2021

Metody Sztucznej Inteligencji. 3. Przeszukiwanie przestrzeni stanów

3. PRZESZUKIWANIE PRZESTRZENI STANÓW

WŁODZIMIERZ KASPRZAK

PRZESZUKIWANIE ŚLEPE, PRZESZUKIWANIE POINFORMOWANE, IDA*, RTA*

Przedstawione zostanie pojęcie „przestrzeni stanów” wyrażającej problem, który ma rozwiązać agent Sztucznej Inteligencji. Tak wyróżniony interfejs pomiędzy podsystemem sterowania agenta a problemem pozwala na definiowanie uniwersalnych strategii decyzyjnych przyjmujących postać przeszukiwania przestrzeni stanów (dla znalezienia ścieżki reprezentującej sekwencje akcji agenta) lub poszukiwania stanu docelowego. Podstawowe strategie przeszukiwania dzielimy na „ślepe” (np. przeszukiwanie w głąb, wszerz, z iteracyjnym pogłębianiem) i „poinformowane” (np. A*, IDA*, RTA*). Dla oceny jakości znalezionej ścieżki istotna jest znajomość kosztów akcji agenta, czyli kosztów przejść pomiędzy stanami.

Spis treści

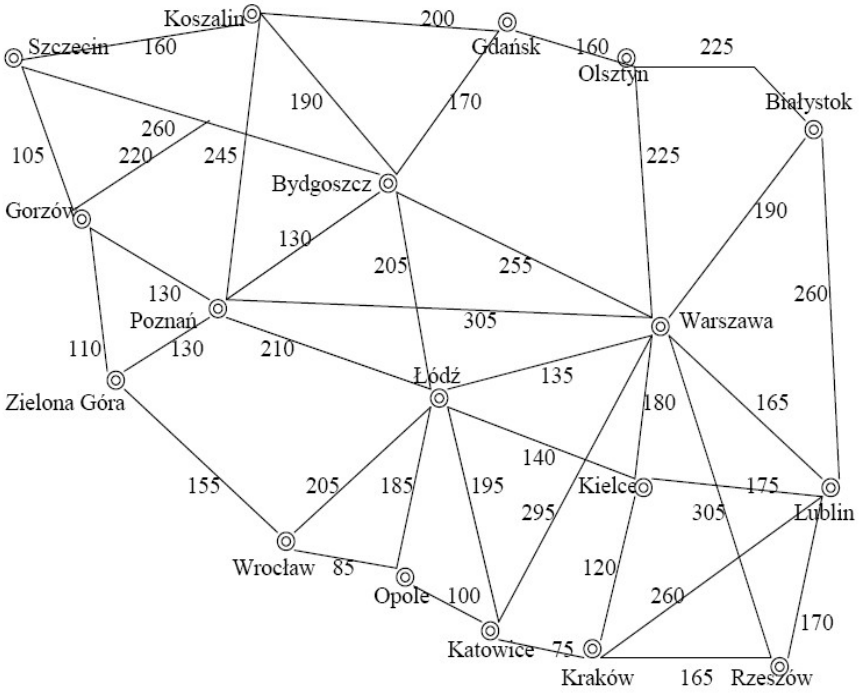
1 Przeszukiwanie przestrzeni stanów

1.1 Przestrzeń stanów

W tym rozdziale omówimy metodykę „przeszukiwania przestrzeni stanów" jako generalny (abstrakcyjny) sposób działania agenta realizującego cel. O ile tylko uda nam się wyrazić konkretny problem z konkretnej dziedziny w terminach tej metodyki to będziemy mogli natychmiast skorzystać z istniejącego sposobu rozwiązania problemu.

1.1.1 Przykład

Dany jest przykładowy problem: „przejazd ze Szczecina do Krakowa". Załóżmy, że aktualnie jesteśmy w Szczecinie i chcemy pojechać do Krakowa transportem drogowym. Celem działania jest: *dotrzeć do Krakowa*. Kluczową sprawą jest wyrażenie możliwych rozwiązań problemu w postaci **grafu stanów**, gdzie pojedynczy stan oznacza „*agent przebywa w danym mieście*". Dla uproszczenia wyróżnimy tylko większe miasta w Polsce (Rysunek 1). Możliwe **akcje** (operatory w przestrzeni stanów) oznaczają: „*przejazd z miasta A do miasta B*". Rozwiązaniem będzie każda ścieżka w grafie stanów prowadząca od stanu początkowego do stanu końcowego. W tym ostatnim spełniony jest **warunek zatrzymania** działania. Rozwiązaniem może być np. sekwencja akcji dla przejazdu: (Szczecin, Bydgoszcz, Łódź, Katowice, Kraków).



Rysunek 1 : Przykład grafu stanów dla problemu „przejazdu z miasta A do miasta B".

1.2 Problem przeszukiwania

Teraz podamy definicję **problemu przeszukiwania**. Problem wyrażony jest w postaci 4 pojęć (rodzajów danych):

1. Zbiór stanów S z wyróżnionym stanem początkowym; np. Szczecin $\in$ S, o znaczeniu: „*agent jest w Szczecinie*" ;

2. Akcje (operacje w przestrzeni stanów), A = {a_1, a_2, ... a_n}, i funkcja następnika stanu:

[(*stan_we*, *a*) $\rightarrow$ *stan_wy*] $\in$ S $\times$ A $\times$S

Np.: (Szczecin, *„ze Szczecina do Bydgoszczy”*) $\rightarrow$ Bydgoszcz

3. Warunek osiągnięcia celu T, który jest spełniony w każdym stanie końcowym. Może on bezpośrednio referować stan, np., T: *s* = „*Kraków*", lub badać własność stanu, np. T: *Szachmat*(*s*) = true .
4. Koszt każdej akcji i sposób obliczenia kosztu rozwiązania (najczęściej jest to suma kosztów akcji tworzących rozwiązanie); np., suma odległości w km pomiędzy miastami, liczba wykonanych akcji, itp.

Niech*, c*(*x ,a, y*), oznacza koszt akcji. W algorytmach realizujących różne strategie przeszukiwania zazwyczaj postuluje się, aby koszt akcji był nieujemny, $c \geq 0$.

Rozwiązaniem problemu jest sekwencja akcji (ścieżka w przestrzeni stanów) prowadząca od stanu początkowego do końcowego.

Uwaga: przedstawiliśmy tu definicję problemu **jednostanowego**, w którym każdy węzeł drzewa przeszukiwania odpowiada jednemu stanowi w przestrzeni problemu. Taki przypadek zachodzi wtedy, gdy środowisko działania agenta jest w pełni obserwowalne. W sytuacji, gdy środowisko nie jest w pełni obserwowalne mamy do czynienia z problemem wielostanowym – z uwagi na niepewność w jakim stanie znajduje się środowisko jeden węzeł drzewa przeszukiwania odpowiada wielu stanom problemu.

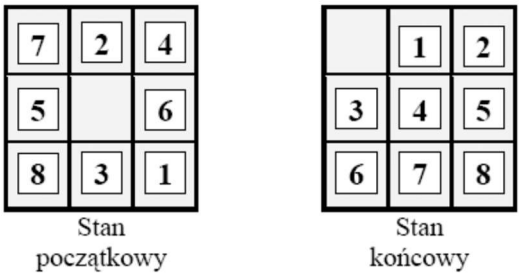
1.3 Wybór przestrzeni reprezentacji problemu

Rzeczywisty problem (świat) jest bardzo złożony a podana powyżej definicja problemu przeszukiwania obejmuje jedynie niezbędne elementy opisu tego świata. Podczas projektowania systemu sztucznej inteligencji trzeba stworzyć uproszczony (abstrakcyjny) model świata, w którym:

- Abstrakcyjny stan odpowiada wielu sytuacjom rzeczywistym;
- Abstrakcyjna akcja odpowiada wielu rzeczywistym akcjom; np., „(Szczecin, „przejazd”) $\rightarrow$ Bydgoszcz" reprezentuje zbiór możliwych dróg, objazdów, miejsc odpoczynku, itd.
- Abstrakcyjne rozwiązanie odpowiada zbiorowi rzeczywistych dróg, które w rzeczywistym świecie prowadzą do celu.

Tym samym każde z abstrakcyjnych pojęć stanowi zwykle uproszczenie oryginalnego pojęcia w rzeczywistym świecie.

Przykład. Definicja problemu przeszukiwania dla świata „8-puzzli” (Rysunek 2).



Rysunek 2. Przykład stanów dla problemu „8-puzzli”.

Stany problemu reprezentują konfiguracje 8 numerowanych kafelków (płytek) w kwadratowym obszarze o rozmiarze 3 $\times$ 3. Pojedyncza akcja polega na „przemieszczeniu” pustego miejsca w lewo,

prawo, na górę lub na dół (dualnie: jest to przesunięcie kafelka sąsiadującego z pustym miejscem). Warunek zatrzymania (stopu) jest podany jawnie w postaci stan końcowego, w którym numery kafelków uporządkowane są w kolejności rosnącej. Koszt każdej akcji wynosi 1.
Rozmiary różnych wersji „świata puzzli" mogą być dowolnie duże (oznaczmy rozmiar przez N). Problem polega na optymalnym (w sensie minimalnego kosztu potrzebnych akcji) uporządkowaniu „świata N-puzzli". Rozwiązanie takiego problemu jest NP-trudne, czyli o złożoności obliczeniowej powyżej wielomianowej względem N. Z naszego punktu widzenia problem „świata N-puzzli" stanowi dogodną ilustrację dla porównywania różnych **strategii przeszukiwania** przestrzeni stanów.

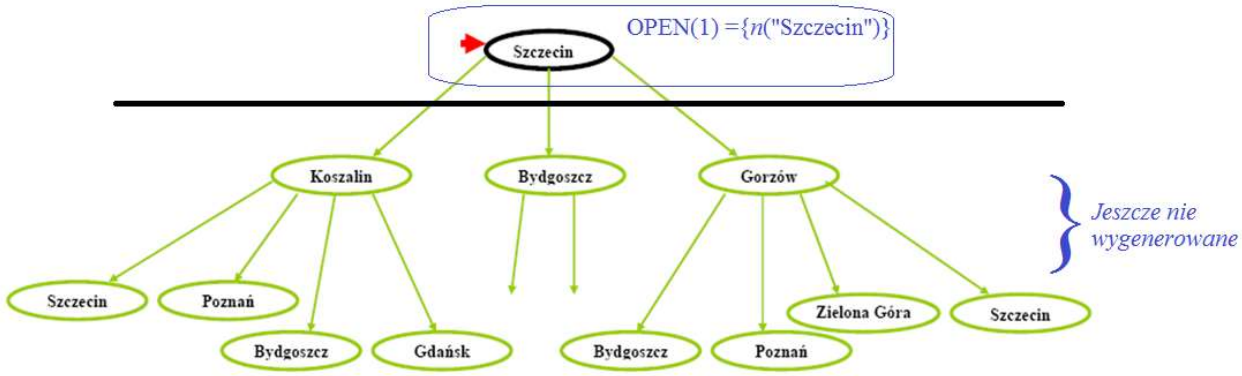
1.4 Drzewo decyzji dla przeszukiwania przestrzeni stanów)

W zasadzie każdą strategię przeszukiwania dyskretnej przestrzeni problemu możemy wyrazić jako wizytowanie (przejście) pewnego **drzewa decyzji**. Należy zauważyć, że przestrzeń stanów ma zazwyczaj topologie **grafu** co sprawia, że możliwe są sekwencje akcji stanowiące cykle w przestrzeni stanów (tzn. powrót do już uprzednio wizytowanego stanu). Strategie przeszukiwania, stosujące reprezentację drzewa decyzyjnego albo będą odporne na takie sytuacje, z uwagi na stosowany algorytm, albo też muszą je wykrywać i rozwiązywać.
Podczas przeszukiwania przestrzeni stanów reprezentujemy wyniki częściowe w postaci węzłów **drzewa przeszukiwania**, a alternatywne akcje przeprowadzają aktualny węzeł w jeden z możliwych węzłów-następników tego drzewa. Rozwijamy drzewo przeszukiwania począwszy od korzenia, reprezentującego *stan początkowy*, poprzez węzły pośrednie do liści drzewa, z których przynajmniej jeden reprezentuje *stan końcowy,* czyli stanowi cel przeszukiwania.
W problemie „jedno-stanowym" *węzeł* drzewa przeszukiwania odpowiada pojedynczemu stanowi problemu, ale często mamy do czynienia z problemem, gdy z uwagi na niepełną informację o sytuacji agenta jeden węzeł drzewa decyzji odpowiada „wielu stanom" w przestrzeni stanów.
Aktualny zbiór węzłów-liści, posiadających jeszcze niewizytowane następniki, to zbiór węzłów gotowych do rozwinięcia, utożsamiany z tzw. **skrajem** drzewa (często w algorytmach przeszukiwania nazywany zbiorem OPEN). Sposób porządkowania węzłów w skraju i tym samym sposób wyboru następnego rozwijanego węzła wyraża określoną strategię przeszukiwania.

