KRZYSZTOF SIUZDAK
IV rok IO
specjalność : bazy danych
nr indeksu : 76855

WROCŁAW 18.01.99r.

Przedmiot : **Wprowadzenie do sztucznej inteligencji (projekt).**

TEMAT :

Zastosowanie metod heurystycznych w grach logicznych

W projekcie chciałbym przedstawić metody jakie zostały wykorzystane przy tworzeniu różnego rodzaju gier logicznych. Będą to między innymi:

1. schemat i drzewo gier - przedstawienie najważniejszych elementów schematu oraz metod przeszukiwania użytecznych przy wyznaczeniu strategii rozwiązania dla gier dwuosobowych z pełną informacją,
2. strategia gry - zajmuje się podejmowaniem decyzji oraz planowaniem rozgrywki na kilka posunięć na przód,
3. algorytmy - przegląd metod wykorzystujących heurystyczne przeszukiwanie w teorii gier logicznych:
    - **minimaksowy** - przestawia twierdzenie o minimaksie oraz konstrukcję planu rozgrywki prowadzącej do punktu „siodłowego”,
    - **strategia „rozwiązania”** - najprostsza metoda wyznaczenia drzewa rozwiązania bez badania wszystkich wierzchołków,
    - **algorytm a-b** (gdzie : **a** - **alfa** , **b** - **beta**) - modyfikacja **algorytmu minimaksowego**, dotyczy jednak gier uogólnionych, procedura znajdująca optymalny element, najlepszy ruch w grze,
    - **algorytm „rozpoznawania”** - początkowo wykorzystywany do teoretycznej analizy drzew gier, stosowany do konstrukcji drzew gier,
    - **algorytm SSS*** - poszukuje optymalnego drzewa rozwiązań, w niejawnym drzewie gry znalezione drzewo reprezentuje najlepszą strategię dla gracza.

KRZYSZTOF SIUZDAK **WROCŁAW 18.01.99r.**
IV rok IO
specjalność : bazy danych
nr indeksu : 76855

Wstęp teoretyczny

Metody heurystyczne należą do jednych z najważniejszych narzędzi sztucznej inteligencji. Są one różnie interpretowane i wykorzystywane na wiele różnych sposobów.

Pojęcie heurystyki :

Słowo heurystyka pochodzi z języka staro greckiego. **Heuriskein** - oznaczało sztukę odkrywania nowych metod rozwiązania lub znajdowania rozwiązań problemów. Zajmował się nią **Pappus z Aleksandrii**, matematyk żyjący na przełomie III i IV wieku. usystematyzował i podsumował on wszystkie prace i osiągnięcia matematyków greckich. Uważał, że to **Euklides**, **Apoloniusz z Pergi** oraz **Aristaeus Starszy** byli twórcami heurystyki. **Pappus**, w swoich wielu pracach, zajmował się przede wszystkim analizą i syntezą. Za podstawę analizy uważał żądany wynik zadania, a sama analizą nazywał proces rozkładania wyniku na elementy, z których da się go osiągnąć. Według **Pappusa**, syntezą był proces odwrotny do analizy, rozpoczynający się on od elementów otrzymanych w wyniku analizy i wyprowadzający wynik. W jego ujęciu, analiza była układaniem planu rozwiązania, natomiast synteza wykonaniem tego planu. Zasługą **Pappusa**, z punktu widzenia współczesnej heurystyki było spostrzeżenie, że przystępując do rozwiązywania zadania (analizy), zakładamy prawdziwość lub spełnienie wyniku. Matematyk ten zajmował się także metodami dowodzenia twierdzeń, dobierania zadań pomocniczych i innymi problemami.

Prace **Pappusa**, nad metodami rozwiązywania problemów, kontynuował filozof i matematyk **Rene Descartes** (1596-1650). W swoich dwóch pracach : „Rozprawa o metodzie.” i niedokończonej : „Reguły kierowania umysłem.” starał się przedstawić zasady dokonywania odkryć, rozwiązywania nowych problemów. Starał się poznać i podać metodę myślenia całkowicie niezawodną, uniwersalną i użyteczną w praktyce. W rozważaniach opierał się głównie na wzorcach rozumowania matematycznego.

Gottfried Leibniz (1646-1716) w „Sztuce dokonywania odkryć.” i wielu innych swoich pracach podkreślał znaczenia wyjaśnienia źródeł odkryć i wyprowadzenia odkryć. Jego ideałem dochodzenia do wiedzy był również matematyczny sposób myślenia. Leibniz stworzył nawet projekt uniwersalnego języka ideograficznego.

