»Лекционен курс

»Изкуствен интелект

Проблеми с ограничения

Virtual learning space

Увод

- Факторизирано представяне на състоянията
- Всяко състояние
  - Множество от променливи
  - Всяка променлива има някаква стойност
- Един проблем е решен, когато всяка променлива има една стойност, която удовлетворява някакви ограничения
  - Нарича се проблем, удовлетворяващ ограничения (Constraint Satisfaction Problem, CSP)
- Алгоритмите за търсене на CSP се възползват от структурата на състоянията и използват общи, а не специфични за домейна евристики, за да позволят решаването на сложни проблеми

Дефиниция

- Проблем с ограничения (Constraint Satisfaction Problem, CSP) се състои от следните три компонента:
  - $X = \{ X_1, . . . , X_n\}$ – множество на променливи
  - $D = \{ D_1, . . . , D_n\}$ – множество домейни, за всяка променлива
    - Всеки домейн D_i се състои от едно множество от разрешени стойности $\{v_1, . . . , v_k\}$ за променливата X_i
  - C - множество ограничения, които специфицират разрешените комбинации от стойности

Ограничения

- Всяко ограничение C_i се състои от двойката (scope, rel), където:
  - scope - вектор от променливи, които участват в ограничението
  - rel – релация, която дефинира стойностите, които могат да заемат тези променливи
- Релацията може да бъде представена по два начина
- Като явен списък от стойности, които удовлетворяват ограничението
- Като абстрактна релация, поддържаща два оператора:
  - Тест – дали един вектор е член на релацията
  - Изброяване – членовете на релацията

Пример

Как може да се представи?

» Пример:

> Ако X_1 и X_2 имат един и същ домейн {A,B}, тогава ограничението, че двете променливи трябва да имат различни стойности …

Пример

Как може да се представи?

- Пример:
  - Ако X_1 и X_2 имат един и същ домейн {A,B}, тогава ограничението, че двете променливи трябва да имат различни стойности …

$<(X_1, X_2), [(A, B), (B, A)]>$

явно

$<(X_1, X_2), X_1 \neq X_2)>$

неявно

Решаване на проблеми с ограничения

- За решаване на CSP трябва да дефинираме
  - Пространство на състояния
  - Представяне на решението
- Всяко състояние в CSP е дефинирано чрез едно присвояване на стойности на някои или всички променливи, т.е. $\{ X_i = v_i, X_j = v_j, \ldots\}$
- Дефиниции:
  - Консистентно (легално) присвояване – всяко присвояване, което не нарушава някое ограничение
  - Пълно присвояване – всяка променлива има присвояване
  - Частично присвояване – само някои променливи имат присвояване
  - Решение на един CSP - **?**

Консистентно пълно присвояване

Пример за CSP: оцветяване на карти

- Как като CSP?
- Как можем да визуализираме един CSP (като абстрактна структура)?

Пример за CSP: оцветяване на карти

Как като CSP?

Множество променливи:

X = { WA, NT, Q, NSW, V, SA,T }

Пример за CSP: оцветяване на карти

Как като CSP?

Множество променливи:

X = { WA, NT, Q, NSW, V, SA,T }

Домейн за всяка променлива:

D_i = { red, green, blue }

Пример за CSP: оцветяване на карти

Как като CSP?

Множество променливи:
X = { WA, NT, Q, NSW, V, SA,T }

Домейн за всяка променлива:
D_i = { red, green, blue }

Ограничения (9):
C = { SA ≠ WA, SA ≠ NT, SA ≠ Q, SA ≠ NSW, SA ≠ V, WA ≠ NT, NT ≠ Q, Q ≠ NSW, NSW ≠ V }

SA ≠ WA е съкращение на <(SA, WA), SA ≠ WA>, където SA ≠ WA може да бъде пълно изброено като { (red, green), (red, blue) , (green, red), (green, blue), (blue, red), (blue, green) }

Съществуват повече възможни решения на този проблем, като напр.
{ WA = red, NT= green, Q = red, NSW = green, V = red, SA= blue, T = red }

Пример за CSP: оцветяване на карти

Как можем да визуализираме един CSP (като абстрактна структура)?

Пример за CSP: оцветяване на карти

Как можем да визуализираме един CSP (като абстрактна структура)?

Целесъобразно CSPs да бъдат визуализирани като графи на ограничения:

- **Възли** – кореспондират с променливите на проблема
- **Ребра** – свързват две променливи, участващи в едно ограничение

Оценка на CSP

- Защо формулираме някои проблеми като CSP?
  - CSPs са естествено представяне на широк клас проблеми
  - Ако имаме система за решаване CSP, често е по-лесно да решим един проблем, използвайки я, вместо да търсим решение с друг метод за търсене

Оценка на CSP

- CSP машините могат да бъдат по-бързи от тези, използващи търсене в класически ПС
  - Понеже CSP машините могат по-бързо да елиминират големи области
  - Напр., щом сме избрали { SA= blue } в проблема Австралия, можем да заключим, че нито една от петте съседни променливи може да бъде „синя“
- Без предимството на разпространяване на ограниченията един метод за търсене би трябвало да разгледа $3^5 = 243$ присвоявания за 5 съседни променливи
  - С разпространяване на ограниченията никога няма да разглеждаме „blue“ като стойност, така че имаме за разглеждане само $2^5 = 32$ присвоявания, т.е. 87% редуциране

Сравнителна Оценка CSP & ПС

- В класическите ПС
  - Обикновено систематично проверяваме дали достигнатите състояния са целеви
- В CSPs
  - След като установим, че едно частично присвояване не е решение, можем незабавно да изоставим по-нататъшна детайлизация на присвояването
  - Освен това, можем да видим защо присвояването не е решение
  - Виждаме, които променливи нарушават едно ограничение
  - Така можем да фокусираме вниманието си върху значимите променливи
- Като резултат, много проблеми, неподатливи за решаване в класическите ПС проблеми, могат бързо да бъдат решени, когато са формулирани като CSP

Видове CSPs

- За представяне на CSPs е съществено да се изследват различните видове домейни и ограничения
- Видове домейни
  - Дискретни, крайни домейни
  - Дискретни, безкрайни домейни
  - Непрекъснати домейни
- Видове ограничения
  - Унарни ограничения
  - Бинарни ограничения
  - N-арни ограничения
  - Глобални ограничения

Дискретни, крайни домейни

- Най-простият вид CSP
- Включва променливи, които имат дискретни, крайни домейни
- Примери:
  - Оцветяване на карти
  - 8-те царици
    - $Q_1, \ldots, Q_s$ - позиции на всяка от цариците в колоните 1, . . . , 8
    - Всяка променлива има домейн $D_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$

Дискретни, безкрайни домейни

- Един дискретен домейн може да бъде безкраен
- Примери:
  - Множеството на целите числа или на символните низове
  - Ако не дадем краен срок за планиране на задачи, може да съществува безкрайно множество от стартови времена за всяка променлива
- При безкрайните домейни не е възможно описания на ограниченията чрез изброяване на всички възможни комбинации от стойности
  - Вместо това, трябва да се използва език за задаване на ограничения
  - Напр., $T_1 + d_1 \leq T2$ (вместо изброяване на възможните двойки)
- Специални алгоритми за линейни и нелинейни ограничения и целочислени променливи

Непрекъснати домейни

- CSPs с непрекъснати домейни са типични за реалните числа и обикновено се изучават в областта на изследване на операциите
- Най-добре позната категория на непрекъснати CSPs – линейното програмиране
  - Могат да бъдат решавани във време, полиномно на броя на броя на променливите
- Освен тях
  - Квадратично програмиране
  - Конично програмиране от втори ред

Унарни ограничения

- Най-простият вид ограничения
- Ограничават стойността на една отделна променлива
- Напр., оцветяване на карти
  - Ако искаме да представим, че южно-австралийците не искат да толерират зеления цвят, можем да използваме унарното ограничение <(SA), SA ≠ green>

Бинарни ограничения

- Едно бинарно ограничение свързва две променливи
  - Напр., SA $\neq$ NSW
- Един бинарен CSP е само с бинарни ограничения
  - Обикновено се представя като граф на ограничения (примера)
- Можем да представяме също ограничения от по-висок ред (n-арни ограничения)
  - Напр., твърдението, че стойността на Y е между X и Z, с троичното ограничение Between(X, Y, Z).