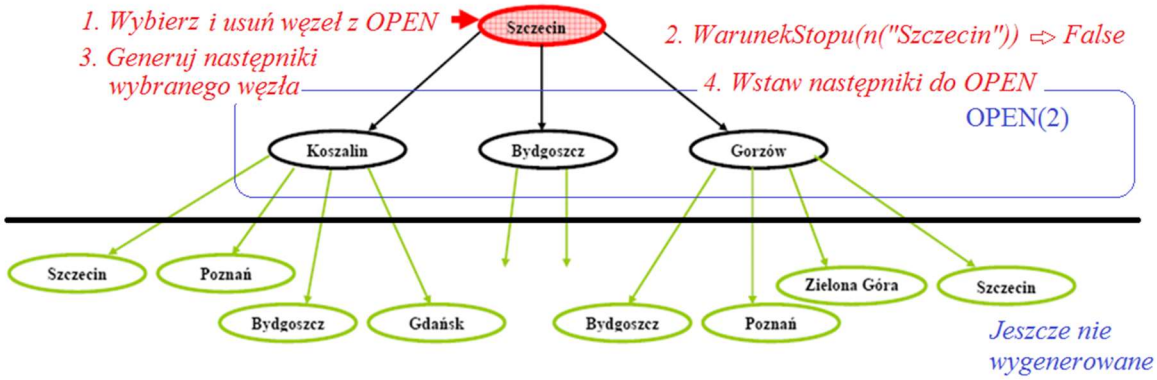
1.4.1 Przykład

Początkowe drzewo decyzyjne dla problemu „powrót ze Szczecina do Krakowa" (Rysunek 3). Węzeł początkowy o etykiecie „Szczecin" reprezentuje sytuację (jest to także jeden stan problemu) „*agent jest w Szczecinie*". Jest to na początku jedyny węzeł w skraju, który jest w pierwszym kroku wybierany do rozszerzenia. Pozostałe widoczne węzły reprezentują możliwe następniki, które zostaną wygenerowane na podstawie opisu problemu (na podstawie grafu problemu) z chwilą wykonania określonej akcji. W pierwszym kroku generowane są następniki węzła „Szczecin", czyli węzły: Koszalin, Bydgoszcz, Gorzów (Rysunek 4). Załóżmy, że w kolejnym kroku, zgodnie z przyjętą strategią przeszukiwania, wybierany jest węzeł „Koszalin". Nie spełnia on warunku celu więc generowane są jego następniki reprezentujące stany: Szczecin, Poznań, Bydgoszcz, Gdańsk (Rysunek 5). W kolejnych krokach postępujemy podobnie, wybierając węzły znajdujące się w skraju drzewa,

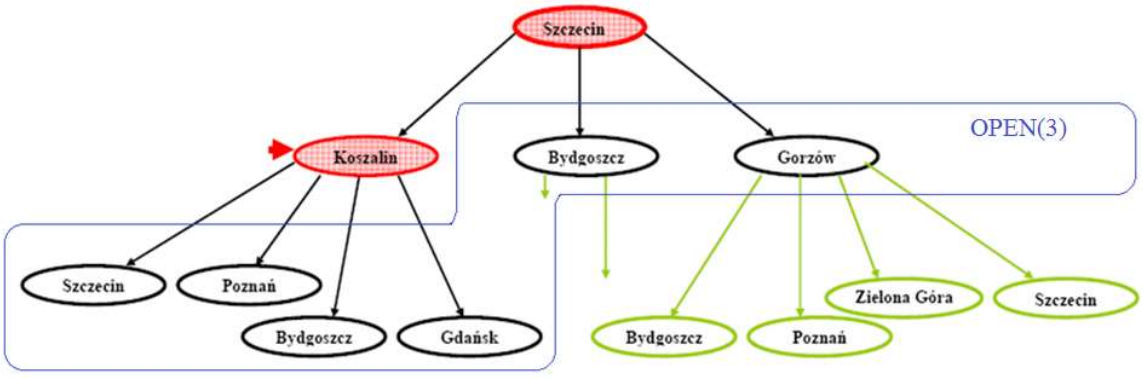
sprawdzając dla nich warunek zatrzymania i w przypadku niespełnienia warunku – generując następników węzła.



Rysunek 3. Przykład początkowego drzewa przeszukiwania dla problemu „powrót ze Szczecina do Krakowa”.



Rysunek 4. Postać drzewa przeszukiwania po wybraniu węzła „Szczecin”.



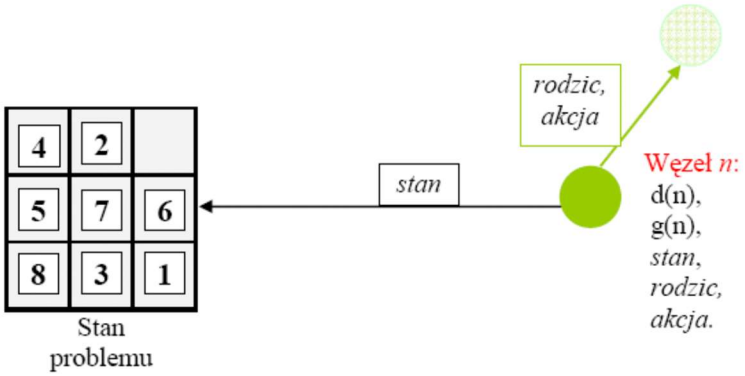
Rysunek 5. Postać drzewa przeszukiwania po wybraniu węzła „Koszalin”.

Zauważmy, ze w powyższym przykładzie możemy wielokrotnie generować węzły reprezentujące ***ten sam*** stan problemu. Nie jest to sytuacja pożądana, gdyż może ona prowadzić do zamkniętych cykli (powtarzanie stanu na tej samej ścieżce), a te z kolei mogą być powtarzane nieskończenie wiele razy, co prowadzi do nieskończenie długiej ścieżki. Dla takich problemów będziemy musieli rozszerzyć algorytm przeszukiwania o wykrywanie takich sytuacji.

1.5 Drzewo przeszukiwania a przestrzeń stanów

Wyjaśnijmy jeszcze *różnicę* pomiędzy **węzłem drzewa** przeszukiwania a **stanem** problemu. Stan jest abstrakcyjną reprezentacją fizycznej konfiguracji świata. Węzeł *n* jest strukturą danych i elementem drzewa przeszukiwania. Dla przykładu węzeł może zawierać (Rysunek 6):

- odniesienie do stanu lub zbioru stanów problemu,
- wskaźnik do węzła rodzica,
- akcję, wykonaną w sytuacji reprezentowanej węzłem rodzica,
- koszt sekwencji akcji prowadzącej do tego węzła g(*n*) i głębokość węzła d(*n*).



Rysunek 6. Przykład struktury węzła w drzewie przeszukiwania.

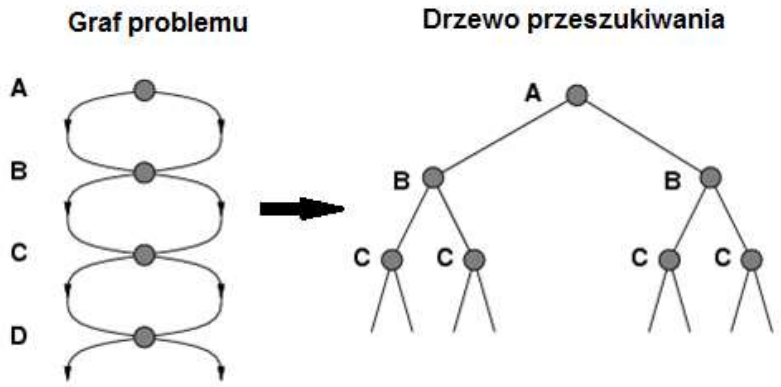
1.6 Unikanie cykli w grafie

Jak zauważyliśmy we wcześniejszym przykładzie, w procesie przeszukiwania drzewa może dochodzić do prób generowania węzłów reprezentujących ponownie te same stany problemu, co już reprezentowane w drzewie decyzji. Przestrzeń problemu zazwyczaj przyjmuje postać grafu (istnieją różne ścieżki łączące te same dwa stany) lub też istnieją dwukierunkowe łuki w grafie. Wyróżnimy dwa przypadki generowania równoważnych węzłów (w sensie reprezentowania tego samego stanu problemu):

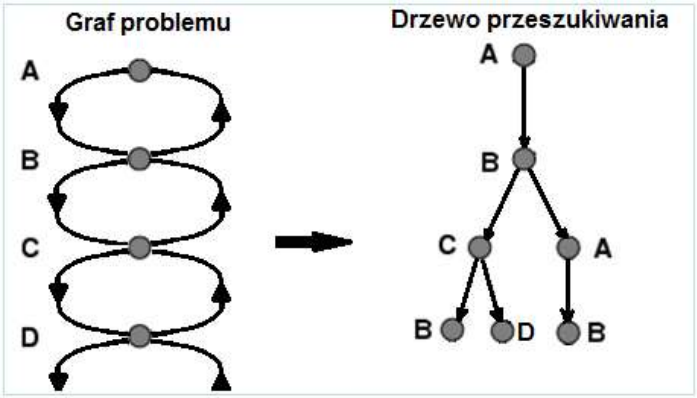
1. określony stan jest wielokrotnie rozwijany na różnych ścieżkach drzewa przeszukiwania (Rysunek 7) lub
2. stan jest powtórnie „wizytowany” na tej samej ścieżce drzewa przeszukiwania (Rysunek 8).

Pierwszy przypadek może prowadzić do niepotrzebnego rozwijania nadmiarowych ścieżek i wpływać negatywnie na efektywność strategii przeszukiwania.

Drugi przypadek odpowiada powstaniu pętli na ścieżce rozwiązania, co może prowadzić do ścieżki o nieskończonej długości i braku rozwiązania. Brak wykrywania takich sytuacji może prowadzić do wielokrotnego powtarzania tej samej sekwencji akcji, czasem nieskończoną ilość razy.



Rysunek 7. Wielokrotne (równoważne) węzły na różnych ścieżkach drzewa przeszukiwania



Rysunek 8. Powtórny węzeł na tej samej ścieżce (pętla)

Powyższe sytuacje wymagają modyfikacji bazowego algorytmu przeszukiwania przestrzeni stanów. Dla przykładu, przeciwdziałamy obu powyższym niepożądanym przypadkom dokonując następujących zmian i uzupełnień w algorytmie:

1. należy zapamiętać wszystkie wcześniej rozwijane węzły (w tym celu wprowadzamy zbiór węzłów CLOSED),
2. każdy nowo generowany węzeł jest porównywany z węzłami znajdującymi się w zbiorach OPEN i CLOSED; po ewentualnym stwierdzeniu identyczności stanów reprezentowanych dwoma różnymi węzłami, jeden z węzłów (domyślnie ten „gorszy” lub kolejny) jest eliminowany.

2 Strategie ślepego przeszukiwania

Ślepe przeszukiwanie (ang. *uninformed search*) wykorzystuje jedynie informację zawartą w sformułowaniu problemu.

- Przeszukiwanie wszerz (*breadth-first search, BFS*) - skraj drzewa jest kolejką FIFO (first-in first-out).

- Przeszukiwanie z jednolitą funkcją kosztu (*uniform-cost search*) - węzły uporządkowane są w skraju według niemalejących sumarycznych kosztów dotychczasowych akcji prowadzących do danego węzła.
- Przeszukiwanie w głąb (*depth-first search, DFS*) - skraj jest stosem LIFO (last-in first-out).
- Przeszukiwanie z ograniczoną głębokością DLS (*depth-limited search*) - tak jak w głąb do zadanego ograniczenia *l*, ale węzły na poziomie ograniczenia nie mają już następników.
- Iteracyjne pogłębianie IDS (*iterative deepening search*) - kolejne, niezależne od siebie wykonywanie przeszukiwań DLS dla coraz większych wartości ograniczenia głębi (*l*=0,1,2,...), aż do momentu znalezienia celu.

2.1 Kryteria oceny strategii przeszukiwania

Przyjęta strategia przeszukiwania drzewa wyraża się w sposobie wyboru kolejno rozwijanych (wizytowanych) węzłów. Oczywiście dobrze byłoby móc porównać działanie różnych strategii. Podamy tu cztery podstawowe **kryteria** oceny strategii.

1. **Zupełność**;

Zupełność strategii przeszukiwania oznacza, że jeżeli istnieje rozwiązanie problemu, to zostanie ono zawsze znalezione.

2. **Złożoność czasowa**: czas przeszukiwania mierzony całkowitą liczbą węzłów wizytowanych podczas przeszukiwania;
3. **Złożoność pamięciowa**: maksymalna liczba węzłów jednocześnie rezydujących w pamięci programu przeszukiwania;
4. **Optymalność**;

Strategia optymalna to taka, która zawsze znajduje najlepsze rozwiązanie.
W przypadku kryteriów złożoności czasowej i pamięciowej interesują nas nie tyle złożoności dla konkretnego problemu, wyznaczone po procesie przeszukiwania, lecz oczekiwane złożoności, szacowane dla danej strategii na podstawie znajomości rozmiaru problemu. Przyjęło się wyrażać rozmiar problemu za pomocą następujących parametrów:

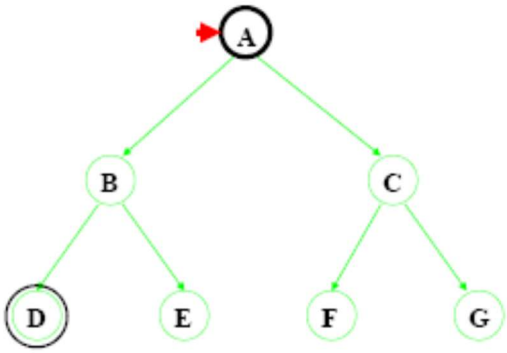
- *b:* maksymalne rozgałęzienia drzewa przeszukiwania,
- *d:* głębokość, na której znajduje się najtańsze rozwiązanie,
- *m*: maksymalna głębokość drzewa przeszukiwania.

2.2 Przeszukiwanie wszerz – BFS

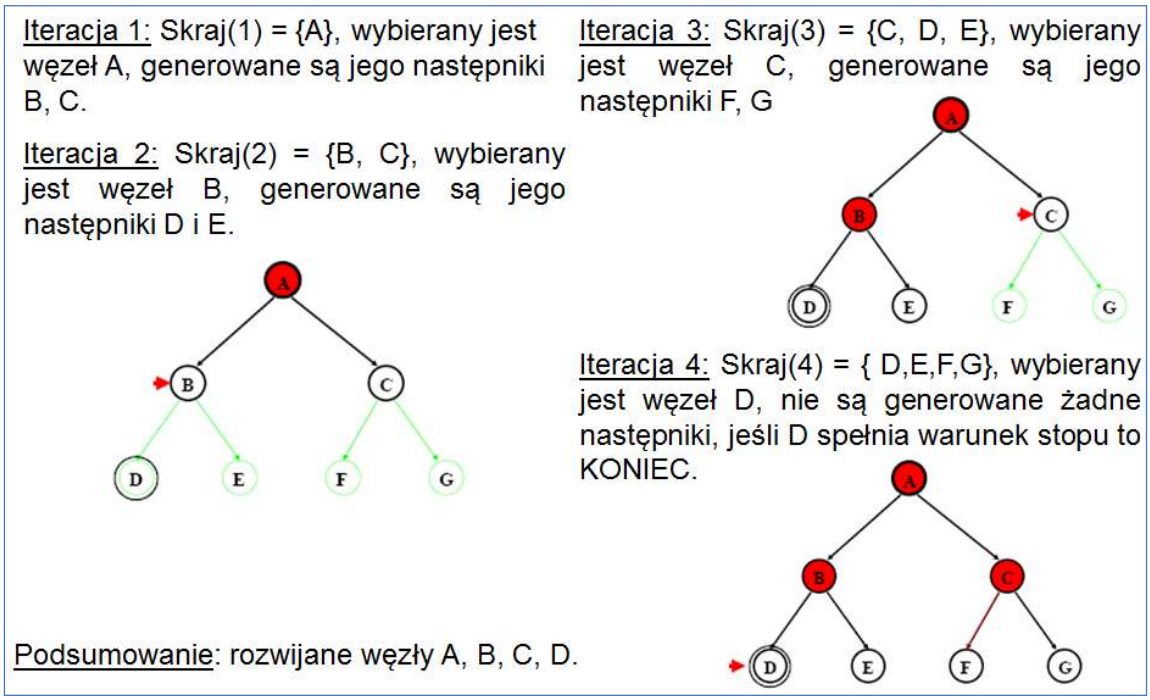
W tej strategii przeszukiwania przestrzeni stanów rozwijany jest zawsze najpłytszy dotąd nie rozwinięty węzeł.
Implementacja strategii przeszukiwania wszerz polega na reprezentowaniu aktualnego skraju drzewa (tzw. zbiór OPEN) w postaci kolejki FIFO – nowo dodawane węzły-następniki ustawiane są zawsze na końcu kolejki a pobieranie węzłów (w celu rozwinięcia) ma miejsce na początku kolejki.

