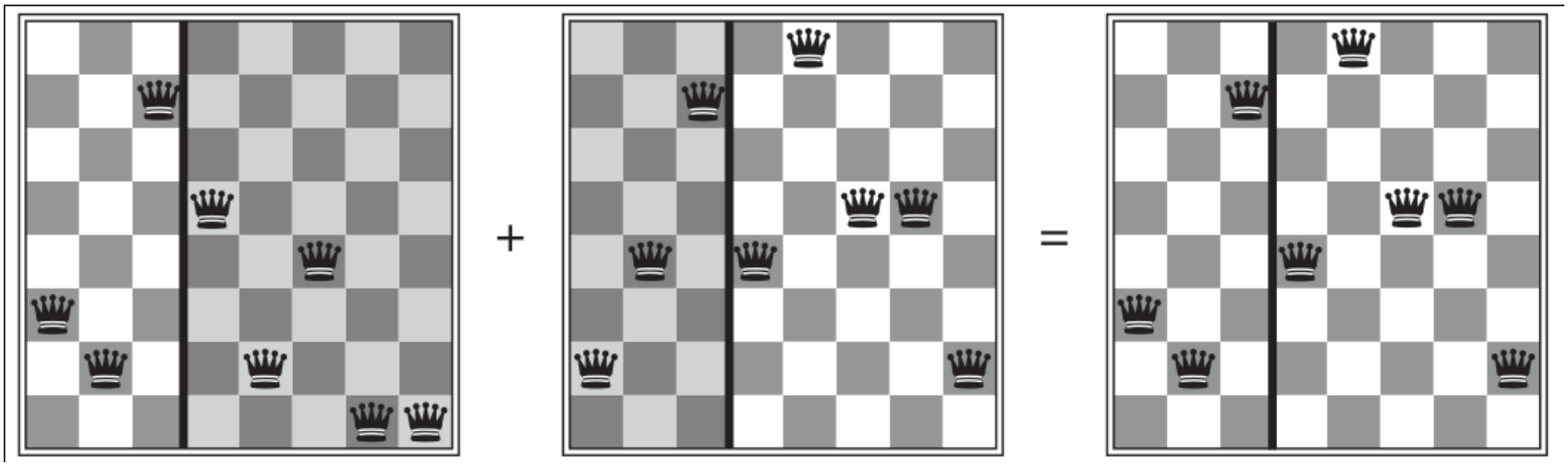
Uwaga

Zauważmy, że choć zmienił się język, jeżeli pominiemy krzyżowanie, to otrzymamy wariant Local beam search (mutacja jako krok w przestrzeni stanów).

Krzyżowanie. Przykład

Pytanie

Czym mogłoby być krzyżowanie dla zadania z N hetmanami?



Algorytmy ewolucyjne. Kilka uwag

1. Krzyżowanie i mutacje można zorganizować tak, że najpierw powstają dzieci, a następnie się mutują z pewnym prawdopodobieństwem.
2. Wybór osobników do rozmnażania może zależeć od funkcji dopasowania (większe szanse na reprodukcję mają lepsze osobniki)
3. Można mieć wiele operatorów krzyżowania i mutacji.