Глобални ограничения

- Едно ограничение, включващо произволен брой променливи се нарича глобално ограничение
  - Не е необходимо да включва всичките променливи на един проблем
- Едно от най-разпространените глобални ограничения е Alldiff
  - Всичките променливи, включени в ограничението, трябва да имат различни стойности
  - Напр., играта „Судоку“
    - Всичките променливи в един ред, колона и каре трябва да удовлетворяват Alldiff ограничението

Пример: Крипто-аритметична задача

Глобални ограничения

Alldiff (F, T, U, W, R, 0)

	T	W	O
+	T	W	O
F	O	U	R

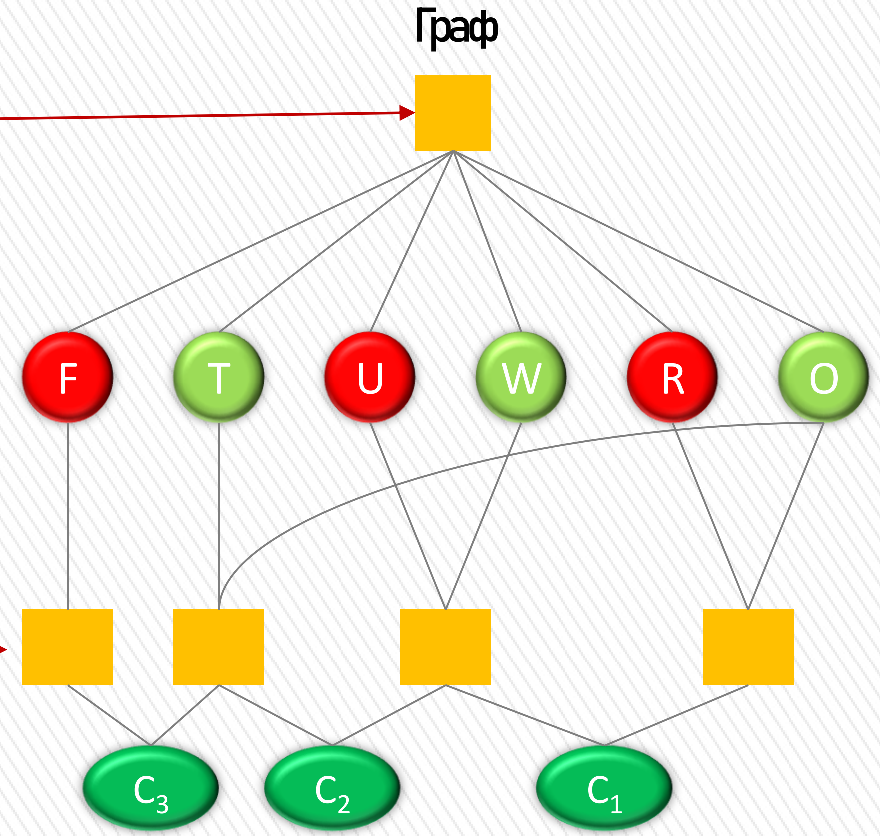
Допълнителни ограничения

$O + O = R + 10 \cdot C_{10}$

$C_{10} + W + W = U + 10 \cdot C_{100}$

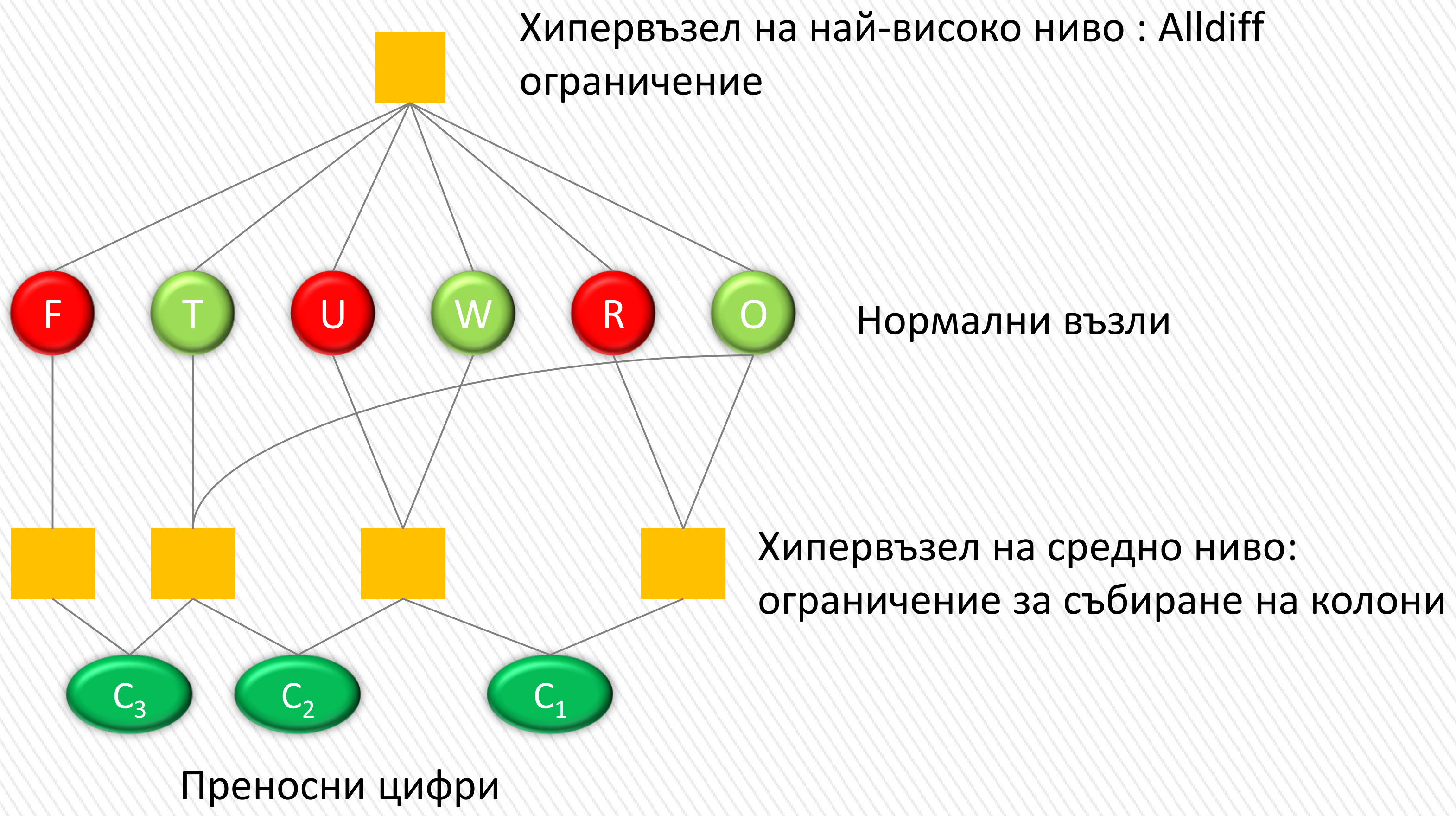
$C_{100} + T + T = O + 10 \cdot C_{1000}$

$C_{1000} = F$



C_{10}, C_{100}, C_{1000} помощни променливи представящи цифрите на преноса в десетичната, стотната и хилядната позиции

Пример: Крипто-аритметична задача



Правене заключения в CSP

- В обикновените ПС:
  - Един алгоритъм може да прави само едно нещо - търсене
- В CSPs съществува избор
  - Един алгоритъм може да
    - Търси – избира ново присвояване на променлива от няколко възможности или
    - Прави специфичен вид извод, нарече разпространение на ограничението – използва ограниченията за редуциране броя на допустимите стойности на една променлива, което от своя страна може да редуцира допустимите стойности за друга променлива и т.н.
- Разпространяване на ограниченията може да бъде комбинирано с търсене или може да се извърши като предварителна стъпка, преди да започне търсенето