Przykład.

Dana jest przestrzeń stanów o topologii drzewa (Rysunek 9). Jest to „problem jednostanowy” a dodatkowo też jest bezpieczny, gdyż nie ma niebezpieczeństwa powielania równorzędnych węzłów podczas przeszukiwania. Drzewo decyzji będzie stopniowo „odtwarzać” topologie przestrzeni stanów - możliwe jest drzewo o trzech poziomach węzłów z węzłem początkowym reprezentującym stan A.



Rysunek 9. Przykład prostej przestrzeni stanów o topologii drzewa.



Rysunek 10. Kolejne kroki przeszukiwania wszerz drzewa z rysunku 9.

Kolejne kroki decyzyjne i historię decyzji ilustrujące strategię „przeszukiwania wszerz” pokazano na Rysunek 10. Węzeł A jest węzłem początkowym, a węzeł D – końcowym. Kolejność wybieranych węzłów to (A, B, C, D), a znaleziona ścieżka rozwiązania to (A, B, D).

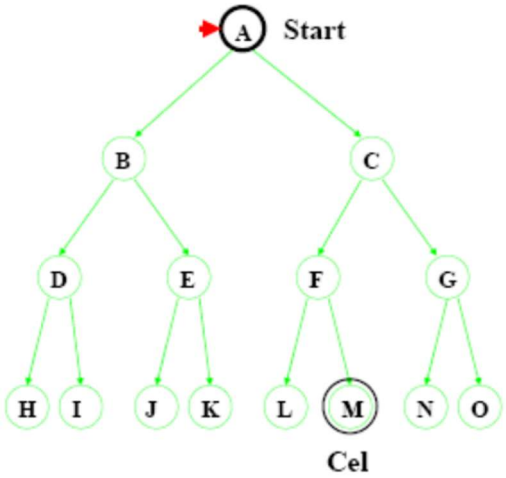
2.3 Przeszukiwanie w głąb – DFS

Zasada strategii przeszukiwania w głąb: rozwijany jest najgłębszy, dotąd nie rozwinięty węzeł. Implementacja *skraju drzewa* (czyli tzw. listy OPEN) ma postać stosu LIFO; nowe następniki

ustawiane są na początku stosu i są rozwijane w pierwszej kolejności. Taka kolejność rozwijania węzłów odpowiada, np. lewostronnemu obejściu drzewa.

Przykład

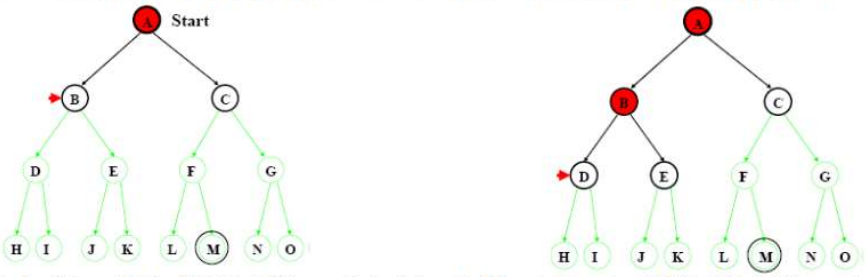
Dana jest przestrzeń stanów do postaci drzewa o czterech poziomach stanów i stanie początkowym A (Rysunek 11). Również w tym przypadku powstające drzewo decyzji będzie stopniowo odtwarzać topologię przestrzeni stanów. Strategia przeszukiwania w głąb znajduje ścieżkę od stanu A (stan początkowy) do stanu/węzła docelowego M, odwiedzając (wybierając „po drodze") następujące stany i generując odpowiednio (Rysunek 12, Rysunek 13): A, B, D, H, I, E, J, K, C, F, L, M.



Rysunek 11. Przestrzeń stanów ilustrująca strategię przeszukiwania w głąb

Krok 1: Skraj(1)={A}, wybór A.

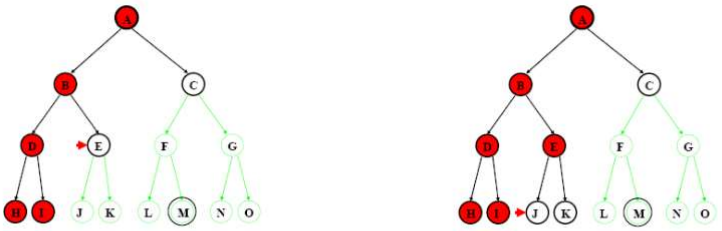
Krok 2: Skraj(2)={B,C}, wybór B. →Krok 3: Skraj(3)={D,E,C}, wybór D.



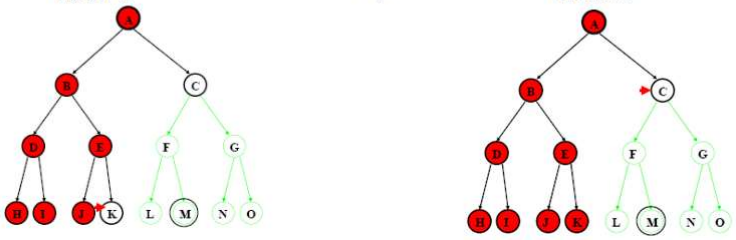
Krok 4: Skraj(4)={H,I,E,C}, wybór H. → Krok 5: skraj(5)={I,E,C}, wybór I.

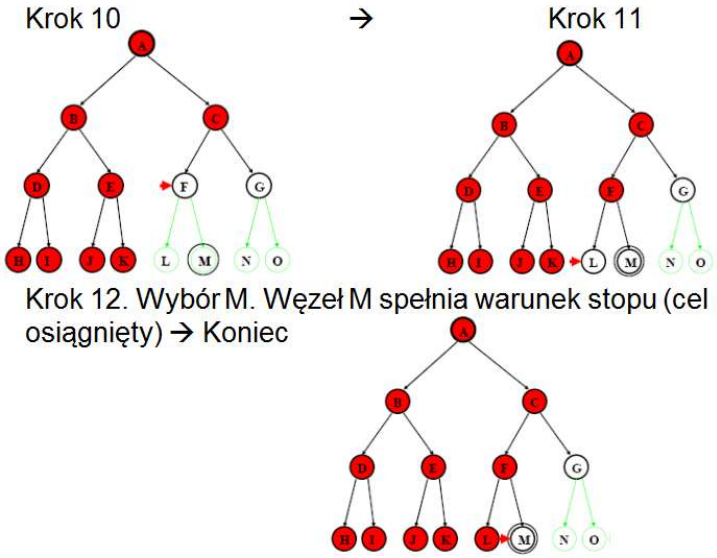
Rysunek 12. Początkowe 5 kroków przeszukiwania w głąb przestrzeni stanów z rysunku 11

Krok 6 → Krok 7



Krok 8 → Krok 9





Rysunek 13. Kontynuacja (kroki 6-12) przeszukiwania przestrzeni z rysunku 11

2.4 Przeszukiwanie z jednolitą funkcją kosztu - UCS

Strategia **przeszukiwania z jednolitą funkcją kosztu**: zawsze rozwija dotąd nie rozwinięty węzeł (znajdujący się w skraju drzewa decyzji) o najniższym koszcie z dotychczasowych. *Skraj* drzewa stanowi wtedy kolejkę uporządkowaną według kosztu ścieżki (sekwencji akcji) prowadzącej do danego węzła. Ta strategia jest równoważna przeszukiwaniu wszerz, jeżeli koszty wszystkich akcji są równe.

Własności.

- Zupełność: tak, jeżeli koszt każdej akcji ≥ ε, gdzie ε ≥ 0.
- Złożoność obliczeniowa: liczba węzłów o koszcie $g(n)$ ≤ koszt optymalnego rozwiązania, $O(b^{(C^*/\varepsilon)})$, gdzie C^* to koszt optymalnego rozwiązania.
- Złożoność pamięciowa: liczba węzłów o koszcie $g(n)$ ≤ koszt optymalnego rozwiązania: $O(b^{(C^*/\varepsilon)})$.
- Optymalność: tak, w sensie minimalizacji kosztu, gdyż węzły rozwijane są zawsze w kolejności zwiększającego się kosztu $g(n)$.

2.5 Przeszukiwanie z ograniczoną głębokością – DLS

Jest to odmiana przeszukiwania w głąb z ograniczeniem nałożonym na głębokość węzła, co oznacza, że jeśli ***l*** jest ograniczeniem głębokości węzła to (z punktu widzenia strategii) węzły na głębokości ***l*** nie posiadają następników.

Strategia nie jest ani zupełna ani optymalna. Jeśli każde rozwiązanie jest dane na głębokości większej niż ***l*** to nie zostanie znalezione żadne rozwiązanie (przeszukiwanie niezupełne). Przeciwnie, jeśli istnieją rozwiązania na ścieżkach krótszych niż ***l*** to nie ma gwarancji, że znalezione rozwiązanie jest optymalne, nawet w sensie minimalnej długości ścieżki. Jest to efektem stosowania strategii przeszukiwania w głąb w ramach ograniczenia do głębokości ***l*** (strategia nieoptymalna).

2.6 Przeszukiwanie z iteracyjnym pogłębianiem – IDS

Strategia polega na iteracyjnym wywoływaniu przeszukiwania z ograniczoną głębokością dla różnych wartości ograniczenia głębokości l (l=0, 1, 2, …), tak długo, dopóki nie zostanie znalezione rozwiązanie, czyli ścieżka prowadząca do węzła końcowego (węzła spełniającego warunek stopu). Funkcja implementująca strategię przeszukiwania IDS podana została w Tabela 1.

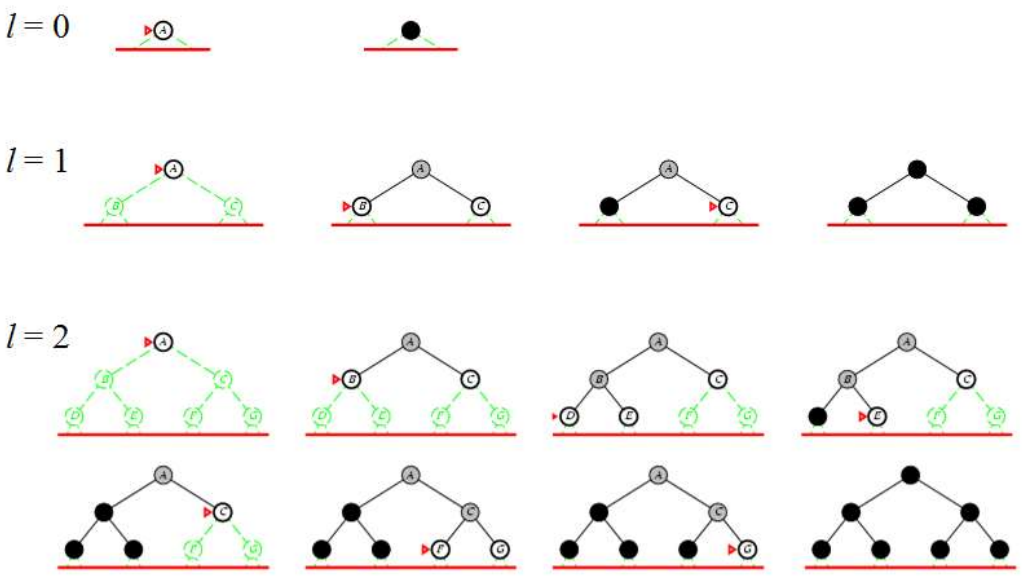
Tabela 1. Strategia „iteracyjnego pogłębiania"

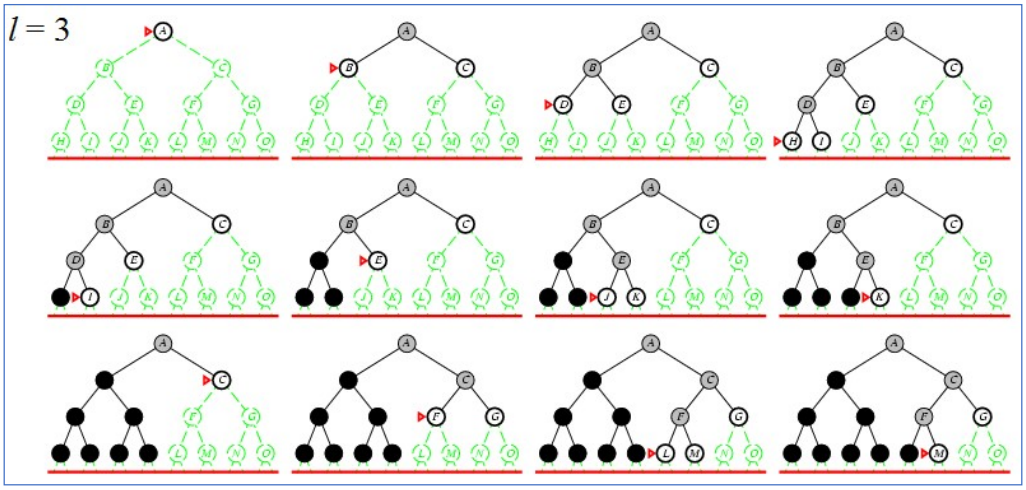
function Iteracyjne_pogłębianie(*problem*), **Wynik**: *ścieżka*
{ **for** (int *głębia*=0; *głębia* < ∞ ; *głębia* ++) {
wynik ←**Przeszukiwanie_z_ograniczoną_głębokością**(
problem, *głębi*a)
if (Typ(*wynik*) = Ścieżka_końcowa) **then return** *wynik*;
}
}

Przykład
Dla przestrzeni stanów podanej na Rysunek 9. Przykład prostej przestrzeni stanów o topologii drzewa.Rysunek 11, strategia IDS wymaga 4 iteracji (Rysunek 14).

