Przeszukiwanie z więzami

Zagadnienia przeszukiwania z więzami (*Constrained Satisfaction Problem, CSP*), albo z ograniczeniami, stanowią specyficzną grupę problemów przeszukiwania w przestrzeni stanów, zdefiniowane jako:

- skończony zbiór zmiennych $X = \{x_1, x_2, ..., x_n\}$,
- dla każdej zmiennej x_i skończony zbiór możliwych jej wartości D_i, zwany **dziedziną** danej zmiennej,
- skończony zbiór C **więzów** (*constraints*) (zwanych również ograniczeniami), na wartości zmiennych, np., jeśli $x_1 = 5$, to x_2 musi być parzyste, a kombinacja $(x_1 = 5, x_2 = 8, x_3 = 11)$ jest niedozwolona.

Rozwiązaniem problemu CSP jest każde przypisanie wszystkim zmiennym wartości z ich dziedzin, które spełnia wszystkie więzy.

Przykład — kolorowanie mapy

Prostym, klasycznym przykładem zagadnienia CSP jest problem kolorowania mapy. Należy przypisać poszczególnym krajom (lub terytoriom, jak na rysunku po lewej), kolorów ze zbioru {R,G,B} tak, aby sąsiednie obszary miały różne kolory:

Aby zdefiniować ten problem formalnie, wprowadzamy zbiór zmiennych $X = \{WA, NT, Q, NSW, V, SA, T\}$. Dziedziną wszystkich zmiennych jest zbiór $D_i = \{red, green, blue\}$. Więzy będą reprezentowały wymaganie, aby sąsiednie terytoria miały różne kolory. Ponieważ jest dziewięć granic pomiędzy terytoriami, więzy określamy tylko dla tych graniczących terytoriów: $C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$

Przeszukiwanie z więzami — bliższa analiza

Zauważmy, że problemy CSP są w istocie szczególnym przypadkiem ogólnego przeszukiwania w przestrzeni stanów, jeśli potraktujemy zbiór więzów jako specyfikację celu, a przypisywanie wartości zmiennym jako operatory. Można zatem zastosować do tych zagadnień wszystkie wcześniej poznane metody.

Dlaczego chcemy przedstawiać zagadnienia CSP jako specjalny rodzaj problemu?

Po pierwsze, jest to naturalna reprezentacja dla wielu rodzajów problemów.

Po drugie, rozwiązując problem CSP możemy znacznie zmniejszyć zakres przeszukiwania przez uwzględnienie więzów. Na przykład, gdy ustalona została wartość jednej ze zmiennych, jak $SA = blue$ w zagadnieniu kolorowania mapy Australii, możemy wyeliminować ten kolor u wszystkich sąsiadów. Bez tego spostrzeżenia, przeszukiwanie ma do rozważenia $3^5 = 243$ możliwości przypisań wartości dla sąsiadów, a po uwzględnieniu tego więzu już tylko $2^5 = 32$.

Istnieje szereg wyspecjalizowanych algorytmów rozwiązywania zagadnień CSP wykorzystujących tego typu mechanizm. Niektóre z nich będą tu przedstawione.

Przykłady zagadnień CSP

Przykładami zagadnień CSP jest wiele klasycznych łamigłówek, takich jak: kryptoarytmetyka, kolorowanie grafu lub mapy, problem 8-hetmanów, lub zagadki sudoku.

```
  T W O
+ T W O
=======
F O U R
```

Problemami CSP jest również wiele rzeczywistych zagadnień technicznych, jak np. harmonogramowanie prac w zakładzie produkcyjnym, gdzie w organizacji procesu produkcyjnego trzeba brać pod uwagę kolejność wykonywania poszczególnych operacji, ale także wykorzystanie narzędzi, maszyn, jak również określonych pracowników.

Szczególnym przypadkiem takiego harmonogramowania jest problem układania planu zajęć dla większej szkoły lub uczelni.

Jeszcze inne przykłady CSP:

- tak zwany problem SAT, czyli przypisanie wartości 0/1 zmiennym formuły logicznej, zapewniające spełnienie tej formuły,
- problem etykietowania węzłów pozwalający rozpoznawać obiekty na obrazach po ich konturowaniu
- projektowanie układów VLSI,
- wiele innych.

Wiele spośród tych zagadnień stanowią problemy NP-trudne.

Warianty zagadnień CSP

Najprostszy rodzaj CSP dotyczy zmiennych z dyskretnymi i skończonymi dziedzinami.

Możliwe są również dziedziny dyskretne ale nieskończone. Na przykład, w zagadnieniu harmonogramowania prac może nie istnieć ograniczenie czasowe na rozpoczęcie poszczególnych operacji.

Istnieją również problemy typu CSP z dziedzinami ciągłymi. Przykładem może być **programowanie liniowe**, gdzie więzami są równania lub nierówności nieliniowe.

W ogólności, rozwiązanie problemu CSP może istnieć lub nie, bądź może istnieć wiele rozwiązań. Celem przeszukiwania może być znalezienie jednego dowolnego rozwiązania, wszystkich rozwiązań, bądź rozwiązania optymalnego w sensie jakiejś danej funkcji kosztu.

Szczególną kwestią w CSP są rodzaje więzów.

Rodzaje więzów

Najprostszym rodzajem więzu jest **więz unarny**, który jest warunkiem na pojedynczą zmienną.

Więzy binarne są warunkami obejmującymi dokładnie dwie zmienne.

Więzy obejmujące więcej niż dwie zmienne będziemy łącznie traktowali jako **więzy globalne**.

Okazuje się, że można traktować więzy występujące w CSP jako binarne, czyli obejmujące pary zmiennych. Więzy na pojedyncze zmienne, jeśli występują w oryginalnym sformułowaniu problemu, można wbudować w definicje ich dziedziny. Natomiast więzy obejmujące więcej niż dwie zmienne można przekształcić na binarne.