Правене заключения в CSP

- Понякога този препроцесинг може да реши цялостния проблем, така че да не е необходимо последващо търсене
- Основна идея
  - **Локална консистентност**
- Ако разглеждаме всяко променлива като възел от един граф и всяко бинарно ограничение като едно ребро
  - Тогава процесът за прилагане на локална консистентност във всяка част на графа причинява това, че неконсистентните стойности се елиминират навсякъде в графа
- Съществуват различни типове локална консистентност

Локална консистентност

» Съществуват различни типове локална консистентност:
  > Консистентност на възли (КВ)
  > Консистентност на ребра (КР)
  > Консистентност на пътища (КП)
  > к-Консистентност

Консистентни възли

- Една отделна променлива (кореспондира с възел в CSP) е консистентен възел, ако всички стойности в нейния домейн удовлетворяват унарните ограничения за променливата
- За примера: жителите на South Australian (SA) не обичат зелен цвят
  - Първоначално променливата SA има {red, green, blue}
  - Можем да направим възела консистентен чрез елиминиране на green, оставяйки SA с редуциран домейн {red, blue}
- Един граф е с консистентни възли ако всеки възел (променлива) на графа е консистентен
- Принципно, винаги е възможно да се елиминират всички унарни ограничения в един CSP чрез прилагане на консистентност на възли

Консистентни ребра

» Една променлива в един CSP е консистентна към ребра, ако всяка стойност от нейния домейн удовлетворява бинарните ограничения на променливата

» Формално, X_i е консистентна към ребра по отношение на друга променлива X_j

  > Ако за всяка стойност в домейна D_i съществува някаква стойност в домейна D_j, която удовлетворява бинарното ограничение върху реброто (X_i , X_j)

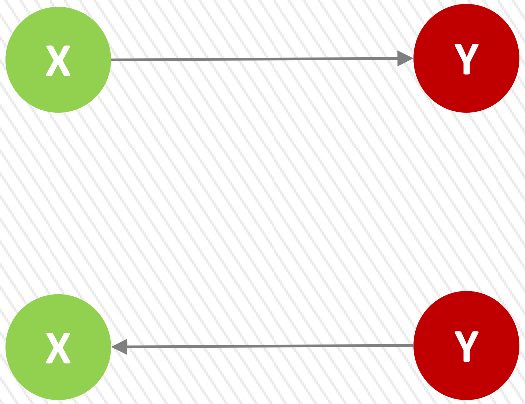
» Един граф е консистентен към ребра, ако всеки възел (променлива) е консистентен към ребра по отношение на всяка от останалите променливи

Пример 1

D_X = { множество на цифрите}

D_Y = { множество на цифрите}

Ограничение: $Y = X^2$



Как явно представяне на ограничението?

Как X КР спрямо Y?

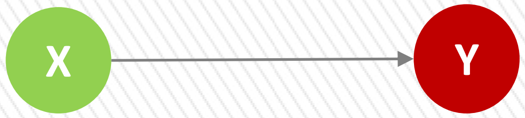
Как Y КР спрямо X?

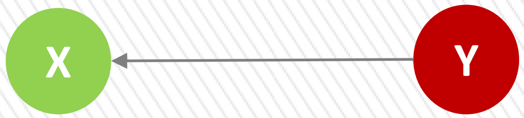
Пример 1

D_X = { множество на цифрите}

D_Y = { множество на цифрите}

Ограничение: $Y = X^2$





Как явно представяне на ограничението?

((X, Y), {(0, 0), (1, 1), (2, 4), (3, 9)}})

Как X КР спрямо Y?

Редукция: D_X = { 0, 1, 2, 3}

Как Y КР спрямо X?

Редукция: D_Y = { 0, 1, 4, 9}

Пример 2

» За картата на Австралия
  > Консистентността към ребра не може да помогне за решаване на проблема
» Да разгледаме следното ограничение за (SA, WA): {(red, green) , (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}
  > Без значение каква стойност избираме за SA (или за WA), съществува валидна стойност за другата променлива
  > Така, прилагането на консистентност към ребра няма ефект върху домейните на всяка от променливите
» Също е възможно всички n-арни ограничения могат да бъдат трансформирани в бинарни
  > Поради това, обикновено CSP машините работят само с бинарни ограничения

Методи

» AC-3 алгоритъм

» Търсене с възврат

» Локално търсене

AC-3 алгоритъм: псевдокод

function **AC-3**(*csp*) **return** false ако е намерена неконсистеност, true – в противен случай
  inputs: *csp*, бинарен CSP с компоненти (X, D, C)
  local variables: *queue*, опашка от ребра, в началото всички ребра в *csp*

  while *queue* **is** not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REVISE(*csp*, X_i, X_j) **then**
      if size of D_i = 0 **then return** *false*
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add (X_k, X_j) to *queue*
return *true*

function **REVISE**(csp, X_i, X_j) **return** true ако и само ако ревизираме домейна на X_i
  revised $\leftarrow$ *false*
  for each *x* **in** D_i **do**
    if no value *y* **in** D_j allows *(x,y)* to satisfy the constraint between X_i and X_j **then**
      delete *x* from D_i
      revised $\leftarrow$ *true*
return *revised*

AC-3 алгоритъм

» Най-известният алгоритъм за консистентност на ребра е AC-3

- За да направи всяка променлива консистентна към ребра, алгоритъмът поддържа една опашка на ребрата, които ще бъдат разглеждани
  - В действителност, редът на разглеждането не е съществен, така че структурата реално е множество
- В началото, опашката съдържа всички ребра на един CSP
- AC-3 взема произволно ребро (X_i , X_j) от опашката и прави X_i консистентен към ребра по отношение на X_j
  - Ако това остави D_i непроменен, алгоритъмът отива към следващото ребро
  - Ако това промени D_i (прави домейна по-малък), тогава добавяме към опашката всички възли (X_k , X_i), където X_k е съсед на X_i

AC-3 алгоритъм

  - Това трябва да се направи, понеже промяната в D_i може да направи възможни последващи редукции в домейните $D_к$
    - Дори преди това да сме разглеждали X_k
- Ако D_i е редуциран до празно множество, тогава алгоритъмът разбира, че целият CSP няма консистентно решение и връща незабавно грешка
- В противен случай, продължаваме с проверката, опитвайки се да задраскаме стойности домейните на променливите, докато опашката стане празна
  - В тази точка, получаваме един CSP, който е еквивалентен на оригиналния CSP
  - Двата има едни и същи решения, но консистентният към ребра в повечето случаи ще търси по-бързо
    - Понеже променливите му имат по-малки домейни

Оценка на AC-3

» Комплексността на AC-3 може да бъде оценена както следва

- Предполагаме един CSP с n променливи, всяка с кардиналност на домейна максимално d и с c бинарни ограничения
- Всяко ребро (X_k , X_i) може да бъде включено в опашката само d пъти, понеже X_i има максимално d стойности за изтриване
- Проверката за консистентност на ребро може да бъде направена $O(d^2)$ пъти, така че получаваме $O(cd^3)$ като най-лошо общо време

Разширение

- Възможно е понятието за консистентност на ребра да бъде разширено за n-арни ограничения
  - Нарича се генерализирана консистентност на ребра (или консистентност на супер ребра)
- Една променлива X_i е генерализирана консистентност на възли по отношение на едно n-арно ограничение, ако за всяка стойност v в домейна на X_i съществува един вектор от стойности, който е член на ограничението, получава всички стойности от домейните на кореспондиращите променливи и X_i му компонент е равен на v

Пример

» Напр., всички променливи имат домейн {0, 1, 2, 3}
  > X_i за да бъде консистентна по отношение на ограничението $X < Y < Z$, трябва да елиминираме 2 и 3 от домейна на X_i
  > Понеже ограничението не може да бъде удовлетворено когато X е 2 или 3

Консистентни пътища

» КР допринася за редуциране на областите на променливите
  > За някои проблеми консистентността на ребрата не е достатъчна за решение

» Напр., оцветяване картата на Австралия, но с два допустими цвята red и blue.
  > Използвайки консистентността на ребрата не можем да изведем нищо, понеже всяка променлива е КР – всяко ребро може да бъде red с blue на другия край (или обратно)
  > Така на този проблем няма решение
    + Понеже всичките Western Australia, Northern Territory и South Australia имат допирни точки, т.е. нуждаем се най-малко от три цвята