2.7 Porównanie strategii przeszukiwania

W Tabela 2 określono zachowanie się strategii ślepego przeszukiwania w oparciu o wcześniej przyjęte kryteria. Dla przypomnienia parametry (symbole) oznaczają: *b* – stopień rozgałęzienia, *d* – długość ścieżki rozwiązania, *m* – maksymalna głębokość drzewa, *l* – ograniczenie głębokości drzewa, *C** - koszt optymalnego rozwiązania, *ε* - najmniejszy koszt akcji.





Rysunek 14. Kolejne iteracje w metodzie przeszukiwania IDS wykonywane dla przestrzeni z rysunku 11

Tabela 2. Porównanie strategii ślepego przeszukiwania

Strategia: Kryterium	„wszerz”	„z jednorodnym kosztem”	„w głąb”	„z ograniczoną głębokością”	„z iteracyjnym pogłębianiem”
Zupełny ?	Tak	Tak	Nie	Nie	Tak
Złożoność czasowa	$O(b^{d+1})$	$O(b^{[C^*/\varepsilon]})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Złożoność pamięciowa	$O(b^{d+1})$	$O(b^{[C^*/\varepsilon]})$	$O(bm)$	$O(bl)$	$O(bd)$
Optymalny ?	Tak	Tak	Nie	Nie	Tak

Iteracyjne pogłębianie (IDS) znajduje zawsze optymalną ścieżkę, posiada liniową złożoność pamięciową i nie potrzebuje dużo więcej czasu niż inne ślepe algorytmy przeszukiwania drzewa. Dlatego stanowi dogodną alternatywę dla **przeszukiwania z jednorodnym kosztem.**

3 Przeszukiwanie poinformowane

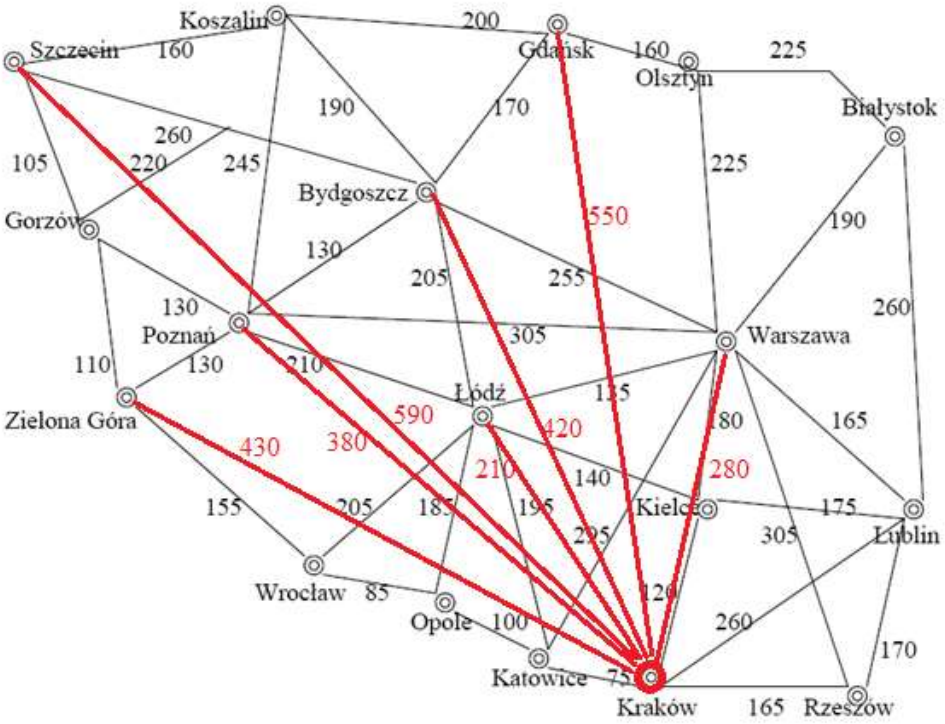
Strategia przeszukiwania jest wyznaczona sposobem wyboru kolejno rozwijanych węzłów w drzewie przeszukiwania, tworzonym dla rozwiązania problemu. Istnieje kilka „ślepych” strategii poszukiwania, takich jak: przeszukiwanie wszerz, przeszukiwanie w głąb, przeszukiwanie z jednorodną funkcją kosztu, przeszukiwanie z ograniczoną głębokością i iteracyjnym pogłębianiem. Określenie „ślepy” oznacza w tym przypadku, że w żadnym przypadku na wybór następnego węzła nie wpływa informacja o stanie końcowym. Jest to ich zasadnicza niedogodność, która sprawia, że zwykle nie osiągają one wystarczającej efektywności obliczeniowej czy pamięciowej, a nawet niektóre z nich nie są optymalne.

Z analizy słabości strategii przeszukiwania ślepego wynika wniosek: należy użyć funkcji oceny dla węzłów drzewa przeszukiwania, która uwzględniałaby (abstrakcyjnie zdefiniowaną) „odległość” węzła od węzła utworzonego dla stanu docelowego. Założenie o istnieniu takiej miary kosztów

„resztkowych” na ścieżce rozwiązania pozwala zaproponować strategie tzw. przeszukiwania **poinformowanego**. Oszacowanie kosztów resztkowych pozwoli ocenić na ile „obiecujący” z punktu widzenia celu jest dany węzeł i wybrać (rozwinąć) najbardziej „obiecujący” nierozwinięty węzeł.

Przykład

Przykład tworzenia składowej heurystycznej kosztu *h*(*n*) dla problemu „powrót do Krakowa”, wyrażającej odległość w linii prostej od danego miasta do Krakowa, zilustrowano na Rysunek 15.



Rysunek 15. Mapa rzeczywistych odległości pomiędzy stanami problem (wyróżnionymi miastami), a składowa heurystyczna dla wybranych stanów to odległość w linii prostej do stanu końcowego “Kraków”.

Zwróćmy uwagę na fakt, że sam sposób szacowania kosztów resztkowych sprawia, że są one optymistyczne, tzn. nie są większe od rzeczywistych kosztów dojazdu do celu (Tabela 3).

Tabela 3. Wartości heurystyczne kosztów resztkowych dla problemu „powrót do Krakowa”.

Białystok	440
Bydgoszcz	420
Gdańsk	550
Gorzów	500
Katowice	70
Kielce	110
Koszalin	580
Kraków	0
Lublin	230
Łódź	210
Opole	150
Poznań	380
Rzeszów	150
Olsztyn	460
Szczecin	590
Warszawa	280
Wrocław	240
Zielona Góra	430

3.1 Strategie „najpierw najlepszy” (*best first*)

W ogólności, rozpatrujemy tu funkcję oceny węzła (typowo jest to *koszt*) złożoną z 2 części:

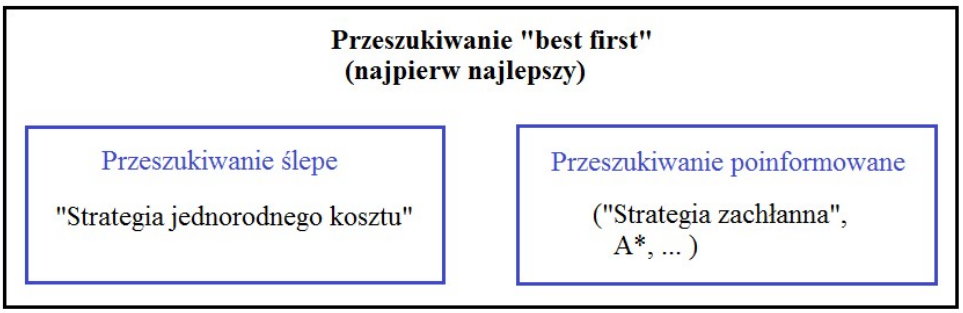
$$f(n) = g(n) + h(n),$$

gdzie $g(n)$ oznacza koszt dotychczasowej ścieżki a $h(n)$ oznacza przewidywany koszt resztkowy pozostałej drogi z węzła n do celu. Implementacja każdej strategii, należącej do kategorii „najpierw najlepszy”, polega na porządkowaniu węzłów w skraju według zwiększającej się wartości kosztu węzła $f(n)$. W zależności od postaci funkcji oceny rozpatrzymy trzy odmiany strategii „najpierw najlepszy” (Rysunek 16):

1. strategia niepoinformowana jednolitego kosztu: $f(n)= g(n)$,
2. strategia poinformowana zachłanna z heurystyką („najbliższy celowi najpierw”): $f(n) = h(n)$,
3. strategia poinformowana zwana „przeszukiwaniem A*” : $f(n) = g(n) + h(n)$.

Strategia **jednolitego kosztu** (względnie dualnie - zysku), nazywana też strategią **równomiernego** kosztu, została już omówiona w punkcie 2.4, gdyż jest to odmiana ślepego przeszukiwania. Posługuje się ona funkcją oceny węzła n, w której uwzględnia się jedynie koszty dotychczasowych akcji (na ścieżce od węzła początkowego do węzła n). Nie zalicza się jej do strategii poinformowanego przeszukiwania. Przeszukiwanie poinformowane ma miejsce wtedy, gdy posługujemy się w funkcji oceny oszacowaniem kosztów resztkowych $h(n)$ (nazywanych **heurystyką**).

Pozostałe dwa przypadki przeszukiwania „*best first*” należą do strategii poinformowanych, gdyż korzystają z oszacowań kosztów resztkowych dla każdego węzła przestrzeni przeszukiwania. Zostaną one omówione w następnych punktach.



*Rysunek 16. Klasyfikacja strategii "najpierw najlepszy" ("*best first*")*

3.2 Strategia zachłanna z heurystyką („najbliższy celowi najpierw")

W tej strategii funkcja oceny węzła przyjmuje postać:

$$f(n) = h(n) ,$$

czyli składa się wyłącznie ze składowej heurystycznej (oszacowania kosztów resztkowych przejścia z węzła *n* do celu). Zakładamy w niej, że najlepszy węzeł to ten, który jest „najbliższy celowi". Dotychczas poniesione koszty dojścia do aktualnego węzła nie mają żadnego znaczenia dla decyzji wyboru. Szczegółowy algorytm implementacji przeszukiwania stanów dla strategii zachłannej podano w Tabela 4. Dzięki istnieniu zbioru CLOSED, do którego wstawiane są wizytowane węzły, i dzięki krokowi 6, uniemożliwiającemu wielokrotne dodawanie do drzewa decyzyjnego równoważnych węzłów (tzn. reprezentujących ten sam stan lub podzbiór stanów problemu), algorytm efektywnie realizuje przeszukiwanie przestrzeni o ogólnej postaci grafu.

Tabela 4. Strategia „zachłanna" przeszukiwania przestrzeni stanów

1	INIT: Pobierz węzeł startowy s i umieść go w zbiorze OPEN. Ustaw $f(s) = h(s)$
2	Pobierz z OPEN najlepszy węzeł n (o najmniejszym koszcie $f(n)$) i przenieś go do CLOSED
3	JEŚLI (n jest węzłem końcowym) TO zakończ i zwróć g(n) oraz całą ścieżkę od s do n.
4	Znajdź węzły następców n - niech będą nimi: $n_1' \ldots n_k'$.
5	Dla każdego z następców $n_1' \ldots n_k'$ wyznacz jego koszt $f_i = h(n_i')$
6	Dla każdego z węzłów $n_1' \ldots n_k'$:
	JEŚLI (n_i' nie należy do zbioru OPEN ani do CLOSED) TO dodaj go do zbioru OPEN i ustaw: $f(n_i') = f_i$.
7	Powtórz od kroku 2.

Przykład.

Strategia „zachłanna" dla problemu „powrót ze Szczecina do Krakowa". Załóżmy, że stany problemu podano na Rysunek 15, a liczby przy łukach reprezentują koszty akcji przejazdów z miasta do miasta.

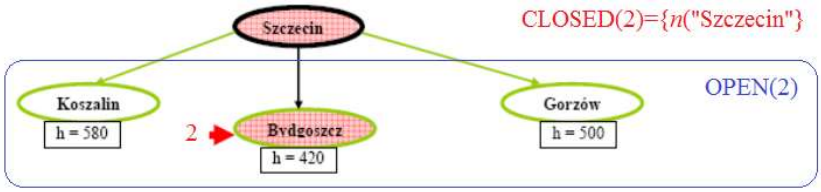
Dla strategii zachłannej nie mają one żadnego znaczenia. Kieruje się ona jedynie oszacowaniem kosztów resztkowych podanych Tabela 3 (ilustrowanych czerwonymi łukami na Rysunek 15). Kroki przeszukiwania podanej przestrzeni stanów, wykonywane zgodnie z poinformowaną strategią zachłanną ilustrują Rysunek 17 i Rysunek 18.

Zbadajmy cechy poinformowanej strategii „zachłannej":

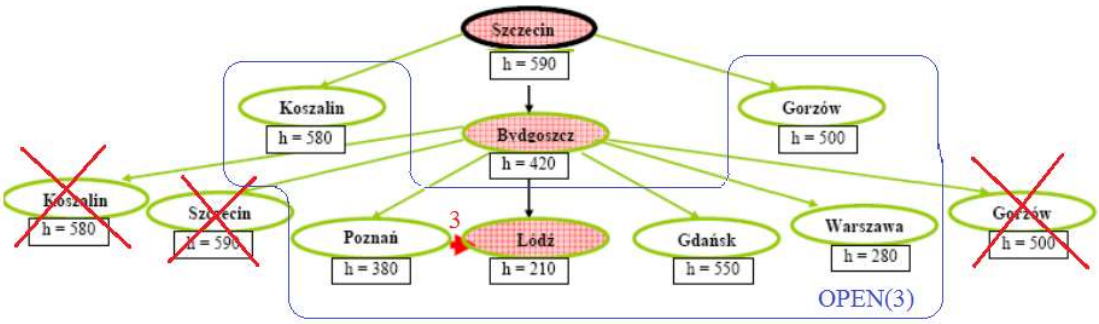
1. zupełność? nie, gdyż przeszukiwanie może utknąć w pętli;
2. czas? $O(b^m)$, ale dobra heurystyka może dać znaczącą poprawę;
3. pamięć? $O(b^m)$, gdyż utrzymuje wszystkie wizytowane węzły w pamięci.
4. optymalność? nie ma takiej gwarancji (np. wybrano drogę przez *Bydgoszcz* o koszcie 735 zamiast drogi optymalnej przez *Gorzów* o koszcie 705).

Zasadniczą **wadą** poinformowanej strategii **zachłannej** jest brak gwarancji uzyskania optymalnego rozwiązania (w sensie minimalizacji rzeczywistego sumarycznego kosztu ścieżki rozwiązania). Tej wady nie posiada następna omawiana strategia poinformowanego przeszukiwania grafu, **A***.