Algorytmy prezentowane poniżej zakładają istnienie wyłącznie węzłów binarnych.

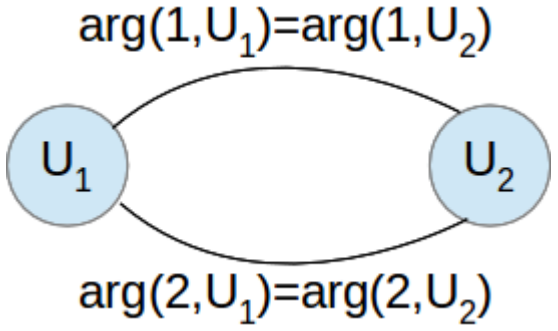
Więzy globalne — kodowanie dualne

Jeden ze schematów konwersji więzów globalnych na binarne, tzw. **kodowanie dualne** (*dual encoding*). Polega ono na wprowadzeniu nowej zmiennej dla każdego więzu oryginalnego problemu. Dziedziną każdej nowej zmiennej jest zbiór n-ek wartości tych oryginalnych zmiennych, które występowały w więzie, spełniających oryginalny więz.

W ten sposób, oryginalne więzy są automatycznie spełnione przez wartości zmiennych zawartych w nowych zmiennych. Pozostaje zapewnić by wartości spełniały wszystkie więzy naraz. W tym celu kodowanie dualne wprowadza nowe więzy pomiędzy wszystkimi parami zmiennych (nowych), które zawierają te same zmienne oryginalne. Treścią więzów jest by odpowiednie zmienne (oryginalne) miały te same wartości.

Rozważmy przykład zagadnienia CSP z trzema zmiennymi: $X = \{x, y, z\}$, ich dziedzinami $D_{x,y,z} = \{1, 2, 3\}$, i dwoma więzami: $\mathcal{C} = \{x + y = z, x < y\}$. Kodowanie dualne dla tego problemu zawiera dwie zmienne i dwa więzy:

$$
\begin{array}{lcl}
U_1 & : & \langle oc_1, [x, y, z] \rangle, D_{U_1} = \{(1,2,3), (2,1,3), (1,1,2)\} \\
U_2 & : & \langle oc_2, [x, y] \rangle, \quad D_{U_2} = \{(1,2), (1,3), (2,3)\} \\
C_1 & : & arg(1, U_1) = arg(1, U_2) \\
C_2 & : & arg(2, U_1) = arg(2, U_2)
\end{array}
$$



Preferencje

Jeszcze inny wariant zagadnień CSP obejmuje problemy, w których więzy nie są traktowane jako warunki absolutne, które muszą być spełnione przez każde rozwiązanie. Zamiast tego, możemy je traktować jako **więzy preferencji**, wskazujące które rozwiązania są preferowane.

Na przykład, w zagadnieniu układania planu zajęć uczelni, pewne więzy, takie jak nieumieszczanie dwóch różnych grup zajęciowych w tej samej sali o tym samym czasie, są nienaruszalne. Ale można również dopuścić takie więzy jak warunki, że profesor Q nie chce prowadzić zajęć po południu, a profesor X nie chce prowadzić zajęć w sąsiedniej sali z profesorem Y. Te więzy mogłyby być traktowane jako preferencje, i konkretne rozwiązanie mogłoby je naruszać, tylko nie byłoby ono optymalne.

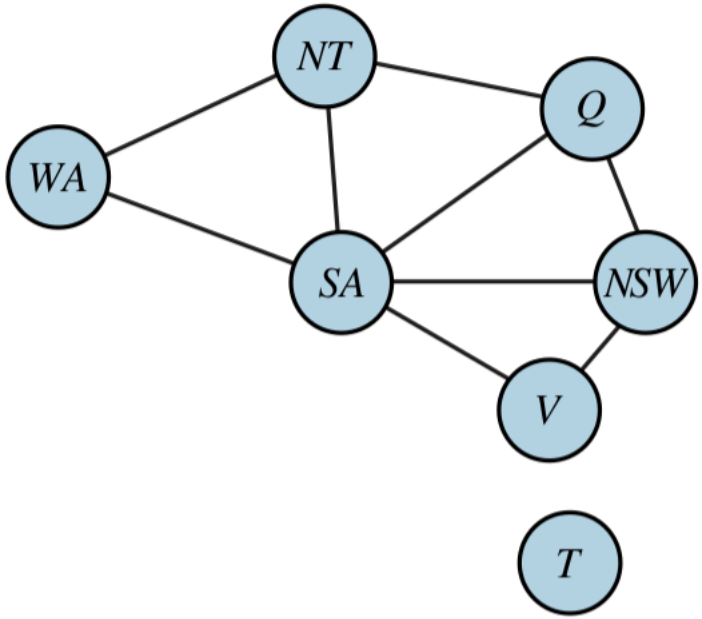
Preferencje można wprowadzić w postaci kosztów poszczególnych przypisań wartości do zmiennych. Wtedy proces poszukiwania rozwiązania CSP dążyłby do znalezienia rozwiązania o minimalnym koszcie.

Pozostała część tej prezentacji obejmuje tylko zagadnienia CSP z binarnymi więzami absolutnymi.

Graf więzów

W analizie zagadnień CSP przydatny jest **graf więzów**, którego węzły odpowiadają zmiennym, a łuki — więzom oryginalnego problemu CSP.