Bernard Bolzano (1781-1848) określił w bezpośredni sposób zadania heurystyki. Polegały one na sformułowaniu reguł i metod badania problemów dokonanym na podstawie analizy metod ich rozwiązywania. Zasługą **Bolzano** było także oddzieleni psychologii od logiki i wprowadzenie pojęcia rozumienia zadań wynikającego z reguł logiki matematycznej.

Nowoczesna heurystyka :

Gyorgy Polya w 1945 roku opublikował swoją pracę pt. „Jak to rozwiązać.”, którą uznano za jedną z najważniejszych w literaturze dotyczącej sztucznej inteligencji. Poświęcona jest ona przede wszystkim nauczaniu matematyki, a dotyczy nauczania myślenia. „Krótki słownik heurystyczny” to najważniejsza część tej rozprawy. Zawiera on podstawowe pojęcia heurystyki. **Polya** potraktował heurystykę jako przedmiot badań i zastosowań praktycznych, podkreślając jej podstawy doświadczalne. Według niego, heurystyka jest formalizacją doświadczeń zebranych przy rozwiązywaniu zadań oraz obserwacji rozwiązywania zadań, a także próbą znalezienia wspólnych cech różnych metod rozwiązywania. Takie podejście pedagogiczne sprowadziło zadania heurystyki do lepszego rozumienia procesu myślenia w rozwiązywaniu problemów i nauczaniu. Poszczególne hasła słownika sygnalizowały

problemy prawdopodobieństwa uzyskania rozwiązania, jego użyteczności oraz nieścisłość operacji myślowych a także ich uwarunkowania psychologiczne.

Wielu autorów proponuje pojęcia heurystyki i algorytmu. Algorytm według nich jest rodzajem generatora rozwiązań, spełniających następujące warunki:

- zastosowanie algorytmu wyznacza skończony ciąg stanów;
- ciąg stanów ma stan początkowy (zbiór wejściowy, opis sytuacji);
- każdy stan (poza końcowym) ma jednoznaczny następnik;
- stan końcowy jest rozwiązaniem lub sygnałem, że problem nie ma rozwiązania.

Algorytm jest więc działaniem według ściśle wyznaczonego schematu; od stanu początkowego do stanu końcowego. Natomiast heurystyka może rozwiązywać dany problem, lecz nie doje gwarancji znalezienia rozwiązania. Działa jak matematyk uzasadniający twierdzenie nie wiedząc przy tym czy jest ono prawdziwe i czy jest metoda postępowania , która zakończy się sukcesem. W tym zestawieniu postępowanie heurystyczne jest bardzo twórcze. Wiele kontrowersji wzbudza takie przeciwstawianie pojęć algorytmu i heurystyki.

Metoda heurystyczna jest także uważana za narzędzie planowania sposobów rozwiązywania, podobnie do analizy opisywanej przez **Pappusa**. W tym sensie każda procedura jest heurystyczna, jeżeli tylko jej celem jest odkrycie nieznanego rozwiązania danego problemu. O wyborze metody rozwiązywania ma decydować właśnie heurystyka.

Pojmowanie heurystyki współcześnie jest coraz bardziej odległe od koncepcji **Polya**. Jego metody dotyczyły głównie odkrywania, wymyślania rozwiązań, dopuszczały także „działania po omacku". Rozwiązanie u **Polya**, z metodologicznego punktu widzenia, było wynikiem ogólnych rozważań, które dopasowywano do konkretnego problemu. Przy heurystycznym rozwiązywaniu zadań duże znaczenie ma obecnie stosowanie metod ściśle związanych ze specyfikacją dziedziny, do której należy problem. Kierunkiem dominującym przy wprowadzaniu i odkrywaniu nowych heurystyk jest poprawianie metod i strategii już istniejących. Dąży się także do formalnego zdefiniowania struktur i przestrzeni obiektów rozwiązywanych problemów.

Przeszukiwanie heurystyczne

Pojęcie „przeszukiwania heurystycznego" pojawiło się we wczesnych latach sześćdziesiątych. Dotyczyło ono początkowo zwiększania efektywności procesu rozwiązywania problemów. Heurystyka była narzędziem eliminowania (podczas przeszukiwania) niektórych rozwiązań z dużego zbioru wszystkich możliwych rozwiązań. Następnie, zasady przeszukiwania heurystycznego były w różny sposób formalizowane. Przyjęto w uproszczeniu, że rozwiązany problem można opisać za pomocą obiektów (stanów) i operatorów. Nowe obiekty są generowane przez zastosowane operatory (jeśli jest to możliwe). Są one również reprezentantami reguł generowania obiektów. Grafy obiektów są definiowane przez stany początkowe i operatory. Przeszukiwanie heurystyczne jest procesem poszukiwania żądanego stanu, drogi do żądanego stanu lub podgrafu spełniającego zadane warunki.