Консистентни пътища

- Консистентността на ребрата ограничава домейните (унарни ограничения), използвайки ребрата (бинарни ограничения)
- За да можем да продължим при проблеми подобни на оцветяването на карта, се нуждаем от по-строго дефиниция за консистентност
- Консистентност на път (КП)
  - Ограничава бинарните ограничения, използвайки неявни ограничения

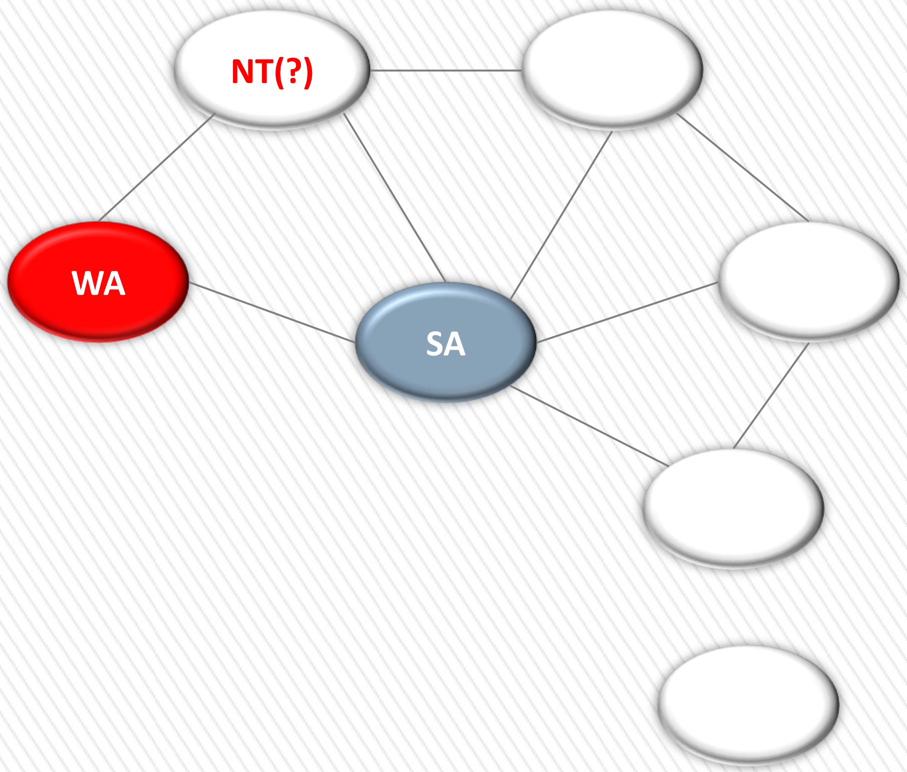
Консистентни пътища

- (X_i, X_j) е КП по отношение на трета променлива X_m, ако за всяко присвояване $(X_i = a, X_j = b)$,консистентно с ограниченията върху (X_i, X_j), съществува присвояване на X_m, което удовлетворява ограниченията върху (X_i, X_m) и (X_m, X_j)
  - Нарича се КП, понеже можем да мислим, като разглеждане на път от X_i към X_j през X_m

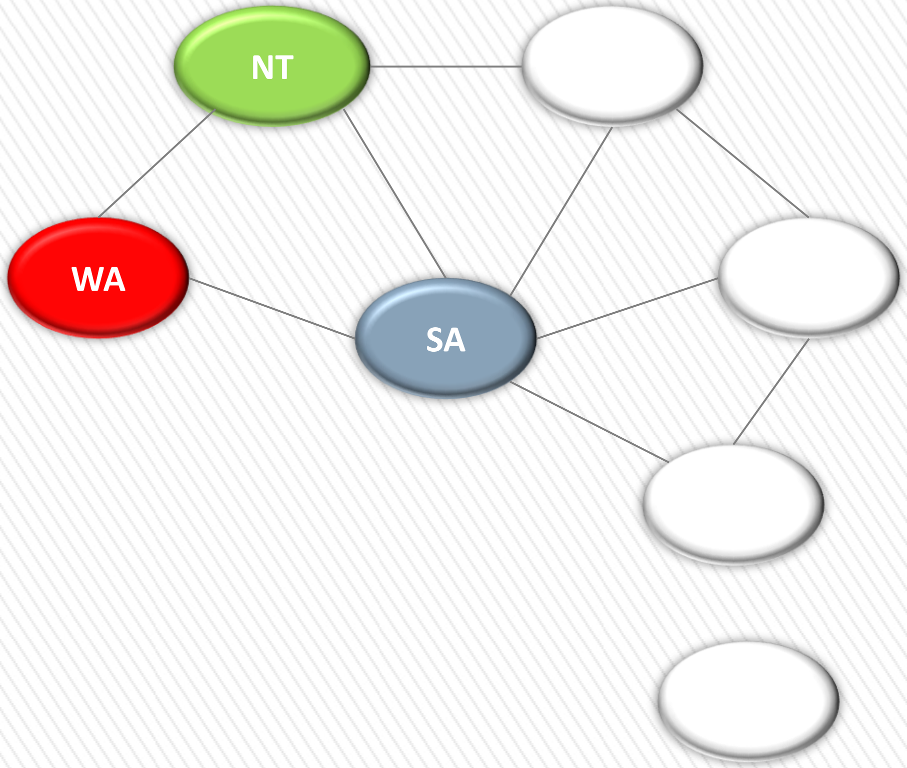
Пример: Австралия в два цвята

- » Ще направим множеството { WA, SA} КП по отношение на NT
- » Започваме с изброяване на консистентните присвоявания за множеството
  - > В случая са само две: { WA = red, SA = blue} и { WA = blue, SA = red}
  - > При тях NT не може да бъде нито red нито blue
    - + Понеже ще бъде в конфликт с някой от двете - WA или SA
- » Понеже няма валиден избор за NT, елиминираме двете присвоявания за { WA, SA}
  - > Следователно няма решение за проблема
- » Прилагане на консистентност на път
  - > PC-2 алгоритъм (подобен на AC-3)

Австралия с два цвята



Австралия с два цвята



K-консистентност

- С концепцията за k-консистентност могат да се дефинират по-строги форми
- Един CSP е k-консистентен, когато за всяко множество от k-1 променливи и за всяко консистентно присвояване на тези променливи, винаги може да се направи консистентно присвояване за k-та променлива
  - 1-консистентност: като се има предвид празно множество, можем да направим всеки набор от една променлива консистентен
    - Това е, което ние нарекохме консистентност на възел
  - 2-консистентност: същата като консистентност на ребро
    - За двоично ограничени мрежи
  - 3-консистентност: същата като консистентност на път
- Един CSP е строго k-консистентен, когато е k-консистентен, също (k-1)-консистентен, също (k-2)-консистентен, …

K-консистентност

- Един CSP е строго k-консистентен, когато е k-консистентен, също (k-1)-консистентен, също (k-2)-консистентен, …
- Да предположим, че имаме CSP с n възли и е строго n-консистентен, т.е. строго k-консистентен за k = n
- Можем да решим проблема по следния начин:
  - Първо, избираме последователна стойност за X_1
  - Гарантирано е, че ще можем да изберем стойност за X_2, защото графът е 2-консистентен, за X_3, защото е 3-консистентен и т. н.