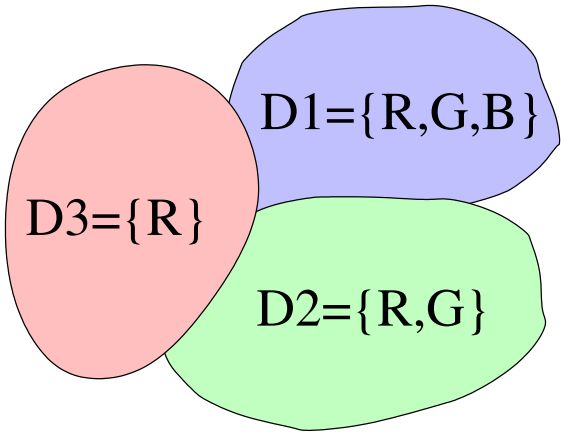
Graf więzów dla powyższego problemu kolorowania mapy Australii:



Oczywiście z każdym węzłem grafu związana jest dziedzina odpowiedniej zmiennej, a z każdym łukiem – odpowiadający mu więz.

Lokalne spełnianie więzów

Rozważmy prosty przykład kolorowania mapy.
Należy tak przypisać obszarom danej mapy kolory ze zbiorów dopuszczalnych kolorów, być może różnych dla różnych obszarów, aby obszary sąsiadujące miały przypisane różne kolory.



Zanim zaczniemy przeszukiwać przestrzeń możliwych podstawień wartości zmiennych, możemy przeprowadzić pewne analizy **lokalnego spełniania więzów**.

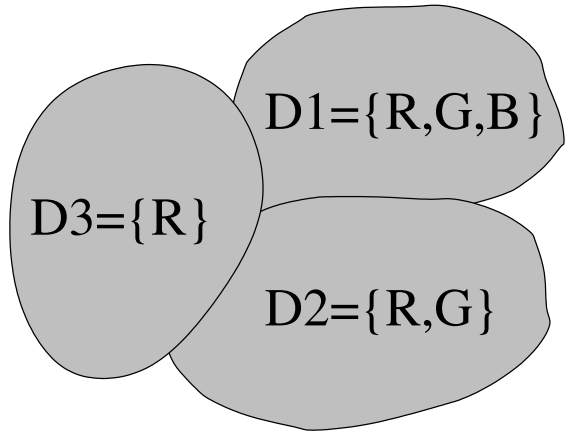
Traktując każdy łuk grafu więzów jako parę przeciwsobnych łuków skierowanych, określamy **spójność łukową** (*arc consistency*) łuku skierowanego $x_i \longrightarrow x_j$ grafu, która zachodzi jeśli $\forall x \in D_i \; \exists y \in D_j$ i para (x, y) spełnia wszystkie więzy określone na łuk.

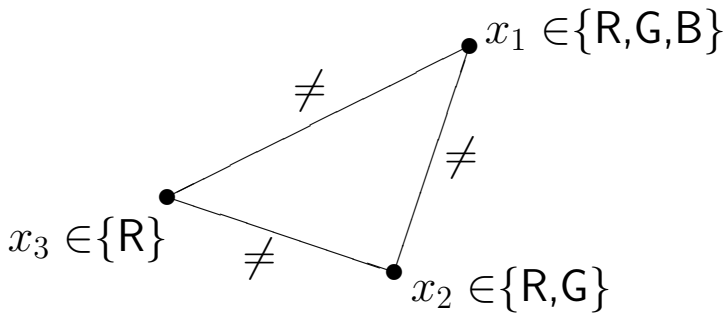
Można **przywrócić spójność** niespójnym łukom grafu więzów przez usuwanie wartości z dziedzin pewnych zmiennych (konkretnie, tych wartości $x \in D_i$ dla których nie istnieje wartość $y \in D_j$ spełniająca jakiś wybrany więz).

Ta procedura pozwala na zredukowanie i uproszczenie oryginalnego problemu.

Spójność łukowa

Rozważmy następujące przykładowe zagadnienie kolorowania mapy:
$D_1 = \{R, G, B\}$,
$D_2 = \{R, G\}$,
$D_3 = \{R\}$,
$\mathcal{C} = \{x_1 \neq x_2, x_2 \neq x_3, x_1 \neq x_3\}$.





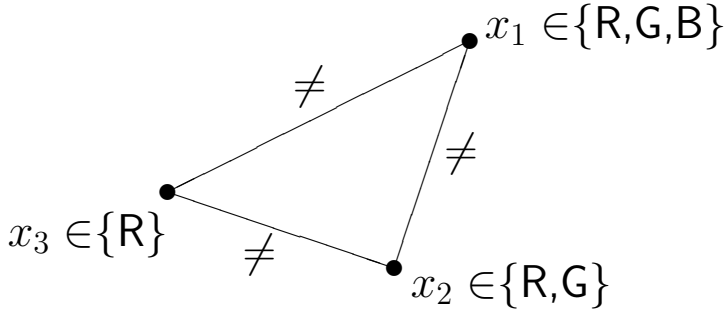
Po lewej graf więzów dla powyższego problemu kolorowania mapy. Dla łuku (x_1—x_2) spójność łukowa zachodzi, ponieważ spełnione jest zarówno: $\forall x \in D_1 \; \exists y \in D_2 \; x \neq y$
jak i: $\forall y \in D_2 \; \exists x \in D_1 \; x \neq y$.

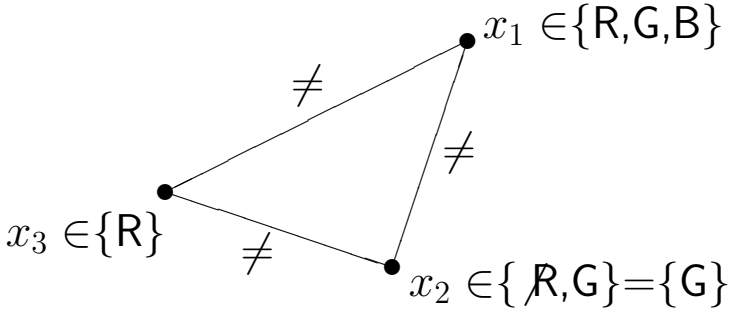
Fakt zachodzenia spójności łukowej to w efekcie niezbyt pozytywna wiadomość. Oznacza on, że analiza spójności tego konkretnego łuku w grafie nic nie wnosi. Jednak pełna analiza grafu CSP może czasami przynieść bardzo wymierne wyniki.