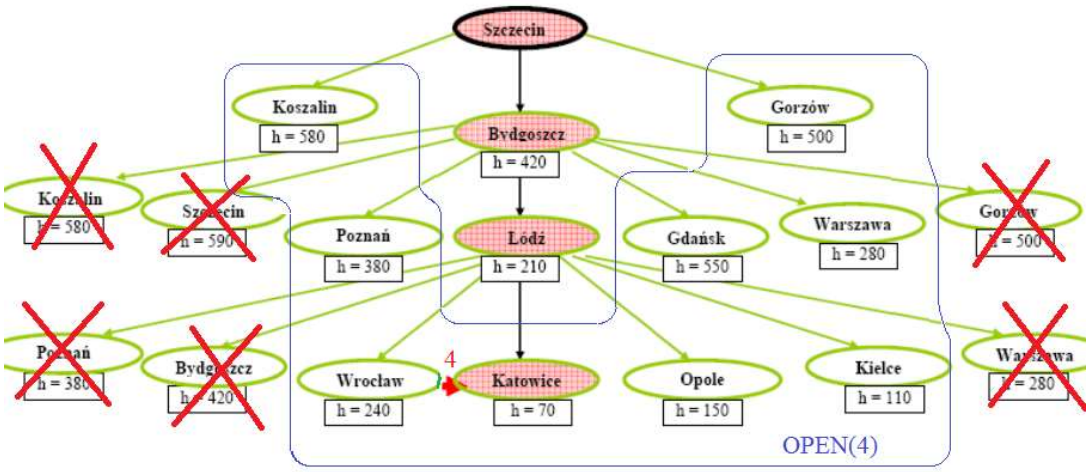
(a)



(b)

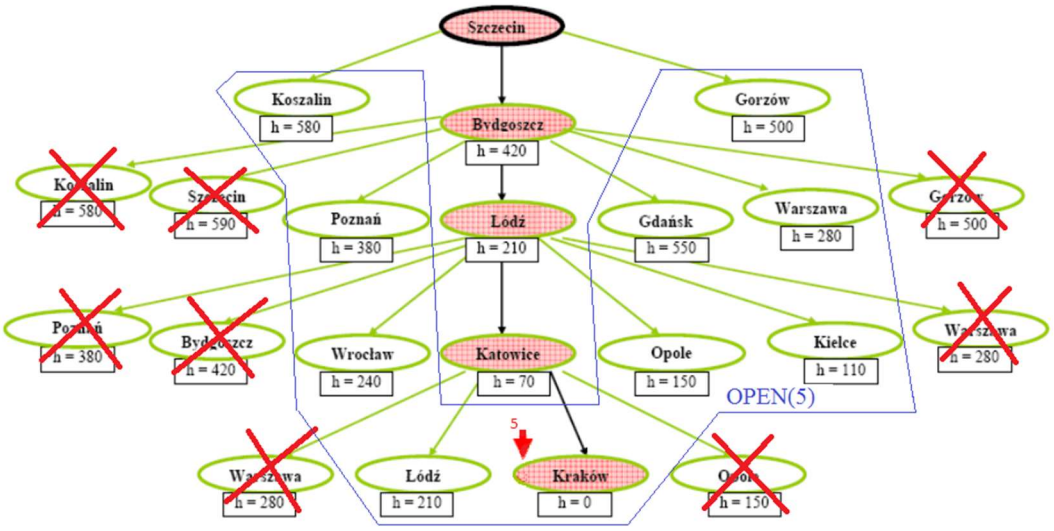


(c)

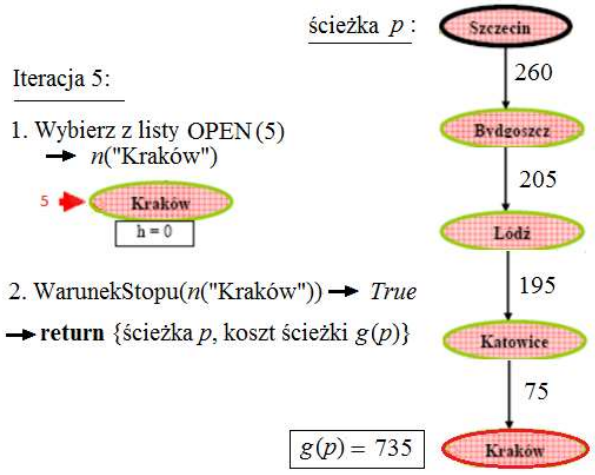


(d)

Rysunek 17. Pierwsze cztery kroki przeszukiwania przestrzeni dla problemu „powrotu do Krakowa” wykonywane przez poinformowaną strategię „zachłanną”: a) wybór węzła „Szczecin” (jedyny i najlepszy węzeł na liście OPEN(1)) i generowanie jego następników, b) wybór węzła „Bydgoszcz, najlepszego z węzłów na liście OPEN(2) – spośród jego następników usuwane są węzły „Koszalin, „Szczecin” i „Gorzów”, gdyż równoważne im węzły już istnieją w drzewie przeszukiwania (decyzyjnym); c) wybór węzła „Łódź”, najlepszego z węzłów na liście OPEN(3); d) wybór węzła „Katowice”, najlepszego z węzłów na liście OPEN(4).



(a)



(b)

Rysunek 18. Ilustracja ostatniego kroku przeszukiwania według poinformowanej strategii zachłannej: (a) wybór węzła „Kraków" jako najlepszego węzła na liście OPEN(5); (b) sprawdzenie warunku zatrzymania („stopu") dla węzła „Kraków" kończy się wynikiem pozytywnym („True") co skutkuje zakończeniem przeszukiwania i wynikiem w postaci ścieżki „p" w drzewie przeszukiwania o rzeczywistym koszcie, g(p)= 735 (liczonym jako suma kosztów akcji przejść pomiędzy miastami podanych w grafie problemu).

3.3 Przeszukiwanie A*

3.3.1 Zasada

Idea strategii A* to: unikać rozwijania ścieżek, które już dotąd są kosztowne a poza tym niezbyt „obiecujące" pod względem możliwości szybkiego dotarcia do celu. Jej realizacja jest możliwa dzięki stosowaniu pełnej funkcji kosztu węzła:

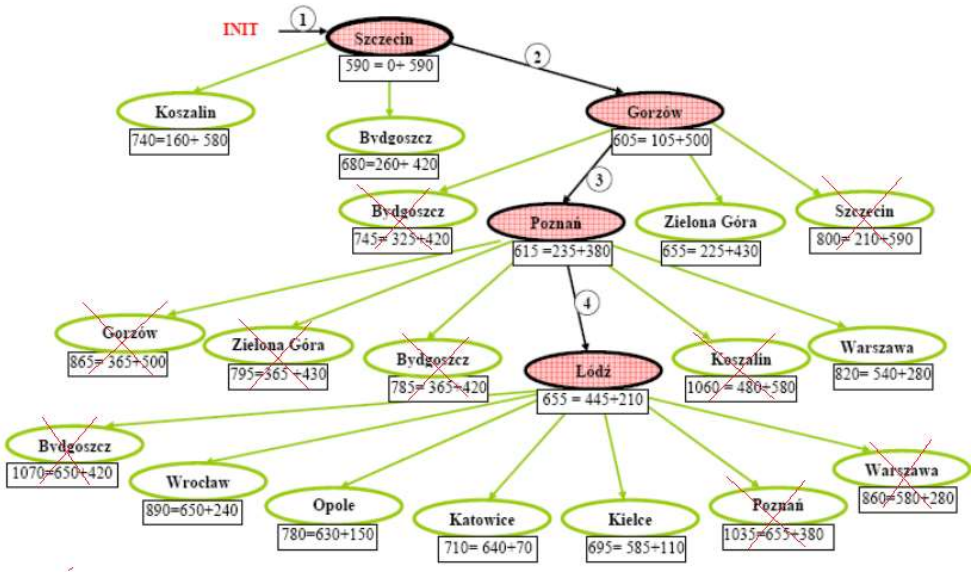
$$f(n) = g(n) + h(n),$$

uwzględniającej zarówno koszt dotarcia do węzła *n* (*składowa g(n)*) jak i przewidywany koszt przejścia z *n* do celu (składowa *h*(*n*)).

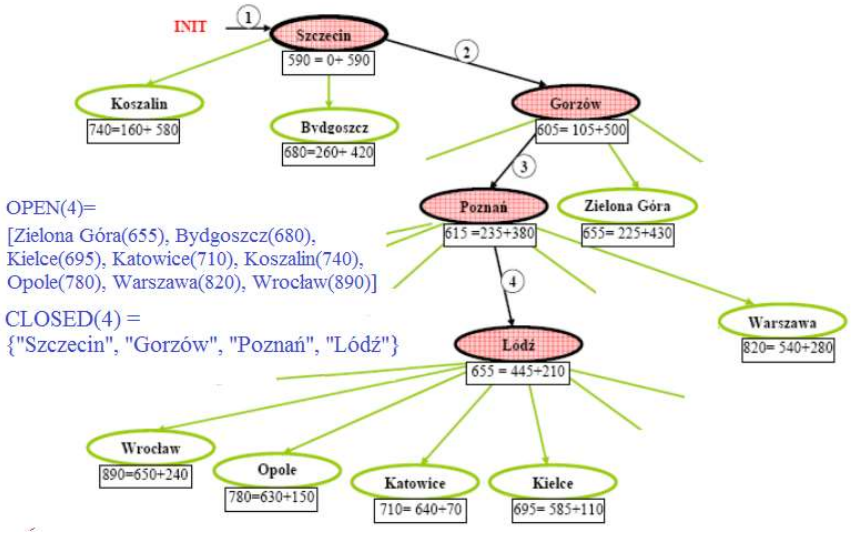
Dzięki temu, że *f*(*n*) reprezentuje przewidywany **całkowity koszt** ścieżki prowadzącej od węzła startowego przez węzeł *n* do celu, a wizytowanie węzłów odbywa się w kolejności rosnącej wartości kosztu, to pod warunkiem **optymistycznego** oszacowania kosztów resztkowych, pierwsze napotkane rozwiązanie będzie jednocześnie najlepszym, czyli optymalnym.

Przykład.

Ilustracja przeszukiwania przestrzeni stanów według strategii A* dla problemu „powrót ze Szczecina do Krakowa" (Rysunek 19, Rysunek 20).

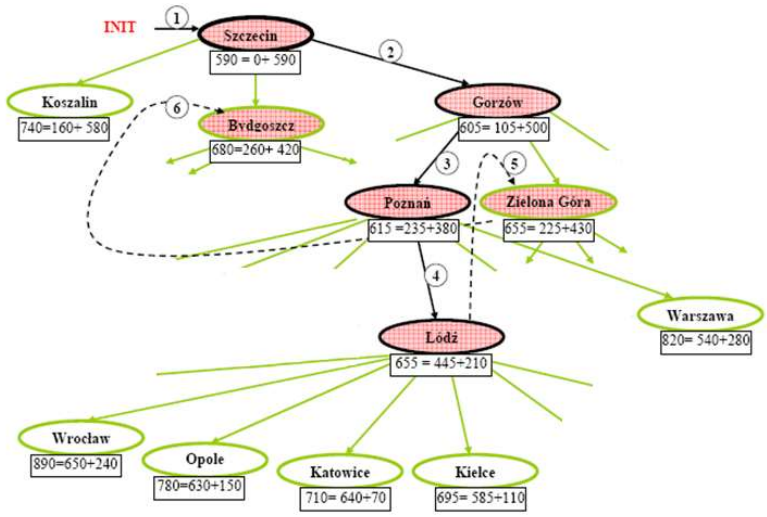


(a)

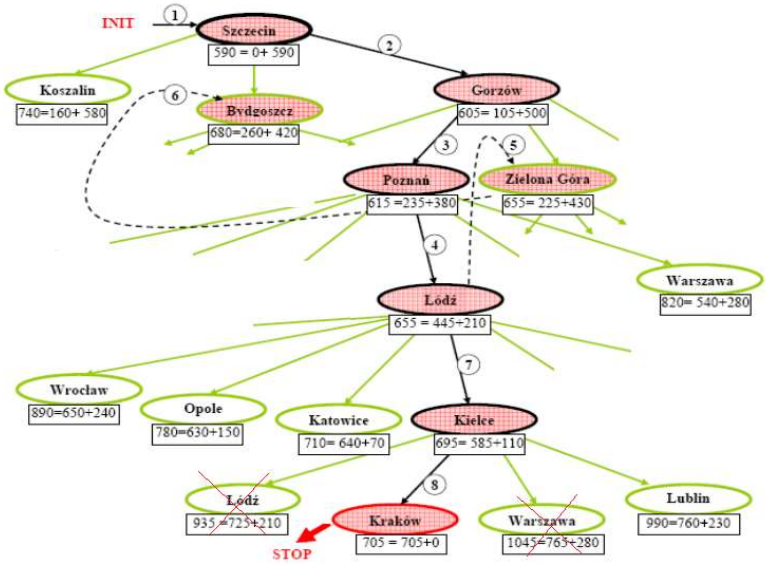


(b)

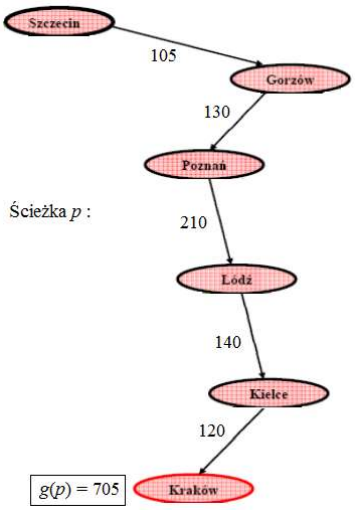
Rysunek 19. Przeszukiwanie przestrzeni zdefiniowanej dla problemu "powrotu do Krakowa" według strategii A: (a) drzewo przeszukiwania po 4 iteracjach wyboru i rozszerzania aktualnie najlepszego węzła (tzn. o najmniejszym koszcie* f(n)*; jeśli generowany jest węzeł równoważny z węzłem, już istniejącym w drzewie przeszukiwania, to jest on dodawany do drzewa (listy OPEN) pod warunkiem posiadania niższego kosztu* g(n) *od węzła równoważnego w drzewie; (b) lista OPEN ma charakter globalny – podobnie jak dla strategii zachłannej.*


INIT
1
Szczecin
590 = 0+ 590
2
Koszalin
740=160+ 580
6
Bydgoszcz
680=260+ 420
Gorzów
605= 105+500
3
5
Poznań
615 =235+380
Zielona Góra
655= 225+430
4
Warszawa
820= 540+280
Łódź
655 = 445+210
Wrocław
890=650+240
Opole
780=630+150
Katowice
710= 640+70
Kielce
695= 585+110

(a)


INIT
1
Szczecin
590 = 0+ 590
2
Koszalin
740=160+ 580
6
Bydgoszcz
680=260+ 420
Gorzów
605= 105+500
3
5
Poznań
615 =235+380
Zielona Góra
655= 225+430
4
Warszawa
820= 540+280
Łódź
655 = 445+210
7
Wrocław
890=650+240
Opole
780=630+150
Katowice
710= 640+70
Kielce
695= 585+110
8
Łódź
935 =725+210
Kraków
705 = 705+0
STOP
Warszawa
1045=765+280
Lublin
990=760+230

(b)


Szczecin
105
Gorzów
130
Poznań
Ścieżka p :
210
Łódź
140
Kielce
120
g(p) = 705
Kraków

(c)

Rysunek 20. Ciąg dalszy przykładu przeszukiwania według strategii A: a) drzewo przeszukiwania po 6 iteracjach; (b) drzewo przeszukiwania po 8 iteracjach – po wyborze węzła „Kraków" i sprawdzeniu warunku stopu cel został osiągnięty; (c) znaleziona ścieżka jest optymalna.*

3.3.2 Implementacja strategii A*

Szczegółowy opis algorytmu implementującego strategię przeszukiwania A* podano w Tabela 5. Dzięki istnieniu zbioru CLOSED, do którego wstawiane są wizytowane węzły, oraz dzięki krokowi 6, uniemożliwiającemu wielokrotne wybieranie równoważnych węzłów (tego samego stanu problemu), algorytm bezpiecznie realizuje przeszukiwanie przestrzeni o topologii grafu.