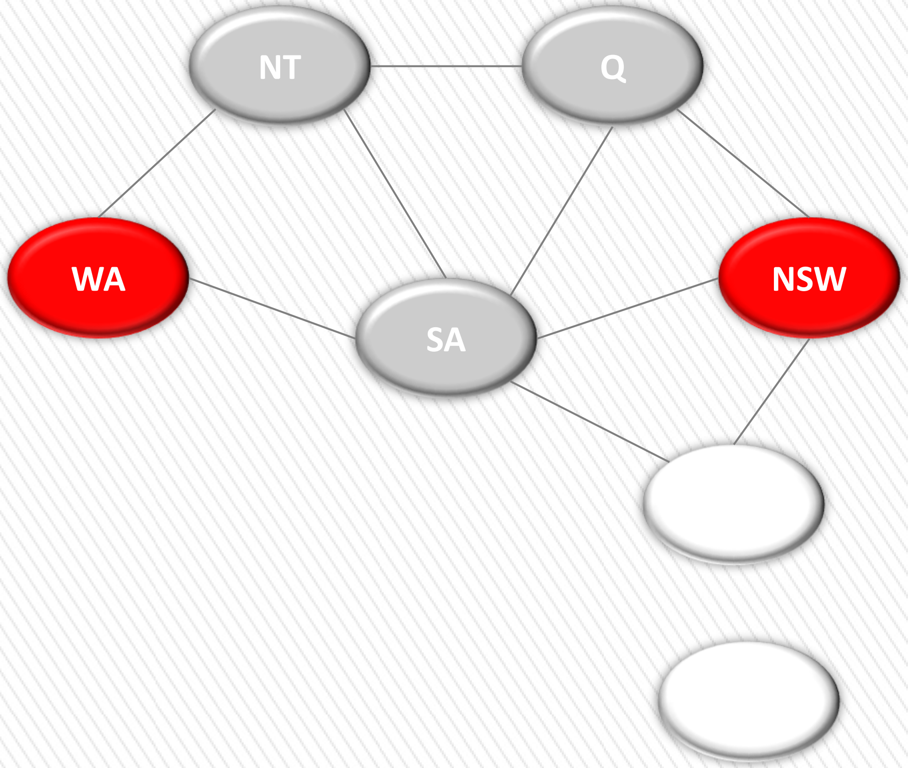
Оценка

- За всяка променлива X_i трябва само да търсим в d стойности в домейна, за да се намери стойност, съответстваща на X_1,. , , , X_{i-1}
- Гарантирано ще намерим решение във време $O(n^2d)$
- Разбира се, няма безплатен обяд:
  - Всеки алгоритъм за установяване n-консистентност отнема време експоненциално на n в най-лошия случай
  - По-лошото е, че n-консистентността също изисква пространство, което е експоненциално на n
  - Проблемът с паметта е дори по-тежък от времето

Глобални ограничения

- Обхващат произволен брой променливи
- Появяват се често в реални проблеми
- Могат да бъдат обработвани със специални алгоритми
- Напр., Alldiff ограничението:
  - Една проста форма за проверка на консистентност функционира както следва: когато в ограничението са включени m променливи, които имат общо n различни стойности и m > n
    - Отстраняваме от ограничението първо променлива, домейнът на която има само една стойност и изтриваме тази стойност от домейните на останалите променливи
    - Повтаряме докато останат променливи с по една стойност
    - Ако се появят празни домейни или домейни с повече стойности е разпозната неконсистентност

Пример



- Разпознаване на неконсистентност за присвояването {WA=red, NSW=red}
- Променливите SA, NT, Q са свързани посредством Alldiff ограничението
- AC-3 редуцира домейните на всяка променлива на {green, blue}, т.е. 3 променливи с две стойности

Пример: судоку

81 променливи

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

27 ограничения

Alldiff(A1, A2, A3, A4, A5, A6, A7, A8, A9)
Alldiff(B1, B2, B3, B4, B5, B6, B7, B8, B9)
…
Alldiff(A1, B1, C1, D1, E1, F1, G1, H1, I1)
Alldiff(A2, B2, C2, D2, E2, F2, G2, H2, I2)
…
Alldiff(A1, A2, A3, B1, B2, B3, C1, C2, C3)
Alldiff(A4, A5, A6, B4, B5, B6, C4, C5, C6)
…

Програмите за решаване на „судоку“ използват по-малко от 0.1 секунди за решаване на най-тежките задачи.

Пример: судоку

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

- Нека да видим колко далеч може да ни доведе консистентността на ребрата
- Да приемем, че ограниченията Alldiff представени като двоични ограничения (като $A1 \neq A2$), така че да можем да приложим директно алгоритъма AC-3

Пример: судоку

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7					?			8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Нека разгледаме променливата E6 - празния квадрат между 2 и 8 в средния квадрат:

- От ограниченията за требро в квадрата можем да премахнем 1, 2, 7 и 8 от домейна на E6
- От ограниченията за ребро в колоната можем да елиминираме 5, 6, 2, 8, 9 и 3
- Това оставя E6 с домейн {4} - с други думи, ние знаем отговора за E6

Пример: судоку

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7					**4**			8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1	**?**	3		

Нека разгледаме променливата I6 - празния квадрат между 1, 3 и 3:

- От ограниченията за ребро в колоната можем да премахнем 5, 6, 2, 4, 8, 9 и 3
- Елиминираме 1 по консистентността на дъгата с I5 и оставаме само стойността 7 в областта на I6

Пример: судоку

	1	2	3	4	5	6	7	8	9
A			3		2	**1**	6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7					**4**			8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1	**7**	3		

- Сега имаме 8 известни стойности в колона 6, така че поради консистентността на реброто можем да заключим, че A6 трябва да бъде 1

Пример: судоку

Решение:

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

Изводът продължава по този начин и в крайна сметка AC-3 може да реши целия пъзел - всички променливи имат домейни, сведени до една стойност

Изводи и търсене

- Проблемите със судоку могат да бъдат решени чрез извод за ограничения
- Много други CSPs не могат да се решават само с извод – необходимо е търсене на решения
- Алгоритми за търсене:
  - Търсене с възврат – върху частични присвоявания
  - Локално търсене – върху пълни присвоявания

Търсене с възврат за CSP

- Търсенето с възврат работи с частични присвоявания, като можем да използваме стандартно лимитирано търсене в дълбочина, където:
  - Състояние: частично присвояване
  - Оператор: добавяне на var = value към присвояване
- Стойностите се избират за една отделна променлива
  - Възврат, когато на променливата не могат да се присвоят допустими стойности

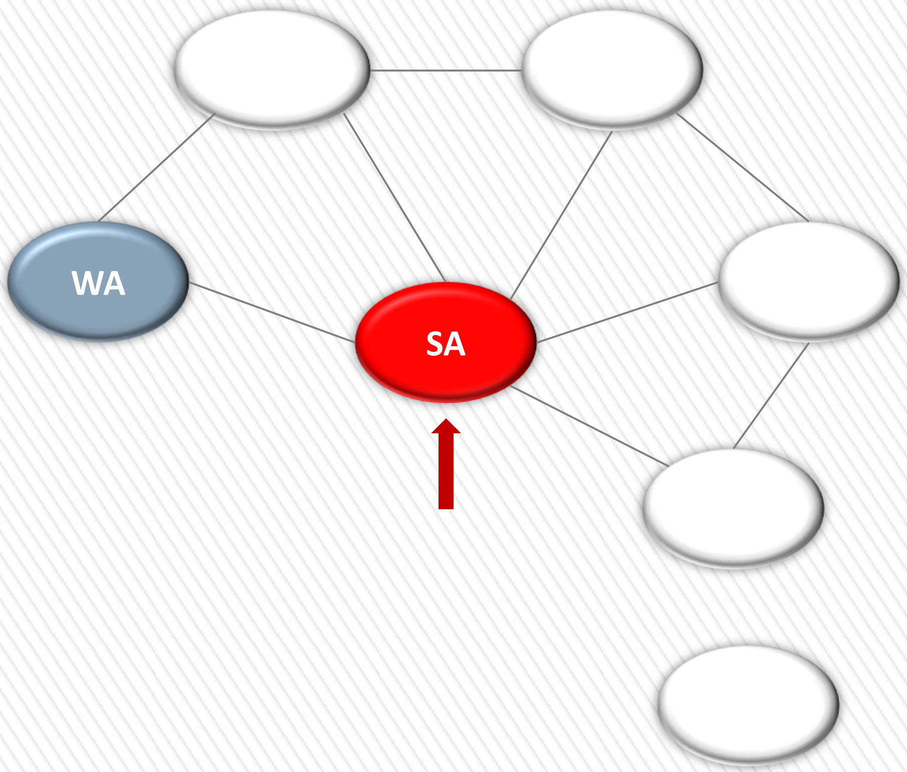
Оценка

- За един CSP с n променливи и размер на домейна d разпознаваме нещо ужасно:
  - Разклоняващият фактор на най-горното ниво е nd
  - На следващото ниво е (n-1)d и т.н.
  - Или получаваме дърво с $n!d^n$ листа – въпреки че има само d^n възможни пълни присвоявания
- Нашето смислено, но наивно формулиране на проблема е игнорира съществено свойство, което притежават всички CSPs: комутативност
  - Един проблем е комутативен ако последователността от прилагане на предварително зададено множество от действия няма влияние върху резултата