Przykład: kolorowanie mapy

Ponownie rozważmy przedstawione wcześniej zagadnienie kolorowania mapy:
$D_1 = \{R, G, B\}$,
$D_2 = \{R, G\}$,
$D_3 = \{R\}$,
$\mathcal{C} = \{x_1 \neq x_2, x_2 \neq x_3, x_1 \neq x_3\}$.

Analiza pierwszego więzu ($x_1 \neq x_2$) nie daje żadnych wyników, bo, jak już stwierdziliśmy, dla tego więzu istnieje spójność łukowa. (Dla każdej wartości z D_1 istnieje w D_2 wartość spełniająca ograniczenie, i na odwrót.)

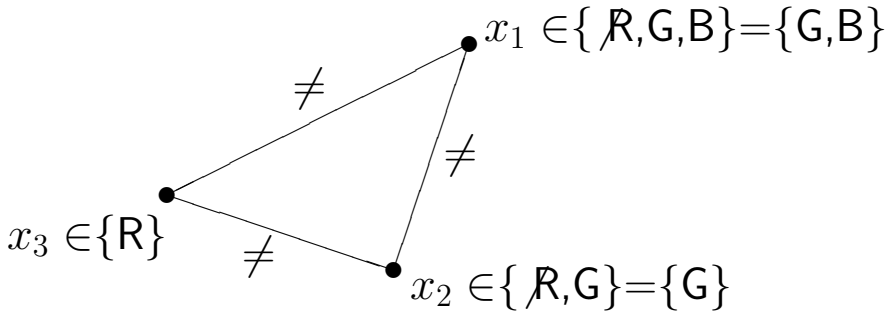




Jednak analiza drugiego więzu ($x_2 \neq x_3$) przynosi pewne pozytywne rezultaty. O ile dla $x_3 =$ R istnieje odpowiednia wartość dla x_2, to dla $x_2 =$ R nie da się dobrać wartości x_3 spełniającej ten więz. Zatem wartość R możemy usunąć z dziedziny zmiennej x_2.

Przykład: kolorowanie mapy (cd.)

Podobna analiza w odniesieniu do więzu ($x_1 \neq x_3$) pozwala wykluczyć z dziedziny zmiennej x_1 wartość R:



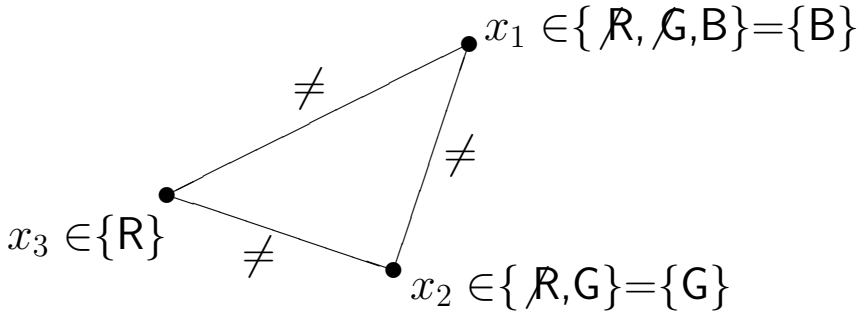
Analiza wszystkich więzów zakończyła się częściowym usunięciem wartości z dziedzin zmiennych. Zagadnienie zostało zredukowane (mniej jest możliwych przypisań wartości zmiennym), ale nadal istnieje więcej niż jedno potencjalne rozwiązanie.

Łatwo jednak zauważyć, że analizę spójności łukowej można, i należy, kontynuować.

Propagacja więzów

Ponieważ wynikiem sprawdzania spójności łukowej jest redukcja dziedzin niektórych zmiennych, ma sens jego powtórzenie nawet dla łuków grafu, które pierwotnie spójność posiadały, bądź została im ona przywrócona. Prowadzi to do **propagacji więzów** (*constraint propagation*), czyli powtarzania analizy spójności więzów i redukcji dziedzin tak długo jak daje to jakieś efekty.

Propagacja więzów w omawianym przykładzie kolorowania mapy powoduje dla łuku $(x_1 \neq x_2)$ — początkowo spójnego — usunięcie wartości G z dziedziny x_1:

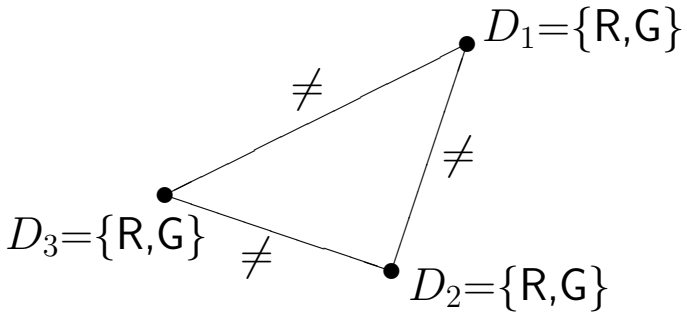


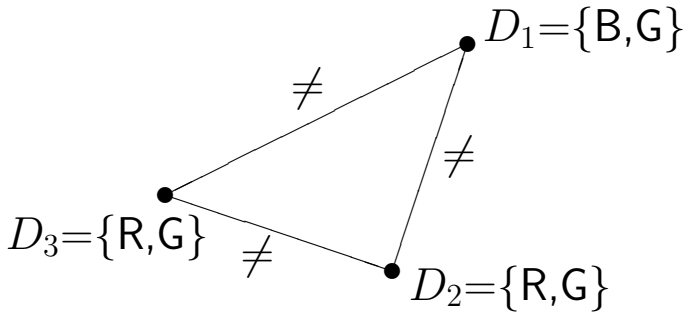
Ostatecznie wszystkie zmienne mają jednoelementowe dziedziny, i w dodatku zawarte w nich wartości spełniają wszystkie więzy. Zatem propagacja więzów doprowadziła w tym przypadku do znalezienia jedynego rozwiązania.