Przy wyborze operatorów najbardziej obiecujących, przeszukiwanie heurystyczne posługuje się różnymi środkami (analogie, uproszczenie), których celem jest ograniczeni zbioru przeszukiwanych obiektów (w zadaniach praktycznych zazwyczaj występują duże przestrzenie stanów). Z drugiej strony heurystyka nie daje gwarancji znalezienia rozwiązania, chociaż „dobra" heurystyka powinna wyznaczać najlepsze wyniki osiągalne w zadanym czasie. Różni autorzy różnie określają heurystyczne przeszukiwanie. Heurystyka:

- jest praktyczną strategią poprawiającą efektywność rozwiązywania złożonych problemów;

- prowadzi do rozwiązania wzdłuż najbardziej prawdopodobnej drogi omijając mniej obiecujące ścieżki;
- podaje proste kryterium wyboru kierunków postępowania bez jednoznacznego określania stanów dobrych i złych;
- powinna pozwolić na uniknięcie badania „ślepych uliczek" i wcześniejsze wykorzystywanie zdobytych w trakcie badania informacji.

Funkcje heurystyki w przeszukiwaniu

Z punktu widzenia efektywności przeszukiwania istotne jest uwzględnienie następujących czynników:

- niepewność wyniku,
- niekompletność dostępnej wiedzy,
- poprawienie uzyskanego wyniku.

Wykorzystanie heurystyki w procesie konstruowania rozwiązania zadania zwiększa niepewność otrzymania wyniku. Niepewność wyniku związana jest z wykorzystaniem wiedzy „nieformalnej" (praw, reguł, intuicji), której użyteczność nie jest do końca znana. Z tego względu metody heurystyczne wykorzystywane są tam, gdzie algorytmy wyznaczają niezadowalające rozwiązania lub nie gwarantują rozwiązania zadania. Maja one szczególne znaczenie przy rozwiązywaniu zadań (problemów) o dużej złożoności (gdzie dokładny algorytm zawodzi), a zwłaszcza w zadaniach związanych z rozpoznawaniem mowy, obrazów, w robotyce, konstruowaniu strategii gier. Prosty sposób modelowania dziedziny problemów za pomocą obiektów i operatorów umożliwia zastosowanie metod heurystycznego przeszukiwania.

Przy rozwiązywaniu problemów często korzysta się z informacji niepewnych i nieprecyzyjnych, szczególnie wtedy, gdy przetwarzana jest informacja pochodząca że świata rzeczywistego. Niepewność polega na tym, że nie można jednoznacznie określić czy są one prawdziwe czy fałszywe. Ocena badanych obiektów jest więc często subiektywna, zależy ona od reguł wypracowanych doświadczalnie (opinia ekspertów). Nieprecyzyjność przejawia się w tym, że nie możliwe jest jednoznaczne umieszczenie informacji w obrębie przyjętej skali wartości.

Za pomocą metod heurystycznych można w naturalny sposób wykorzystać informację niepewną i nieprecyzyjną. Przy definiowaniu heurystyk opierających się na doświadczeniu, duże znaczenie ma zrozumienie i uwzględnienie uwarunkowań rozwiązywanego zadania.

Główne zadanie heurystyki polega na usprawnieniu algorytmu rozwiązywania danego problemu. Najważniejszym jest eliminowanie z dalszych rozważań tej części obiektów, które nie zostały jeszcze sprawdzone. Jakość zastosowanej heurystyki ma wpływ na złożoność rozwiązania problemu. Natomiast za najważniejszą funkcję heurystyki uważa się kierowanie procesem decyzyjnym bezpośrednio (przez wskazanie najlepszych kierunków poszukiwania rozwiązania) lub pośrednio (przez eliminowanie najmniej obiecujących kierunków).