Tabela 5. Algorytm A poinformowanego przeszukiwania.*

1		INIT: Pobierz węzeł startowy s i umieść go w zbiorze OPEN. Ustaw $f(s)=0$, $g(s)=0$.
2		Pobierz z OPEN węzeł n o najmniejszej wartości funkcji $f(n)$ i umieść go w zbiorze CLOSED.
3		JEŚLI (n jest węzłem końcowym) TO zakończ i zwróć g(n) oraz całą ścieżkę od s do n.
4		Znajdź węzły następców n - niech będą nimi: $n_1' \ldots n_k'$.
5		Dla każdego z następców $n_1' \ldots n_k'$ oblicz koszt dojścia do niego: $g_i' = g(n) + c(n, ni')$.
6		Dla każdego z węzłów $n_1' \ldots n_k'$:
	a	JEŚLI (n_i' nie należy do zbioru OPEN ani do CLOSED) TO dodaj go do zbioru OPEN i ustaw: $g(n_i') = g_i'$, $f(n_i') = g_i' + h(n_i')$.
	b	JEŚLI (n_i' należy do zbioru OPEN lub CLOSED i $g(n_i') > g_i'$) TO ustaw $g(n_i') = g_i'$, $f(n_i') = g_i' + h(n_i')$, usuń ścieżkę od s do n_i'. Jeśli n_i' był w zbiorze CLOSED, to umieść go w zbiorze OPEN.
7		Powtórz od kroku 2.

Strategia A* posiada bardzo pożyteczne własności:

- Zupełność? Tak, dla skończonej przestrzeni (nie istnieje nieskończenie wiele węzłów n, dla których, $f(n) \leq f(G)$, gdzie G jest optymalnym celem).
- Czas? Potencjalnie wykładniczy ale znaczne zmniejszenie czasu przeszukiwania jest możliwe przy istnieniu dobrej heurystyki (bliskiej rzeczywistym kosztom).
- Pamięć? Wszystkie rozwijane węzły są pamiętane, z uwagi na możliwość wystąpienia cykli w grafie.
- Optymalność? Tak, jeśli heurystyka jest **dopuszczalna** to A* zawsze znajduje najlepsze rozwiązanie. Dalej wyjaśnimy, co oznacza dopuszczalna heurystyka - w skrócie mówiąc oznacza to, że oszacowanie musi być optymistyczne.

Przeszukiwanie A* dysponuje jeszcze jedną ciekawą cechą, jest nią **optymalna efektywność**: przy istnieniu *spójnej* heurystyki dla problemu, żadna inna strategia poinformowanego przeszukiwania nie rozwija mniej węzłów niż algorytm A* dla dotarcia do celu.

3.3.3 Dopuszczalna heurystyka

Heurystyka $h(n)$ jest dopuszczalna (ang. *admissible*), jeżeli dla każdego węzła n zachodzi

$$h(n) \leq h^*(n),$$

gdzie $h^*(n)$ jest prawdziwym kosztem osiągnięcia celu z węzła n. Dopuszczalna heurystyka nigdy nie przecenia kosztu osiągnięcia celu, jest więc optymistycznym oszacowaniem rzeczywistego kosztu. Dla przykładu, heurystyka $h(n)$ podana w Tabela 3 dla grupy problemów *„powrót do Krakowa”* nigdy nie przecenia faktycznej odległości, liczonej jako suma odcinków drogi.

Można pokazać, że jeżeli $h(n)$ jest *dopuszczalna* heurystyką, to strategia przeszukiwania A* jest optymalną strategią przeszukiwania grafu. Dowód wynika z kilku obserwacji:

1. Strategia A* rozwija węzły w kolejności nie malejących wartości funkcji oceny (kosztu) f : $f_i \leq f_{i+1}$
2. Strategia A* nie może wybrać węzła o określonym koszcie f_i zanim nie wybierze uprzednio wszystkich węzłów o niższym koszcie.
3. Jeśli heurystyka jest dopuszczalna to koszt każdego częściowego rozwiązania (ścieżki) nie przekracza rzeczywistego kosztu rozwiązania zawierającego tę ścieżkę.
4. Stąd pierwsze wybrane rozwiązanie końcowe nie może być gorsze od żadnego innego rozwiązania. Gwarantuje to osiągnięcie optymalnego rozwiązania.

3.3.4 Spójna heurystyka

Heurystyka jest spójna (ang. *consistent*), jeżeli jest dopuszczalna i dla każdego węzła n i dla każdego następnika n', wygenerowanego przez akcję a, spełniony jest „warunek trójkąta”:

$$h(n) \leq c(n,a,n') + h(n'),$$

gdzie $c(n,a,n')$ oznacza koszt przejścia z n do n' za pomocą akcji a. O koszcie każdej akcji $c(n,a,n')$ z góry zakładamy, że jest nieujemny.
Spójność h oznacza, że:

1) $h(n) \leq h^*(n)$, składowa heurystyczna funkcji kosztu jest „dopuszczalna”;
2) $f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n') \geq g(n) + h(n) = f(n)$

Stąd, $f(n') \geq f(n)$, czyli funkcja kosztu $f(n)$ jest niemalejąca (monotoniczna) wzdłuż dowolnej ścieżki.
Można pokazać, że jeżeli heurystyka $h(n)$ jest spójna, to strategia A* jest efektywnościowo optymalną strategią poinformowanego przeszukiwania grafu. Oznacza to, że żadna inna strategia optymalna, aby osiągnąć cel nie rozwija mniej węzłów niż strategia A*.

3.4 Wyznaczanie składowej heurystycznej

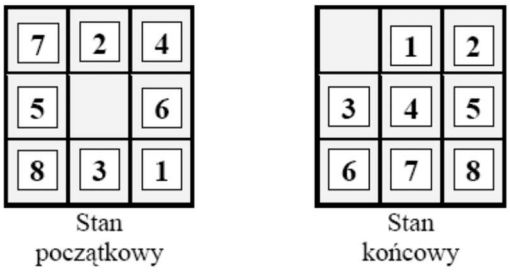
3.4.1 Dominująca heurystyka

Przykład.
Dane są wybrane składowe heurystyczne dla rozwiązania problemu „8-puzzli”:

- $h_1(n)$ - liczba kafelków nie na swoim (docelowym) miejscu,
- $h_2(n)$ - całkowita odległość kafelków od swoich (docelowych) miejsc wyrażona w metryce Manhattan.

Dla stanu początkowego na Rysunek 21. Przykład problemu “8-puzzli” wartości obu heurystyk wynoszą:

- h_1(stan początkowy) = 8;
- h_2(stan początkowy) = 3+1+2+2+2+3+3+2 = 18



Rysunek 21. Przykład problemu “8-puzzli”

Heurystyki możemy ze sobą porównywać. Wprowadźmy określenie **dominującej** heurystyki:

„Jeżeli $h_2(n) \geq h_1(n)$ dla każdego węzła n (i obie heurystyki są dopuszczalne) to h_2 dominuje nad h_1.”

Wtedy h_2 jest bliższa rzeczywistym kosztom ale pozostaje optymistycznym oszacowaniem i dlatego też jest lepszą heurystyką od h_1 dla poinformowanej strategii przeszukiwania.

3.4.2 Generowanie heurystyk metodą „złagodzonego problemu”

Zastanówmy się w jaki sposób „zautomatyzować” proces generacji dobrych heurystyk. Dla bardzo złożonych problemów człowiek jest w stanie podać analityczną postać (wzór matematyczny) jedynie dla obliczenia stosunkowo słabych heurystyk, np. dla heurystyki h_1 w problemie „N-puzzli”. Istotne jest też, aby mogła to zrobić maszyna dysponująca ograniczonym zestawem metod obliczeniowych. Stąd bierze się idea, aby wyznaczenie dobrej heurystyki potraktować jako oddzielny problem przeszukiwania, ale duży prostszy niż oryginalny problem.

Problem „uproszczony”, o zmniejszonych w porównaniu z oryginalnym problemem wymaganiach nakładanych na przejścia pomiędzy stanami, nazywamy problemem **złagodzonym**. Znalezienie optymalnego rozwiązania złagodzonego problemu (ścieżki rozwiązania w tym złagodzonym

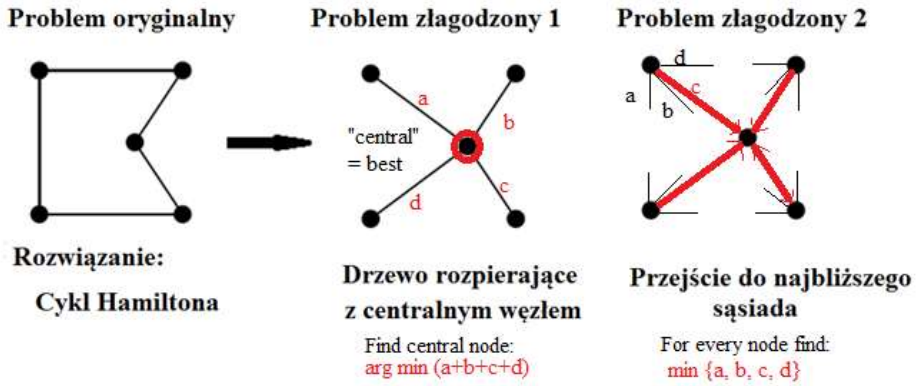
problemie rozpoczynanej w aktualnym stanie *n*) stanowi jednocześnie dobrą, dopuszczalną heurystykę dla tego stanu *n* w oryginalnym problemie.

Np. jeśli złagodzimy problem „8-puzzli" tak, że kafelek może zostać przesunięty w dowolne miejsce, to heurystyka $h_1(n)$ dla oryginalnego problemu powstaje z rozwiązania problemu złagodzonego i wynosi tyle co koszt optymalnego rozwiązania rozpoczynanego w stanie *n*.

Podobnie, jeśli złagodzimy problem „8-puzzle" tak, że kafelek może zostać przesunięty w dowolną sąsiednią pozycję (tzn. nawet zajętą), to heurystyka $h_2(n)$ dla oryginalnego problemu odpowiada kosztowi optymalnej ścieżki w tym problemie złagodzonym.

Przykład.

Generowanie heurystyki metodą „złagodzonego problemu" dla „problemu komiwojażera" (Rysunek 22). W oryginalnym problemie należy znaleźć najkrótszą trasę odwiedzenia *N* miast, odwiedzając każde miasto jedynie raz (jest to problemem o wysokiej złożoności obliczeniowej O(*N*!)). Problem złagodzony wobec oryginalnego problemu – wyznaczyć drzewo rozpierające dla (pod-) zbioru węzłów pozostałych do odwiedzenia (jest to problem o złożoności jedynie $O(N^2)$). Inny problem złagodzony może polegać na znalezieniu długości ścieżki wiodącej od aktualnego węzła przez węzły pozostałe jeszcze do odwiedzenia, których kolejność wynika z zasady wyboru najbliższego następcy. Koszt znalezionego rozwiązania problemu złagodzonego o węźle początkowym *n*, stanowi wtedy optymistyczne oszacowanie kosztu resztkowego dla węzła *n* w problemie oryginalnym, gdy pozostaje jeszcze wizytowanie zadanego (pod-)zbioru miast (węzłów w grafie).



Rysunek 22. Ilustracja problemu oryginalnego i „problemu złagodzonego" dla „problemu komiwojażera": (a) oryginalny problem sprowadza się do znalezienia „cyklu Hamiltona" dla zadanego grafu problemu, o łukach etykietowanych kosztami przejść; (b) problem złagodzony 1 – dla zadanego węzła „centralnego" znaleźć drzewo rozpierające w grafie; (c) problem złagodzony 2 – lokalny wybór najlepszego następnika.

4 Algorytmy „A*-podobne"

Istnieje szereg strategii poinformowanego przeszukiwania, które są podobne do strategii A*, ale posiadają pewne dodatkowe pożądane właściwości. W szczególności omówimy tu strategie IDA*, SMA* i RTA*.

4.1 IDA* („Iterative deepening A*)

Algorytm A* wymaga list OPEN i CLOSE dla pamiętania wszystkich wygenerowanych węzłów. Może to prowadzić do dużych wymagań jego implementacji programowej wobec pamięci. Z pomocą dla zaprojektowania rozwiązania o wymaganiu mniejszej pamięci przychodzi nam zasada podobna do iteracyjnego przeszukiwania w głąb (IDS).