W ogólności jednak analiza spójności i propagacja więzów prowadzi jedynie do uproszczenia, a niekoniecznie do całkowitego rozwiązania problemu.

Spójność łukowa — przykłady nierozwiązane

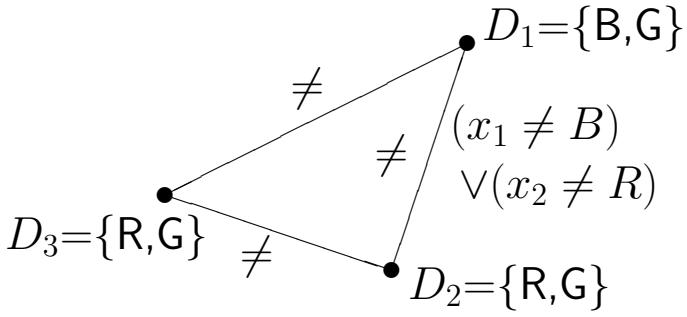
Jak łatwo zauważyć, w przedstawionym tu innym przykładzie zagadnienia kolorowania mapy, wszystkie łuki są spójne. Pomimo to, zagadnienie nie posiada rozwiązania.





W kolejnym przykładzie ponownie wszystkie łuki są spójne. Zagadnienie posiada dwa rozwiązania, lecz propagacja więzów nie pozwala ich wyznaczyć, a wręcz nie przynosi żadnych redukcji dziedzin zamiennych.

Jeśli do poprzedniego przykładu dodamy więz: $(x_1 \neq B) \vee (x_2 \neq R)$ to otrzymamy wariant, w którym tylko jedno z dwóch rozwiązań jest dopuszczalne, ale nie da się go wyznaczyć metodą propagacji więzów.



Zatem obliczanie spójności łukowej i propagacja więzów same w sobie nie gwarantują znalezienia rozwiązania. Konieczne jest przeszukiwanie.

Algorytmy propagacji więzów

Najprostszym sposobem sprawdzania spójności łukowej jest branie po kolei wszystkich więzów, i obliczanie ich warunków logicznych, oraz powtarzanie procesu tak długo, aż pełny cykl sprawdzania wszystkich więzów nie przyniesie żadnych zmian. Przy wielu więzach może to trwać długo. Jednak ponieważ te same warunki sprawdzane są wiele razy, możliwe są pewne oszczędności.

Można zauważyć, że po wykonaniu redukcji pewnej dziedziny D_i propagacja może przynieść nowy wynik tylko przy powtórnym sprawdzaniu łuków o postaci (D_k, D_i), i tylko takie warto sprawdzać. Co więcej, przy jakiejkolwiek redukcji w D_k nie ma już potrzeby sprawdzania łuku (D_i, D_k), ponieważ elementy usunięte w wyniku tej redukcji z D_k nie były już potrzebne dla zapewnienia spełnienia jakiegokolwiek więzu dla żadnego elementu z D_i. Taki sposób propagacji więzów nazywa się algorytmem AC-3 (1977).

Kiedy spójność łuku sprawdzana jest po raz kolejny, warunki są sprawdzane dla tych samych par wartości. Zapamiętanie tych sprawdzonych par wartości w odpowiedniej strukturze danych pozwoliłoby nie powtarzać ich sprawdzania w kolejnych cyklach. Wykorzystuje to algorytm propagacji więzów AC-4 (1986).

Spójność ścieżkowa

Definiujemy dla grafu więzów problemu CSP pojęcie **K-spójności**. Graf jest K-spójny (dla pewnego K), jeśli dla dowolnych (K-1) zmiennych, które mają spełnione wszystkie więzy między sobą, dla dowolnej (K-1)-tki wartości tych (K-1) zmiennych spełniającej wszystkie więzy dla (K-1) zmiennych, istnieje wartość w dziedzinie dowolnie dobranej K-tej zmiennej, taka, że powstała w ten sposób K-tka wartości zmiennych spełnia wszystkie więzy dla K zmiennych.

Graf więzów jest **silnie K-spójny** jeśli jest K-spójny dla każdego J, J$<$K.

Zauważmy, że zdefiniowana wcześniej spójność łukowa jest równoważna silnej 2-spójności grafu więzów.

Silna 3-spójność grafu jest również nazywana **spójnością ścieżkową** (*path consistency*).

Znaczenie K-spójności jest takie, że jeśli graf problemu CSP z n węzłami jest silnie n-spójny, to problem można rozwiązać bez przeszukiwania. Jednak algorytmy osiągania K-spójności są eksponencjalne, więc zwykle się to nie opłaca. Wyjątkiem jest osłabiona wersja spójności ścieżkowej — **ograniczona spójność ścieżkowa** (*restricted path consistency*), dla której algorytm jest czasem stosowany.

Przeszukiwanie w zagadnieniach CSP

W zagadnieniach CSP można stosować wszystkie omówione wcześniej algorytmy przeszukiwania. Jednak w większości istotnie trudnych problemów CSP, gdzie więzy mają charakter trudnych do rozwiązania, ciasnych kompromisów, największe znaczenie ma właśnie syntaktyczna i semantyczna analiza więzów.

Natomiast zwykle trudno jest wypracować użyteczną heurystykę, która byłaby zdolna pokierować procesem przeszukiwania przestrzeni przypisań wartości zmiennym.