Една променлива за всеки възел

- Това е случаят за CSPs
  - При присвояване на стойности на променливи получаваме едни и същи частични присвоявания, независимо от последователността
- За това разглеждаме само една единствена променлива във всеки възел на дървото
  - С това ограничение броят на листата е d^n

Една променлива за всеки възел



В корена на дървото изборът е само между {SA=red, SA=green, SA= blue}, а не между {SA=red, WA=blue}

BACKTRACKING-SEARCH

```
function BACKTRACKING-SEARCH(csp) returns решение или failure
return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns решение или failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
       if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, value)
          if inferences != failure then
             add inferences to assignment
             result ← BACKTRACK(assignment, csp)
             if result != failure then return result
         remove {var = value} and inferences from assignment
return failure
```

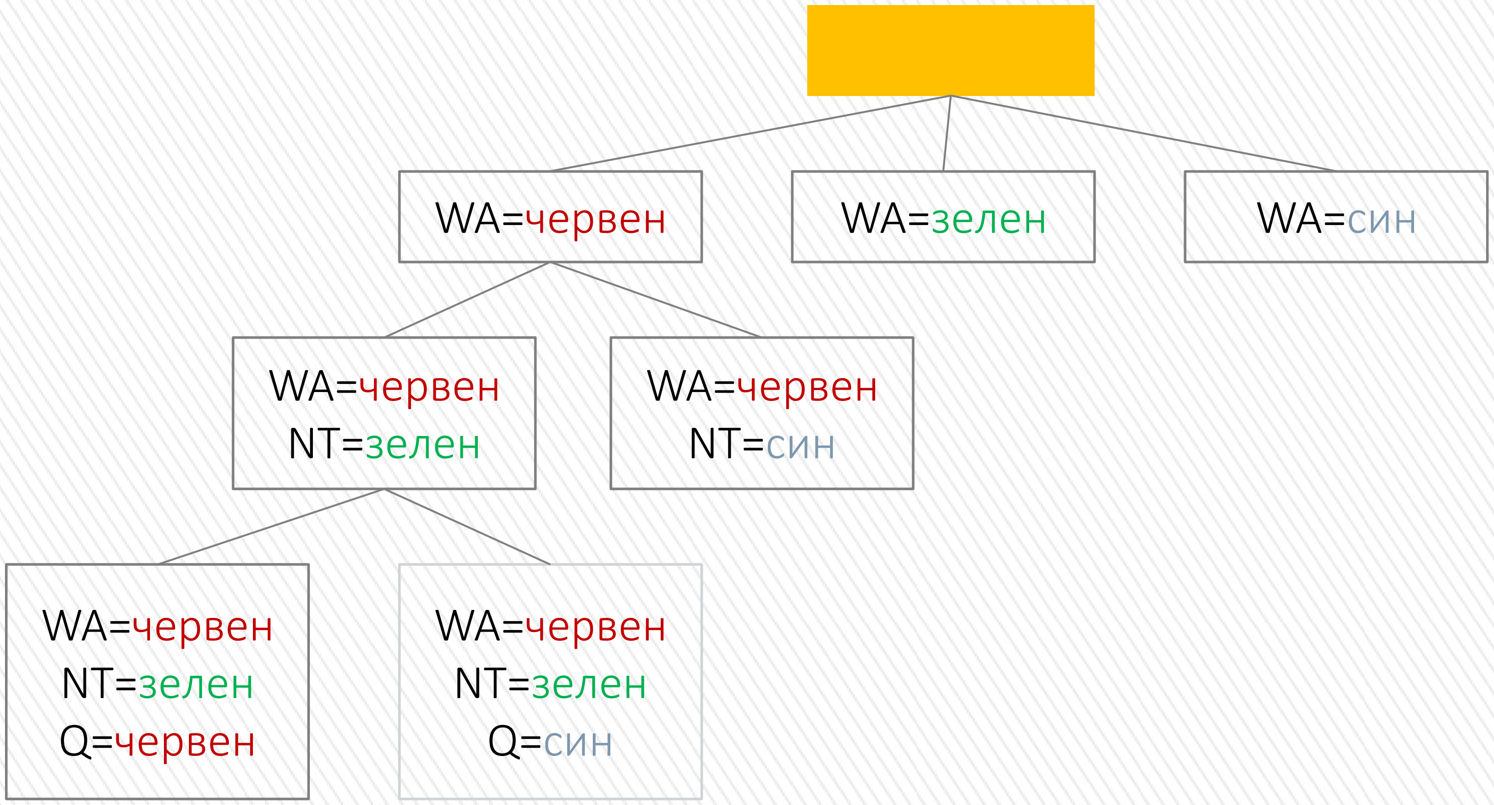
Коментар

- Алгоритъмът се моделира на базата на рекурсивното търсене в дълбочина
  - Връщане назад избира стойности за една променлива и се връща назад, когато дадена променлива няма легални стойности, които да се зададат
  - Той многократно избира една неприсвоена променлива и опитва всички стойности в областта на тази променлива да намери решение
  - Ако се открие несъответствие, тогава BACKTRACK връща грешка, причинявайки предишното повикване да опита друга стойност
    - Ако изборът на стойност води до неуспех (забелязан от INFERENCE или от BACKTRACK), тогава стойностите (включително тези, направени от INFERENCE) се премахват от текущото присвояване и се изпробва нова стойност

Коментар

- Можем да приложим евристики с общо предназначение (не зависими от приложната област):
  - SELECT-UNASSIGNED-VARIABLE – коя променлива първо ще получи стойност
  - ORDER-DOMAIN-VALUES – ред, в който се пробват стойностите
  - INFERENCE – какъв извод ще бъде пробван на всяка стъпка в процеса на търсене (опционно представяне на консистентност на ребро, път или k-консистентност)

Оцветяване на картата на Австралия



Коментар

- При обикновеното търсене използваме евристика, която извличаме от знанията за проблемната област
- За CSPs се оказва, че можем да търсим ефективно решения без такива специфични за областта знания
- Вместо това можем да снабдим недефинираните в алгоритъма функции с известна интелигентност за да отговорим на следните въпроси:
  - Коя променлива трябва да се присвоява като следваща (Select-Unassigned-Variable) и в каква последователност ще се изпробват стойностите (Order-Domain-Values)?
  - Какъв извод може да се прави след всяка стъпка (Inference)?
  - Когато търсенето се натъкне на присвояване, което нарушава ограничение, може ли да се избегне повтаряне на грешката?

Последователност променливи

- Най-простата стратегия за Select-Unassigned-Variable е да избира следващата неприсвоена променлива в последователност { X_1, X_2, …)
  - Рядко ефективно търсене
- MRV (Minimum Remaining Values) евристика
  - Избор на променлива с най-малко допустими стойности
- Degree евристика
  - Редуцира разклоняващия фактор на бъдещите избори, като избира променливата, която се съдържа в най-много ограничения върху останалите неприсвоени променливи

MRV евристика

- Ако имаме показаното присвояване, смислено е да следващото присвояване да бъде за SA (SA = син) вместо за Q, понеже има само една допустима стойност за SA
- След като сме присвоили стойност за SA, присвояванията за Q, NSW, V са принудително твърдо фиксирани

MRV евристика

» Обикновено работи по-добре в сравнение с чисто случайния избор или статично зададена последователност

» В определени случаи тази евристика не помага за търсене на решение

  > Напр., не помага за решаване коя област от Австралия да изберем като първа за оцветяване, понеже за всички области първоначално съществуват 3 цвята

  > В такива случаи можем да използваме degree евристиката

Degree евристика

- Степента на SA = 5 е най-високата
- Останалите променливи имат степени 2 или 3
- Степента на T е 0