4.1.1 Idea algorytmu IDA*

Zakładamy, że funkcja kosztu f jest monotoniczna (tzn. heurystyka jest „dopuszczalna" i „spójna"):

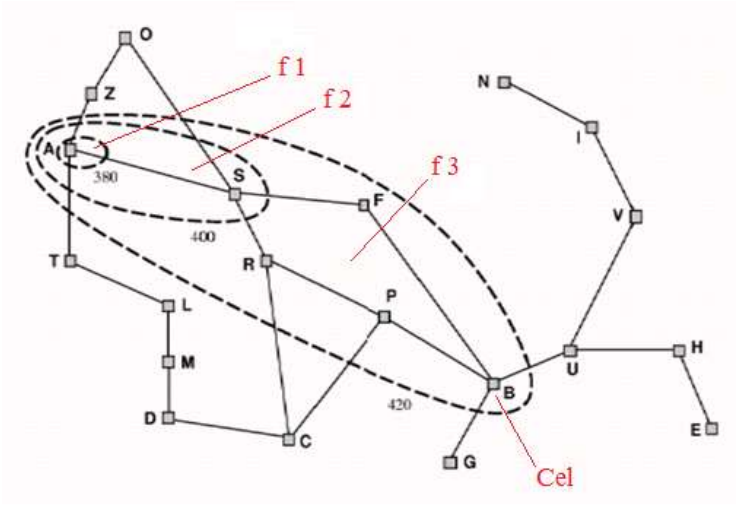
- $f(n) = g(n) + h(n) \leq f^*(n)$ (rzeczywisty koszt),
- $f(n) \leq f(n')$, gdzie n' jest następnikiem n

Wykonujemy iteracje (poinformowanej wersji) przeszukiwania „w głąb" ograniczone wartością maksymalnego progu f_B dla wartości oceny ścieżki $f(n)$. Tzn. powstaje podprzestrzeń ograniczona aktualnie przyjętym progiem dla funkcji oceny $f(n)$, która przeszukujemy zgodnie ze strategią „w głąb" („depth-first"). Jeśli w danej iteracji nie zostaje znalezione rozwiązanie to w kolejnej iteracji próg jest odpowiednio zwiększany do najmniejszej wartości $f(n)$ w poprzedniej podprzestrzeni przewyższającej jej próg.

Własności IDA* i porównanie z A*.

- Jeśli funkcja kosztu jest monotoniczna to IDA* znajduje optymalne rozwiązanie.
- IDA* wymaga znacznie mniej pamięci (liniowa złożoność pamięciowa w funkcji długości ścieżki) niż A*.
- Może wymagać więcej obliczeń – jeśli koszt jest znacząco niedoszacowany i wymaganych jest wiele kroków dla osiągnięcia $f = f^*$.
- W najgorszym przypadku dla IDA*: N obliczeń dla A* ale N^2 dla IDA*

Monotoniczność funkcji kosztu wzdłuż każdej ścieżki umożliwia wyróżnienie kolejnych podprzestrzeni o coraz większych wartościach progu fi (Rysunek 23). Kolejne wartości f_B (f1, f2, f3, …) odpowiadają coraz lepszemu „poinformowaniu" agenta i „zbliżaniu" się podprzestrzeni do optymalnego celu. Przy „dobrej" heurystyce (tzn. gdy oszacowania są bliskie rzeczywistym kosztom resztkowym) podprzestrzenie „rozciągają się" w kierunku celu, unikając nadmiernego przeszukiwania podprzestrzeni położonych dalej od ścieżki rozwiązania.



Rysunek 23. Ilustracja podprzestrzeni przeszukiwanych w kolejnych iteracjach algorytmu IDA.*

4.1.2 Algorytm IDA*

W Tabela 6 podano strukturę algorytmu IDA*.

Tabela 6. Algorytm IDA („Iterative-Deepening-A*”)*

1) (Init) Ustaw f_B = oszacowanie wartości *f* dla węzła początkowego; 2) Wykonaj przeszukiwanie „w głąb” z węzła początkowego, przycinając gałęzie drzewa za liśćmi wtedy, gdy koszt wygenerowanego liścia: $(g + h) > f_B$. 3) Jeśli znaleziono rozwiązanie to zakończ → **return**(optymalne rozwiązanie). 4) Zwiększ f_B do najmniejszej spośród wartości kosztu węzłów liści wygenerowanych (ale nie rozszerzonych) w aktualnej iteracji i kontynuuj od kroku (2).

Efektywność algorytmu IDA*

- Złożoność czasowa: IDA* asymptotycznie zdąża do złożoności A*.
- Złożoność pamięciowa IDA* wynosi tylko O(d) , a A* tymczasem - $O(b^d)$.
- W IDA* unika się sortowania kolejki węzłów.
- IDA* posiada prostszą implementację – brak listy CLOSED (mniejsza lista OPEN).
- IDA* może wykonywać się szybciej mimo, iż generuje więcej węzłów niż A*.
- IDA* może rozwiązać problem, którego A* z powodu braku pamięci nie rozwiąże (lub rozwiąże po znacznie dłuższym czasie).

4.2 SMA* („Simplified memory bounded A*”)

4.2.1 Idea strategii SMA*

SMA* zakłada istnienie jawnego ograniczenia na liczbę węzłów, które mogą być pamiętane podczas przeszukiwania A* i na długość ścieżki. Występują tu zasadniczo dwa parametry:

1. Ograniczenie 1 określa maksymalną liczbę pamiętanych węzłów drzewa decyzyjnego.
2. Ograniczenie 2 określa maksymalną długość ścieżki stanowiącej rozwiązanie.

Sensowne jest założenie, że ograniczenie 1 jest nie mniejsze od ograniczenia 2.
Główne elementy strategii SMA* ():

- Rozwijany jest zawsze najgłębszy liść o aktualnie najmniejszej wartości kosztu *f*.
- Po rozwinięciu węzła (określeniu następników) modyfikuje się koszt *f* tego węzła – przyjmuje on wartość najmniejszej wartości kosztu spośród jego następników.
- Gdy wyczerpie się pamięć, należy obciąć płytsze węzły o najwyższej wartości *f*, ale zapamiętać koszt najlepszego odrzuconego następnika w węźle-rodzicu.
- Ścieżka dłuższa niż „limit długości ścieżki" nie prowadząca do celu uzyskuje koszt ∞ (nieskończenie duży).

4.2.2 Algorytm

Przykładowy pseudokod funkcji implementującej strategię SMA* podano w Tabela 7.

Tabela 7. Pseudokod implementacji strategii SMA.*

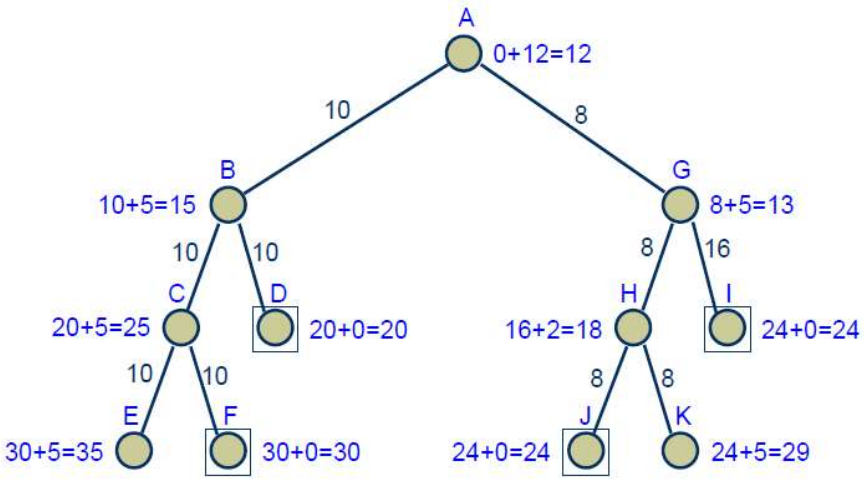
```
function SMAstar(problem, Lm, Lp) returns [ ścieżka, koszt ścieżki ]
{ static: Queue; // lista węzłów uporządkowana kosztem f
   Queue ← MakeQueue(MakeNode(InitialState[problem])); // początkowa lista
   do {
      if Queue jest puste then return [∅, ∞]
      n ← „najgłębszy węzeł o najmniejszym koszcie w Queue"
      if GoalTest(n) then return [ścieżka do n, jej koszt];
      if n jest na maksymalnej głębokości Lp  then   f(s) ← ∞  // nie może zostać rozwinięty
      else {
            s ← NextSuccessor(n); // najlepszy następnik n
            f(s) ← g(n) + c(n->s) + h(s)); // koszt węzła s
            ewentualnie wstaw s do Queue; // jeśli nowy lub „lepszy" węzeł
            f(n) ← Max (f(n), f(s)); // ewentualnie modyfikuj f(n)
            if wszystkie następniki (n) są w pamięci then usuń n z Queue ;
            if pamięć jest pełna (liczba węzłów > Lm) then {
                  usuń węzeł m : najpłytszy z węzłów o najwyższym koszcie f z Queue  i  z pamięci;
                  wstaw jego przodka p do Queue, jeśli konieczne;
                  zapamiętaj koszt usuniętego węzła w węźle przodka: f_mem(p) = f(m).
                  }
     } // koniec pętli do
}
```

Własności strategii SMA*:

- Jest zupełna, jeśli w pamięci zmieści się ścieżka do najpłytszego węzła końcowego.
- Jest optymalna, jeśli w pamięci zmieści się optymalna ścieżka.
- Jeśli w pamięci mieści się całe drzewo, SMA* zachowuje się identycznie jak A*.

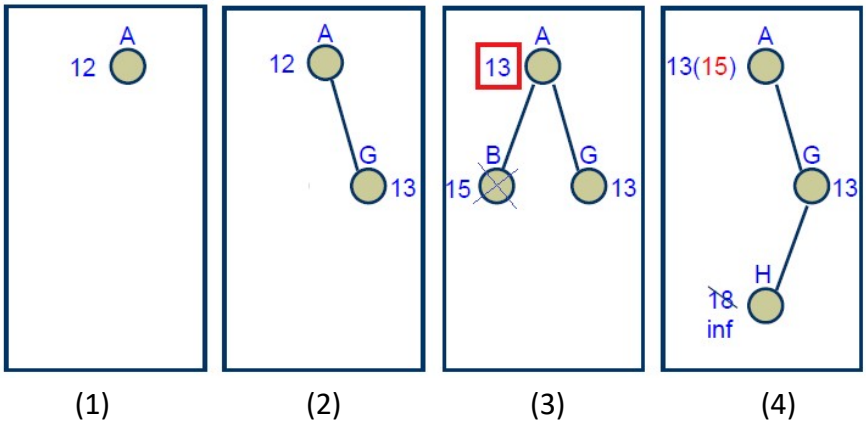
4.2.3 Przykład

Załóżmy (jeszcze nieznaną algorytmowi) pełną przestrzeń stanów (o postaci drzewa) taką, jak podana na Rysunek 24.



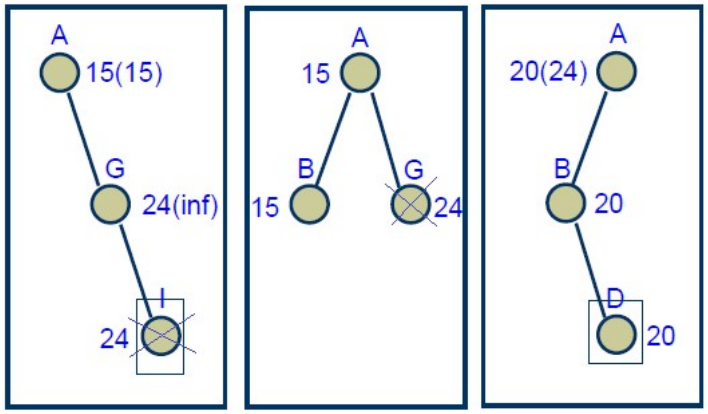
Rysunek 24. Przykład przestrzeni stanów o postaci drzewa.

Wprowadzamy ograniczenia: „limit pamięci = 3” i „limit długości ścieżki (liczby przejść) = 2”. Pierwsze cztery kroki algorytmu SMA* dla problemu z Rysunek 24 przedstawia Rysunek 25.



*Rysunek 25. Cztery początkowe kroki przeszukiwania przestrzeni algorytmem SMA**

W porównaniu z A* nowe są kroki określone jako 2, 3 i 4. W kroku 2 następuje dodanie węzłów G i B (jeden po drugim) oraz modyfikacja kosztu *f*(A) najniższą wartością *f* jego następnika, czyli 13. W kroku 3 wybrany zostaje węzeł G, nie jest węzłem końcowym a limit pamięci (3) uniemożliwia jego rozwinięcie. Dlatego z pamięci usuwany jest („gorszy”) węzeł B. Jego koszt (15) zostaje zapamiętany w węźle rodzica, czyli A. Efekt tych operacji widoczny jest w kroku 4. Teraz jest miejsce na dodanie najlepszego następnika węzła G – jest nim węzeł H o koszcie 18. Zostaje on wybrany. Węzeł H jest nie-końcowy ale nie może zostać rozwinięty, gdyż osiągnięto limit długości ścieżki. W takiej sytuacji węzeł H uzyskuje koszt „nieskończony” i jest usuwany po to, aby zrobić miejsce w pamięci dla kolejnego następnika węzła B.

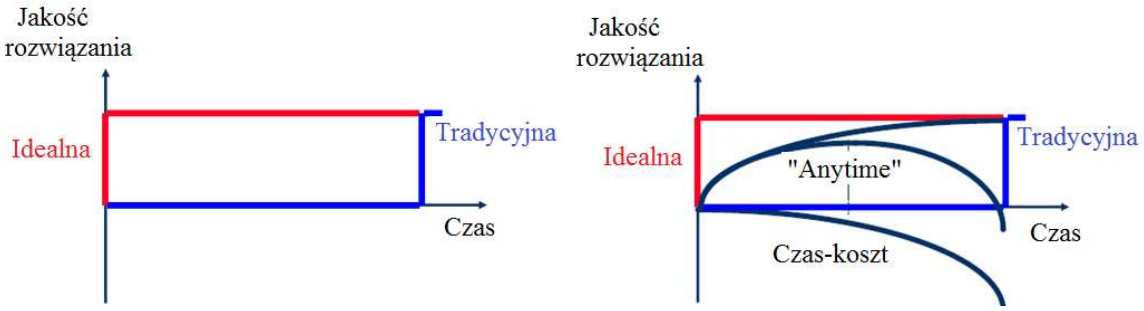


Rysunek 26. Kolejne kroki 5-7 podczas przeszukiwania algorytmem SMA.*

Kolejne kroki (5) – (7) ilustruje Rysunek 26. Po usunięciu węzła H jego rodzic pamięta jego koszt (nieskończoność) oraz przyjmuje koszt aktualnie najlepszego następnika, czyli 24. Węzeł I jest wprawdzie węzłem końcowym ale nie zostanie wybrany, gdyż rodzic węzła G, czyli A pamięta o innej ścieżce, której koszt wynosi aktualnie 15. Wybór pomiędzy węzłem I o koszcie 24 a inną ścieżką o koszcie 15, oczywiście kończy się ta druga ewentualnością. Co więcej węzeł I zostaje usunięty, gdyż osiągnięto limit pamięci. W kroku (6) ponownie dodany został węzeł B, a węzeł G w kolejnym kroku zostaje usunięty, aby móc rozwijać lepszy węzeł B. Koszt usuniętego G pamiętany w węźle jego rodzica A. Efekt ten widoczny jest w kroku (7), w którym też dodany zostaje najlepszy następnik węzła B. W następnym kroku ten węzeł D sam zostaje wybrany jako aktualnie najlepszy i okazuje się, że jest to węzeł końcowy.