Dlatego często stosowanym algorytmem jest najprostszy z algorytmów przeszukiwania, czyli przeszukiwanie z nawracaniem BT. Zamiast dobrej heurystyki wybierającej najlepsze możliwości w pierwszej kolejności, ten algorytm jest uzupełniony lokalną analizą więzów. Powoduje to redukcję liczby możliwości w kolejnych krokach wgłąb algorytmu. W skrajnym przypadku, gdyby dziedzina którejś ze zmiennych zredukowała się do zbioru pustego, algorytm dokonuje natychmiastowego nawrotu do alternatywnych wartości wcześniejszych przypisań.

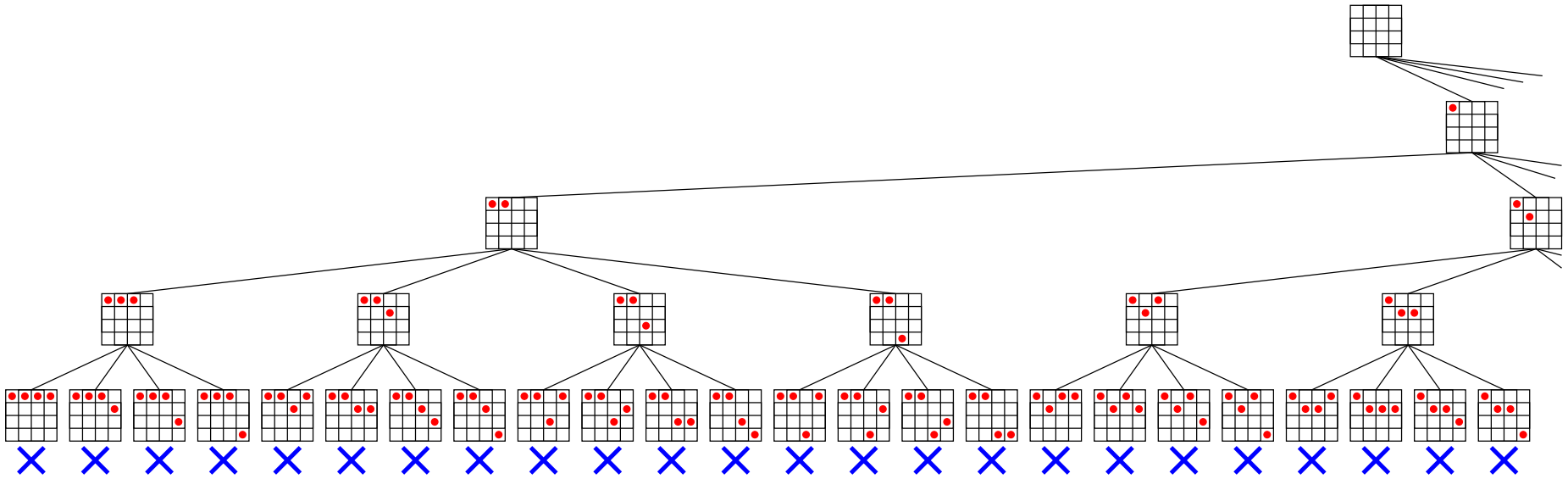
Przeszukiwanie z nawracaniem

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
    return BACKTRACK(csp, { })

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

Przykład: problem 4 hetmanów

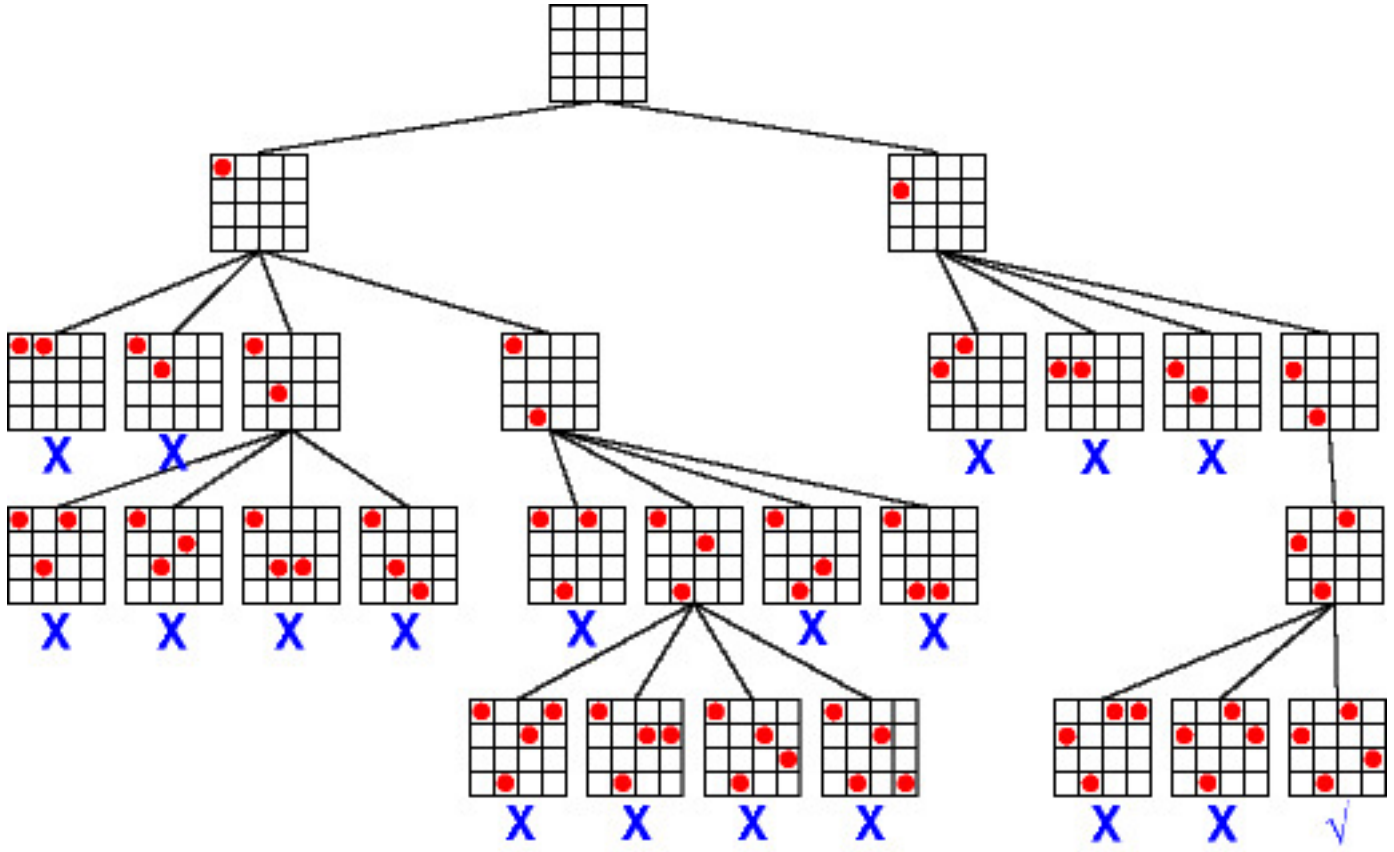
Rozważmy zastosowanie algorytmu przeszukiwania z nawracaniem (BT) do problemu 4 hetmanów. Poniższy rysunek przedstawia fragment drzewa przeszukiwania (należy pamiętać, że algorytm BT przeszukuje to drzewo systematycznie, lecz nie pamięta całej jego struktury, a jedynie bieżącą ścieżkę):