Ограничения (9):

C = { SA ≠ WA, SA ≠ NT, SA ≠ Q, SA ≠ NSW, SA ≠ V, WA ≠ NT, NT ≠ Q, Q ≠ NSW, NSW ≠ V }

Последователност стойности

- След като е избрана една променлива, алгоритъмът трябва да определи последователността, в която се разглеждат нейните стойности
- Least-Constraining-Value
  - Предпочита стойността, която изключва най-малкото избори за съседните променливи в графа на ограничения

LCV евристика

- Ако имаме показаното присвояване и следващият ни избор е за Q
- Q = син е лош избор, понеже елиминира последната допустима стойност за съседа SA
- Евристиката предпочита Q = червен

Бележки

- Когато искаме да намерим всичките решения на един CSP, а не само първото възможно, тогава последователността не играе никаква роля
  - Понеже трябва да се изпробват всичките стойности
  - Същото се отнася за случая, когато няма решение
- Защо изборът на променливи се прави преди този на стойности?
  - За широк спектър от CSPs изборът първо на променлива с минимален брой на оставащи стойности, помага за по-ранно съкращаване на по-големи части от дървото на търсене
  - Когато търсим едно решение е смислено да търсим първо най-вероятните стойности

Свързване на търсене с извод

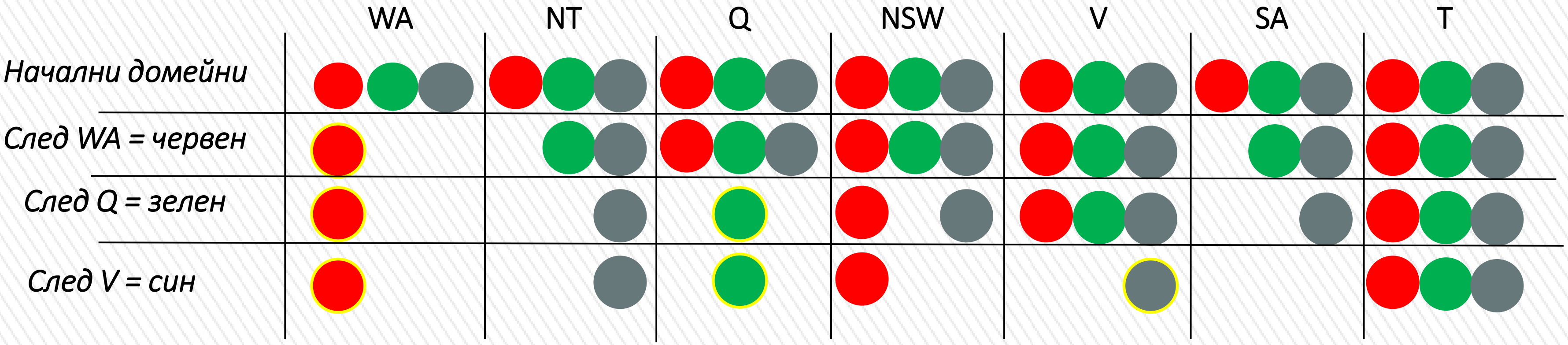
- До сега показахме как AC-3 и другите алгоритми могат да включат редуциране на домейните, преди да започне търсенето
- Изводът може да бъде още по-производителен в процеса на търсене
  - Винаги когато изберем стойност за една променлива, имаме напълно нова възможност да включим редуциране на домейните на съседни променливи

Предварителна проверка

- Forward checking - една от най-простите форми за извод
  - Винаги когато една променлива X е присвоена, стартира се проверка на КР за нея
  - Т.е., за всяка неприсвоена променлива Y, която е свързана с ограничения с X, се изтрива всяка стойност от домейна на Y, която е неконсистентна с избраната за X стойност
    - Ако в предишна стъпка е реализирана КР, няма смисъл от тази стъпка
- За много проблеми търсенето е по-ефективно, когато MRV евристиката се комбинира с forward checking

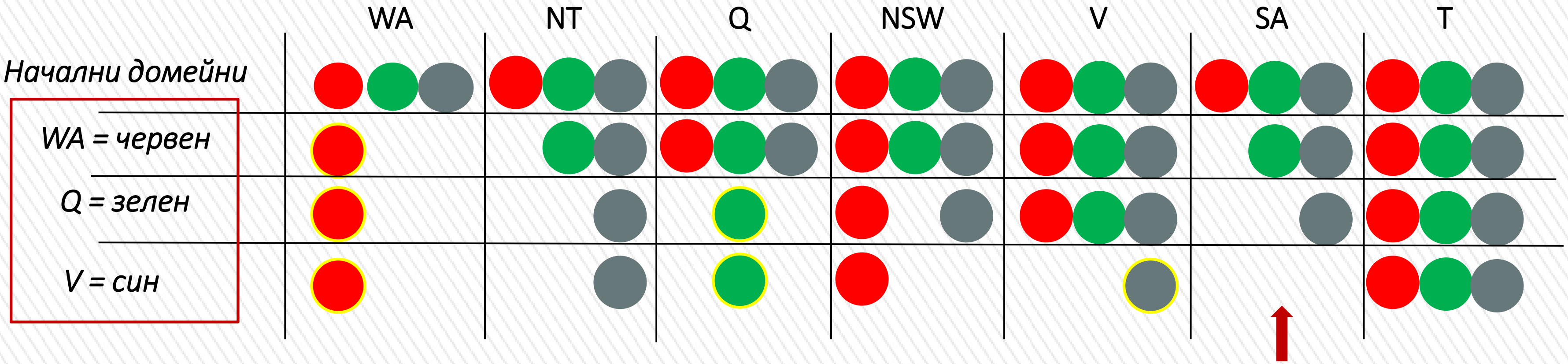
Пример

Изпълнение на търсене с възврат с forward checking за Австралия-CSP



Пример

- Forward checking разпознава, че за частичното присвояване проблемът не е консистентен
- Алгоритъмът предприема възврат



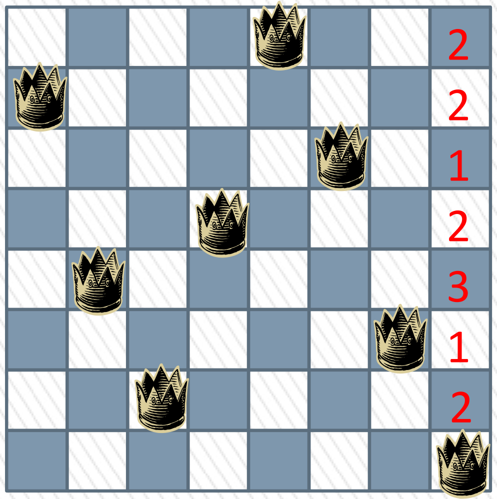
Локално търсене за CSP

- Алгоритмите за локално търсене се оказват много ефективни за решение на много CSPs
  - Използват пълна формулировка на състоянията
  - Начално състояние: присвоява стойност на всяка променлива
  - Търсене: променя стойността на една променлива
- Обикновено началните конфигурации нарушават много ограничения – ядрото на локалното търсене се състои в това да елиминира нарушените ограничения
- Min-Conflicts евристика
  - При избор на нова стойност за една променлива се избира тази, която причинява най-малко конфликти с останалите променливи