4.3 „Anytime A*" (przeszukiwanie z wynikiem „o dowolnym czasie")

Strategia przeszukiwania byłaby idealna wtedy, gdy rozwiązanie optymalne uzyskane zostaje natychmiast po rozpoczęciu przeszukiwania. Jednak zazwyczaj maksymalizuje się jakość rozwiązania bez uwzględnienia nakładu czasu na jego znalezienie. Przeciwną filozofią kierują się strategie o „wyniku w dowolnym czasie" (typu „anytime"). Uwzględniają one oba kryteria - jakości i czasu uzyskania rozwiązania. Ich zaletą jest to, że dysponują rozwiązaniem niemal zaraz po rozpoczęciu. Rozwiązanie początkowe nie musi być optymalne (i zwykle nie jest). Jednak w dalszym czasie działania algorytmu rozwiązanie to jest poprawiane (Rysunek 27).



*Rysunek 27. Idea przeszukiwania typu "anytime A**

Istotne różnice pomiędzy „anytime A*" a A*:

1. „Anytime A*” stosuje **niedopuszczalną** heurystykę, co umożliwia szybkie znalezienie sub-optymalnych rozwiązań.
2. Po znalezieniu pierwszego i kolejnych rozwiązań, Anytime A* kontynuuje przeszukiwanie po uprzednim zredukowaniu listy OPEN o ścieżki gorsze od aktualnego rozwiązania.
3. Jeśli lista OPEN staje się pusta znalezione rozwiązanie staje się optymalne.

Przykład niedopuszczalnej funkcji kosztu i heurystyki:

$$f'(n) = (1-w)\cdot g(n) + w\cdot h(n)$$

Powyższa funkcja jest tylko wtedy dopuszczalną funkcją kosztu, gdy $h(n)$ jest dopuszczalne i $w \leq 0.5$.

4.4 Real-time A* (RTA*)

A* najpierw znajduje pełną ścieżkę od węzła startowego do końcowego w trybie off-line, a dopiero potem jest ona wykonywana (jako sekwencja akcji) przez agenta. W trakcie przeszukiwania algorytm A* operuje na globalnym horyzoncie, co uniemożliwia jego wykonywanie przez agenta w trybie on-line (czyli w rzeczywistym czasie), bo agent nie może fizycznie przenosić się w jednym kroku z jednego końca przestrzeni do drugiego. Algorytm czasu rzeczywistego (np. RTA*) pozwala na natychmiastowe wykonywanie akcji pomimo, że brak jest jeszcze pełnego rozwiązania. Jest to możliwe dzięki operowaniu na lokalnym horyzoncie aktualnego stanu, jednak w odróżnieniu od algorytmów lokalnego przeszukiwania typu algorytmu „wspinaczkowego”, RTA* jednocześnie gwarantuje osiągnięcie optymalnego rozwiązania.

4.4.1 Idea strategii RTA*

- Może być wykonywana on-line, gdyż następniki stanu wybierane są zawsze z jego lokalnego otoczenia;
- Pod pewnymi warunkami gwarantuje osiągnięcie optymalnego stanu końcowego w skończonej liczbie kroków, ale ilość wykonanych akcji (rozwiniętych węzłów) zwykle nie będzie optymalna.

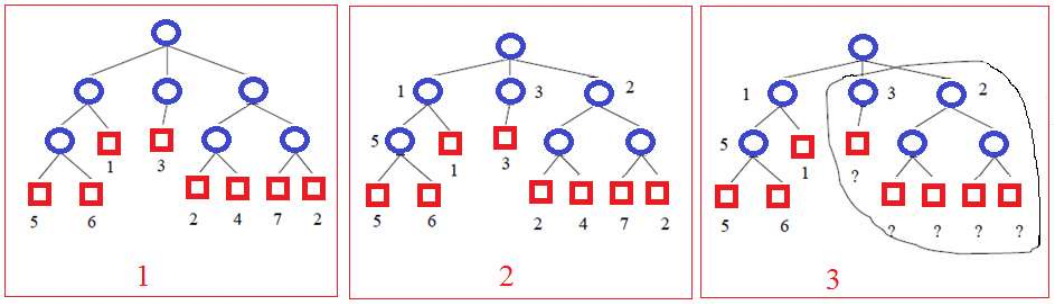
RTA* to strategia A* -podobna, ale o ograniczonym (lokalnym) horyzoncie węzłów z możliwością wykonania **nawrotu** (powrotu do węzła przodka). Wymagane jest, aby heurystyka była dopuszczalna i funkcja kosztu była monotoniczna (czyli aby heurystyka była spójna). RTA* zawiera następujące modyfikacje A*:

1. Podejmij decyzję o pojedynczej akcji:
   - Wykonaj podgląd w przód dla bezpośrednich następników aktualnego stanu (przeszukiwanie drzewa „*mini-min*” z cięciem „*alfa*”);
   - Wybierz najlepszy węzeł spośród następników lub wykonaj nawrót (do węzła-poprzednika) - ocena wszystkich kandydatów ustalana jest względem aktualnego węzła.
2. Pamięć związana z sekwencją decyzji:

- wszystkie uprzednio wizytowane węzły są pamiętane w wykazie – umożliwia to wykonanie nawrotów i zapobiega ponownemu wykonywaniu „podglądu wprzód” dla znanego węzła;
- dzięki modyfikacji kosztu wizytowanego węzła unika się zapętlenia procesu przeszukiwania (po wybraniu najlepszego następnika, w wykazie dla wizytowanego węzła pamiętany jest koszt jego „drugiego najlepszego” węzła-następnika).

4.4.2 Podgląd w przód (przeszukiwanie drzewa „mini-min” z cięciem „alfa”)

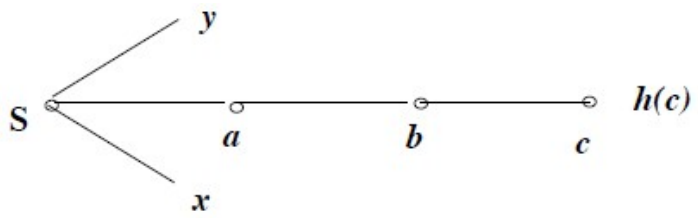
Funkcja podglądu w przód działa następująco (Rysunek 28):
1: Przeszukuje drzewa następników aktualnego stanu dla (niewizytowanych jeszcze) następników – bada je od korzenia (dany następnik) do pewnej zadanej głębokości w celu określenia szacowanego najmniejszego możliwego kosztu *f* ścieżki rozwiązania dla danego następnika.
2: Dla uniknięcia systematycznego przeglądania drzewa stosuje „cięcie alfa” – pamięta najmniejszą wartość „*alfa*” kosztu *f* dla już wizytowanych liści w drzewie danego następnika i przycina gałęzie, gdy zaczynają przekraczać lub być równe wartości *alfa*. W takiej sytuacji, z uwagi na monotoniczność funkcji kosztu wzdłuż ścieżki drzewa, nie ma już potrzeby kontynuowania takiej ścieżki.



Rysunek 28. Idea podglądu w przód (dla aktualnego stanu) w celu lepszego poinformowania o kosztach ścieżki dla jego następników. Podczas podglądania wprzód następników stanu metodą “w głąb” (tu założono lewostronne obejście drzewa) uwzględnia się przycinanie poddrzewa (gałęzi) wtedy, gdy ich korzeń ma większy koszt (lub równy) niż inne już wizytowane poddrzewo (gałąź).

4.4.3 Krok wyboru następnika (ruch „do przodu”)

Po wykonaniu „podglądu w przód” dla aktualnego węzła strategia RTA* jest lepiej poinformowany niż dotychczas o możliwych kosztach ścieżek rozwiązań wiodących przez wszystkich następników aktualnego węzła i może wyznaczyć najlepszego następnika dla ruchu „do przodu”. Załóżmy, że agent jest w stanie reprezentowanym węzłem S (Rysunek 29).



Rysunek 29. Ilustracja stosowania „podglądu w przód" dla węzła S.

Po „podejrzeniu ścieżki w przód" (od węzła „*a*" do węzła „*c*"), węzeł „*a*" uzyskuje nowe oszacowanie kosztów resztkowych h'(a) – koszt ten nie jest mniejszy od dotychczasowego (h'(a)≥h(a)), ale nadal pozostaje dopuszczalny:

$$h(a') \le g(a \rightarrow c) + h(c) \le h^*(a);$$

Podobnie wyznaczane jest nowe oszacowanie kosztów resztkowych dla pozostałych następników *x* i *y*. Teraz możliwa jest lepiej poinformowana decyzja co do tego, jaki krok do przodu wykonać ze stanu S - do *„y", „a*", lub „*x*".

Jednak istotnym nowym elementem RTA* jest istnienie alternatywy dla ruchu „do przodu" w postaci wykonania nawrotu do poprzednika węzła S (jeśli istnieje).

4.4.4 Nawrót i pamięć związana ze ścieżką

W procesie wyboru następnego węzła odróżnia się kroki wykonane „do przodu" i krok nawrotu, czyli powrotu do węzła-rodzica. Podczas przeszukiwania w tej strategii nie ma potrzeby stosowania globalnego skraju drzewa (listy OPEN), ani też pamiętania wszystkich dotychczasowych ścieżek, a jedynie należy pamiętać aktualną ścieżkę (od startu do aktualnego węzła) oraz informację o podglądach wprzód wykonanych dla wcześniej badanych węzłów (dla uniknięcia powtórnego wykonywania podglądu).

Idea nawrotu

Za każdym razem, gdy należy wybrać następnika aktualnego węzła, agent rozpatruje czy „pójść do przodu" czy tez wykonać nawrót do swojego węzła-poprzednika.

Funkcja kosztu dla ruchu do przodu, np. z aktualnego węzła „S" do następnika „*a*" jest postaci,

$$f(a) = c(S \rightarrow a) + h'(a),$$

czyli składa się z kosztu akcji przejścia z „S" do „*a*" i ze zmodyfikowanego kosztu resztkowego dla „a". Ewentualny koszt dotarcia do węzła S, czyli f(S) nie odgrywa roli w podjęciu decyzji, gdyż f(*a*) liczony jest względem aktualnego stanu a nie stanu początkowego.

Jeśli z węzła S wykonany zostanie ruch do przodu to dla węzła S pamiętany jest koszt f(x) jego „drugiego najlepszego" następnika „x". Ma to na celu uniknięcia pętli – utknięcia w stale wykonywanej sekwencji ruchów do przodu i nawrotów. Wizytowany i rozwijany „ruchem do przodu" węzeł S przyjmuje jako swoje nowe oszacowanie kosztów resztkowych $h_{drugi}(S)$, „drugą najlepszą" wartość *f*(*x*) spośród swoich następników.

Koszt akcji nawrotu może być określony w RTA* na dwa alternatywne sposoby – w zasadzie możemy mówić o dwóch wersjach algorytmu RTA*.

W oryginalnym artykule wprowadzającym strategię RTA* (R.E.Korf: Real-time heuristic search) koszt wykonania nawrotu z aktualnego węzła „S” do poprzednika „A” określany jest jako:

$$f(A) = c(S->A) + h_{drugi}(A),$$

Należy wykonać **nawrót** do poprzedniego rzeczywistego stanu wtedy, gdy szacowany koszt osiągnięcia celu (rozwiązania) z tego stanu PLUS koszt powrotu do niego jest mniejszy niż koszt pójścia „do przodu” z aktualnego stanu do jego najlepszego następnika.
Cel takiego RTA* określony jest jako najszybsze znalezienie rozwiązania – niekoniecznie będzie to rozwiązanie optymalne w sensie ścieżki (liczonej w trybie off-line) od stanu startowego do końcowego. Optymalny charakter ma jedynie ścieżka on-line, rzeczywiście wykonywana przez agenta.

Alternatywne rozwiązanie określenia kosztu nawrotu ma postać:

$$f(A) = - c(S->A) + h_{drugi}(A),$$

W tym przypadku należy wykonać **nawrót** do poprzedniego rzeczywistego stanu wtedy, gdy szacowany koszt osiągnięcia celu z tego stanu MINUS koszt powrotu do niego jest mniejszy niż koszt pójścia „do przodu” z aktualnego stanu do jego najlepszego następnika.
Celem takiej wersji RTA* jest gwarancja znalezienia optymalnej ścieżki (tej samej co znaleziona w trybie off-line przez A*) w sensie kosztu ścieżki (liczonej w trybie off-line) od stanu startowego do końcowego.

Istnieje wykaz węzłów - węzły wizytowane podczas aktualnego ruchu „do przodu” są zapisywane w wykazie i ich składowa heurystyczne przyjmują drugą najlepszą wartość. Stany znajdujące się w wykazie nie są ponownie modyfikowane w kroku „podglądu w przód”.

5 Pytania

1. Przedstawić wybrany problem decyzyjny w postaci problemu przeszukiwania przestrzeni stanów.
2. Na czym polegają strategie ślepego przeszukiwania? Wymienić główne strategie i zilustrować je na przykładzie.
3. Przy pomocy jakich kryteriów porównujemy ze sobą strategie przeszukiwania?
4. Omówić problem przeszukiwania grafu.
5. Porównać ze sobą strategie ślepego przeszukiwania.
6. Przedstawić strategie poinformowanego przeszukiwania? Kiedy strategia „najlepszy najpierw” jest poinformowana a kiedy nie?
7. Przedstawić strategie przeszukiwania: zachłanną i A*. Która z nich jest optymalna i w jakich warunkach?
8. Co oznaczają pojęcia: „dominacja heurystyki” i „problem złagodzony”?
9. Omówić strategie A*-podobne: IDA*, SMA*.
10. Na czym polega strategia RTA* - co oznacza jej optymalność w sensie ścieżki on-line?

6 Bibliografia

1. S. Russel, P. Norvig: *Artificial Intelligence. A modern approach.* Prentice Hall, 2002 (2nd ed.), 2013 (3d ed.). [Rozdział 3. Podrozdział 4.5]
2. M. Flasiński: *Wstęp do sztucznej inteligencji*. Wydawnictwo Naukowe, PWN, 2011. [Rozdział 4]