Algorytm oczywiście poradzi sobie z problemem, jednak sprawdza wiele możliwości, które możnaby łatwo wykluczyć z rozważań sprawdzając spełnienie więzów. Zauważmy, że wszystkie zaznaczone konfiguracje końcowe są niepoprawne ze względu na umieszczenie drugiego hetmana, i widać to już na drugim poziomie zagłębienia.

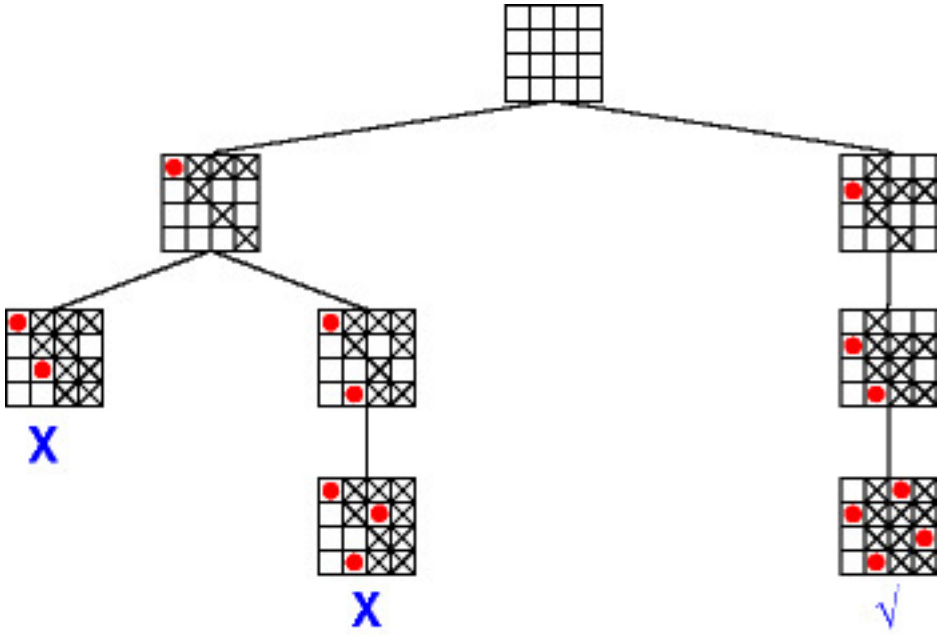
Przykład: problem 4 hetmanów (cd.)

Oczywistym ulepszeniem algorytmu jest więc natychmiastowe sprawdzanie wszystkich więzów binarnych, gdy tylko przypisane zostaną zmienne wchodzące w skład danego więzu. Stwierdzenie naruszenia któregokolwiek więzu powoduje nawrót. Nazwiemy ten algorytm BT-EC (*early checking*). Zauważmy, że wcześniejsze sprawdzanie więzów zawsze się opłaca, ponieważ te same więzy musiałyby i tak być sprawdzone później.



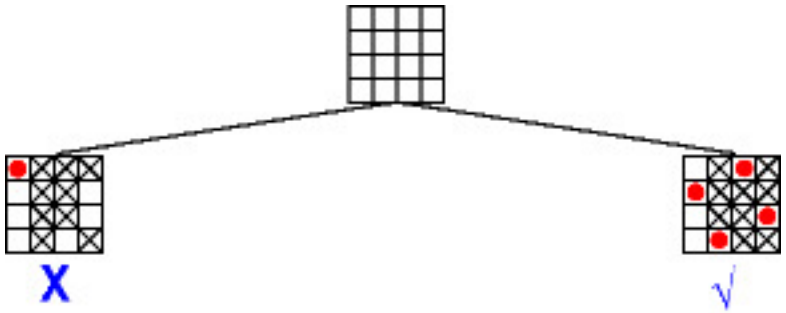
Przykład: problem 4 hetmanów (cd.)

Połączenie przeszukiwania z nawracaniem z minimalną formą sprawdzania spójności więzów zwane jest algorytmem **sprawdzania wprzód** BT-FC (*forward checking*). Sprawdzane są wszystkie więzy dotyczące każdorazowo podstawionej zmiennej, i tylko one. Ten algorytm w prawie każdym przypadku daje lepsze wyniki niż BT-EC, i oczywiście BT.