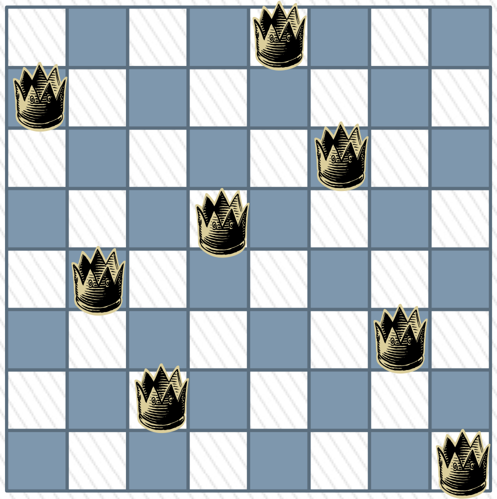
Min-Conflicts

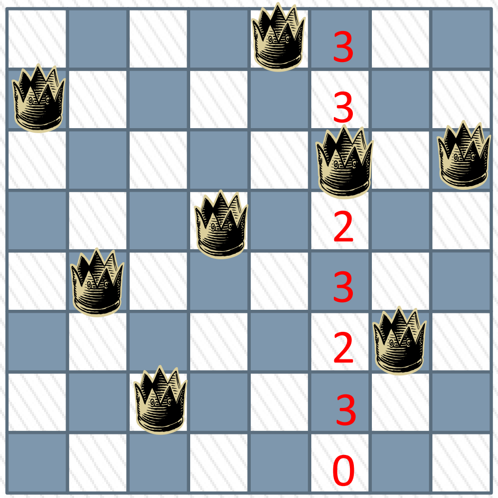
```
function MIN-CONFLICTS(csp, max-steps) returns a solution or failure
   inputs: csp, a constraint satisfaction problem
   max-steps, the number of steps allowed before giving up
   current ← an initial complete assignment for csp
   for i = 1 to max-steps do
      if current is a solution for csp then return current
      var ← a randomly chosen conflicted variable from csp.VARIABLES
      value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
      set var = value in current
 return failure
```

Пример: 8 царици

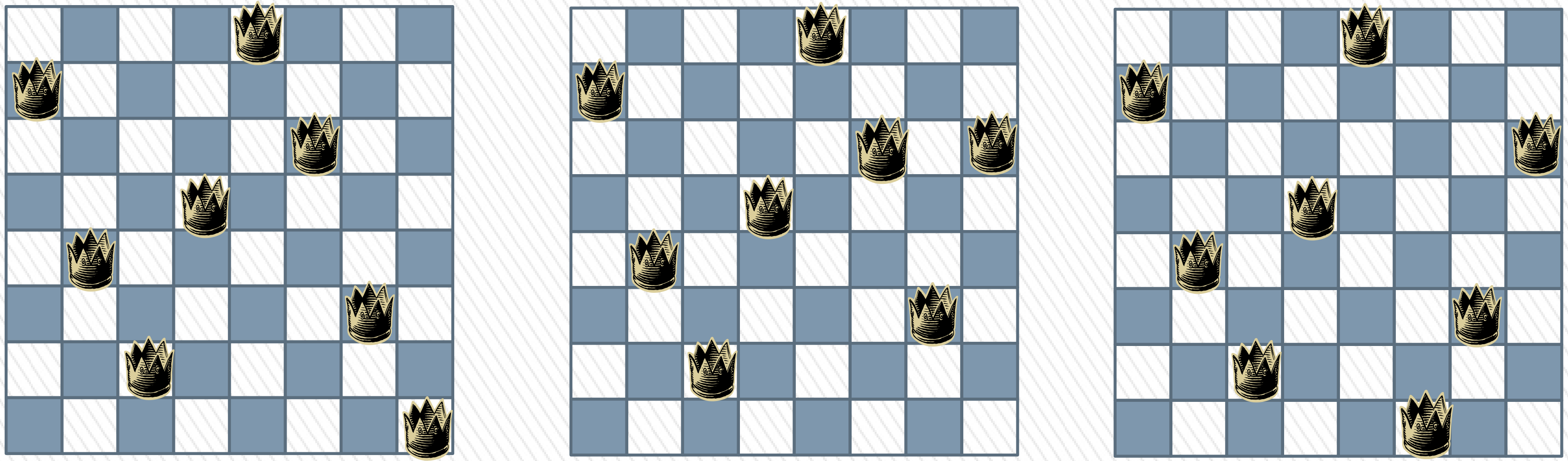


Пример: 8 царици

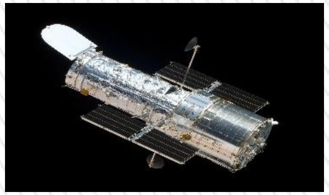




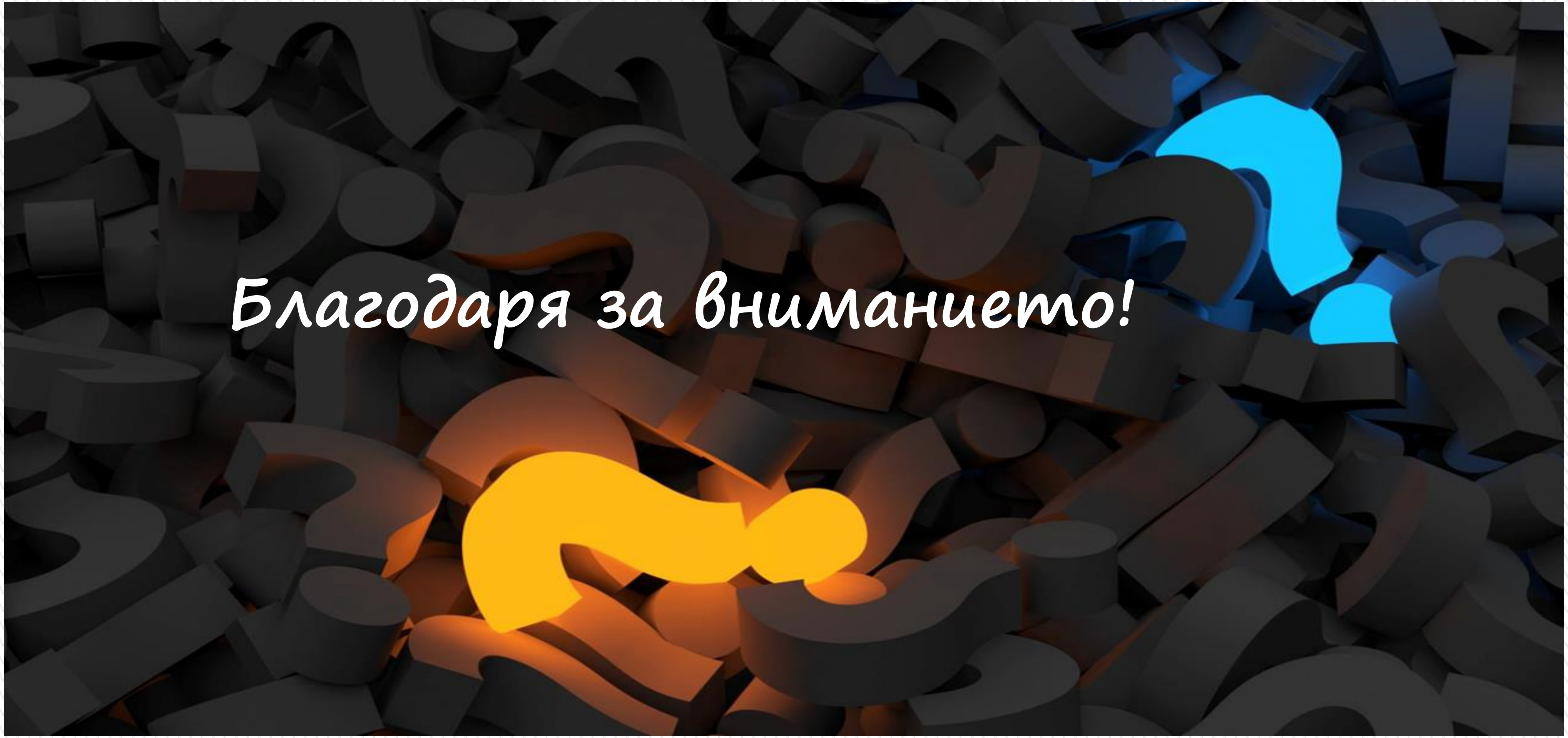
Пример: 8 царици



Коментар



- » Min-Conflicts – за много CSPs изненадващо ефективен
- » Ефективността му е почти независима от размера на проблема
  - > Напр., за примера в милиони царици намира решение средно в 50 стъпки
- » През 90-те години интензивни изследвания на локалното търсене
- » Min-Conflicts функционира също за тежки проблеми
  - > Използван за планиране на наблюденията на телескопът Хъбъл (Hubble)
    - + За 1 седмица наблюдения с предишни методи необходими 3 седмици за планиране
    - + С Min-Conflicts времето за планиране е редуцирано на 10 минути


Благодаря за вниманието!