Przykład: problem 4 hetmanów (cd.)

Możemy również zastosować pełne sprawdzanie spójności łukowej, z propagacją więzów na dziedziny wszystkich zmiennych, również tych jeszcze niepodstawionych. Taki wariant przeszukiwania nazywany jest algorytmem BT-LA (*look-ahead*). W wielu przypadkach pozwala to wyeliminować większość przeszukiwania, jednak ilość obliczeń w tym przypadku może być większa niż w przypadku innych algorytmów, takich jak BT-FC, i stosowanie BT-LA nie zawsze się opłaca.



Nawracanie sterowane zależnościami

W przeszukiwaniu drzewa CSP możemy natrafić na porażkę wywołującą nawrót algorytmu BT, której przyczyną nie było jednak ostatnio dokonane przypisanie wartości, ale któryś z wcześniejszych kroków. W takim przypadku algorytm będzie próbował różnych możliwości, stale generując porażki, aż do momentu, kiedy nawróci dostatecznie głęboko, i zmieni wartość kluczowej zmiennej.

Jest możliwe wykrycie takiej sytuacji — gdy zbiór zmiennych wchodzących w więzy z bieżącą zmienną, tzw. zbiór konfliktowy (*conflict set*), nie obejmuje zmiennej ostatnio przypisanej. Wtedy nawrót mógłby zostać wykonany do ostatnio przypisanej zmiennej z tego zbioru. Algorytm taki zwany jest BJ (*backjumping*).

Prosty algorytm BJ ma znaczenie raczej tylko historyczne, ponieważ rozwiązuje problem, do którego nawet nie dopuszczają algorytmy spójności: BT-FC i dalsze. Jednak można skonstruować bardziej wyrafinowaną (i szerszą) definicję zbioru konfliktowego, jako zbioru tych zmiennych, których przypisane wartości spowodowały porażkę bieżącej zmiennej, wraz z później przypisanymi zmiennymi. Wersja BJ oparta na takiej definicji, zwana *conflict-directed backjumping* wprowadza dalsze usprawnienie procesu przeszukiwania.

Dynamiczne porządkowanie

Wspomnieliśmy wcześniej, że w większości problemów CSP trudno o rzeczywiste heurystyki wspomagające wybór „dobrych" ruchów na drzewie przeszukiwania. Istnieją jednak inne techniki wspomagające przeszukiwanie, związane z **dynamicznym porządkowaniem** (*dynamic ordering*) zarówno zmiennych, dla których opłaca się dokonywać przypisania w pierwszej kolejności, jak i wartości, które warto próbować wpierw.

Heurystyka **najbardziej ograniczonej zmiennej** (*most constrained variable*) (zwana również MRV, *minimum remaining values*), sugeruje wybór zmiennych z najbardziej ograniczonymi (licznymi) dziedzinami. Taki wybór daje największą szansę natrafienia na niespójności, i wynikających z nich redukcji problemu. Ta heurystyka dobrze współpracuje z algorytmem BT-FC.

Inną jest heurystyka **rzędu** (*degree heuristic*), sugerująca wybór zmiennej występującej w największej liczbie więzów z niepodstawionymi zmiennymi.

Po wyborze zmiennej przydaje się heurystyka **najmniej ograniczonej wartości** (*least constraining value*), preferująca wartości wykluczające najmniej wartości innych zmiennych.

Przeszukiwanie lokalne w zagadnieniach CSP

Możliwe jest jeszcze inne podejście do problemów CSP, polegające na mniej lub bardziej przypadkowym wyborze wstępnego przypisania wartości zmiennych, i następnie naprawienia go, jeśli naruszało jakieś więzy. Co więcej, proces tego naprawiania polega na przeszukiwaniu zachłannym, a więc niesystematycznym.

Jedną z heurystyk sprawdzających się w przeszukiwaniu lokalnym jest heurystyka zwana *min-conflict* i polegająca na losowym wyborze jednej ze zmiennych, których wartość narusza jakieś więzy, i przypisaniu jej innej wartości, wybranej tak, aby powodowała najmniej konfliktów (naruszeń więzów) z innymi zmiennymi.

Algorytmy przeszukiwania lokalnego oparte na powyższym schemacie sprawdzają się zaskakująco dobrze w wielu zagadnieniach CSP. Kluczowym ich elementem jest losowość, która pozwala uciec od maksimów lokalnych i innych pułapek, oraz natrafić na kluczową zmienną do poprawienia, lub pominąć niefortunnie wybraną zmienną, której wartość należy przypisać później.

Krótkie podsumowanie — pytania sprawdzające

1. Rozważ problem CSP z czterema zmiennymi: A, B, C, D z dziedzinami: $\{1, 2, 3\}$ dla każdej z nich, i zbiorem więzów podanym niżej. Narysuj graf więzów zadania, po czym spróbuj je rozwiązać metodą propagacji więzów (spójności łukowej). Pokaż każdy krok rozwiązania (bez rysunku). Przedstaw graf po zakończeniu propagacji więzów. Ile reprezentuje on możliwych rozwiązań problemu CSP? Zapisz jedno z nich.

   Zbiór więzów:
   $\mathcal{C} = \{C \neq D, B > D, B > C\}$

Literatura i materiały pomocnicze

1. Stuart Russell, Peter Norvig: Sztuczna inteligencja. Nowe spojrzenie, Wydanie IV, Tom 1, Wyd.Helion, 2023, rozdział 6.

2. Dobre wprowadzenie do technik rozwiązywania zagadnień CSP Romana Bartáka `http://ktiml.mff.cuni.cz/~bartak/constraints/constrsat.html`