Cechą, która budzi najwięcej kontrowersji (spośród wymienionych), jest ta, która mówi, że heurystyka nie gwarantuje znalezienia rozwiązania. Mimo to, to właśnie ta cecha dla wielu autorów ma zalety. Twierdzą oni, że postęp w dziedzinie rozwiązywania problemów jest opóźniany przez nadmierne wykorzystywanie zupełnych reguł decyzyjnych, tzn. takich, które nie eliminują z rozwiązań żadnych kierunków prowadzących do rozwiązania (o ile takie istnieje). Najnowsze metody przeszukiwania heurystycznego mają stanowić pomost pomiędzy zupełnością algorytmów a ich optymalną złożonością. Strategie modyfikowane są w kierunku wyznaczenia rozwiązania quasi-optymalnego (zamiast optymalnego) ze znaczną redukcją kosztów [4].

KRZYSZTOF SIUZDAK **WROCŁAW 18.01.99r.**
IV rok IO
specjalność : bazy danych
nr indeksu : 76855

Charakterystyka przeszukiwania heurystycznego:

Wykorzystanie heurystyki jako strategii kontrolnej lub zarządzającej wyborem najbardziej obiecujących kierunków przeszukiwania (bezpośrednio lub pośrednio) jest najważniejsze we wszelkich zadaniach przeszukiwania. Przy rozstrzygnięciach heurystycznych może być wykorzystana wiedza „nieformalna", której zastosowanie nie daje całkowitej pewności jej użyteczności. Czynniki te zostały jednak uwzględnione podczas formowania definicji (określenia) przeszukiwania heurystycznego.

„Heurystyczne", w zadaniach przeszukiwania, określa się wszelkie prawa, kryteria, zasady i intuicje (również takie, których skuteczność nie jest całkowicie pewna), które pozwalają jednak wybrać najbardziej efektywne kierunki działania z punktu widzenia osiągnięcia zamierzonego celu.

Celem poprawienia efektywności systemu rozwiązującego dany problem dołącza się do niego heurystykę. Wynikiem zastosowania praktycznych reguł (często intuicyjnych i empirycznych), ściśle związanych z dziedziną, z której pochodzi problem daje lepszą jakość rozwiązania.

Klasyczne strategie przeszukiwania:

Teraz chciałbym przedstawić przegląd klasycznych i najpopularniejszych strategii przeszukiwania grafów. Później postaram się pokazać strategie najczęściej stosowane.

Reprezentacja zadań i przestrzeń przeszukiwani:

Czynności wykonywane podczas rozwiązywania zadań można sklasyfikować jako przeszukiwanie lub jako konstruowanie obiektów o zadanej charakterystyce. Aby rozwiązać określone zadanie trzeba często przeszukać zbiór wszystkich możliwych obiektów, zwany przestrzenią przeszukiwania. Całkowite przebadanie tej przestrzeni jest mało efektywne (choć często realizowane praktycznie) i możliwe jedynie w przypadku zbiorów o niewielkiej liczbie obiektów.

Generowanie obiektów według określonych zasad i badanie ich własności jest metodą o wiele efektywniejszą, dla dużych zadań. Zasady generowania obiektów nazywane są operatorami. Operatory i sytuacja początkowa definiują drzewo obiektów (przestrzeń przeszukiwania).

Najistotniejszymi i podstawowymi wymaganiami (z punktu widzenia opisu zadania) są trzy wymagania:

- kod - sposób reprezentowania każdego obiektu przestrzeni przeszukiwania;
- operator - metoda obliczeniowa pozwalająca wygenerować kod kolejnego obiektu na podstawie kodu danego obiektu;
- strategia sterowania - metoda wyboru operatorów spośród operatorów możliwych do zastosowania.

Najbardziej pożądanymi cechami kodu obiektu są: **jednoznaczność i uwzględnienie struktury zadania**. Oznacz to, że kod obiektu powinien reprezentować indywidualne cechy obiektu a także podzbiór potencjalnych rozwiązań związanych z obiektem. Reprezentacja taka umożliwia efektywne transformacje zbioru za pomocą operacji rozszczepiania. Polega ona na podziale problemu reprezentowanego przez dany obiekt na podproblemy, odrzucenie części obiektów i badanie jedynie najbardziej obiecujących . Kolejne rozszczepienia mogą sprowadzić problem początkowy do łatwo rozwiązywalnego problemu.

Przedstawiony powyżej sposób rozwiązywania zadań nazywany jest **metodą rozszczepienia i odrzucenia** (ang. **split-and-prune**). W dziedzinie **sztucznej inteligencji**, z

powodu zadań o dużych rozmiarach, stosuje się analogiczne reguły postępowania nazywane **metodą generowania i sprawdzania** (ang. **generate-and-test**). Zamiast odrzucania obiektów pewnego zbioru, generuje się nowe obiekty i tylko część z nich jest wykorzystywana do dalszych badań.

Głównymi transformacjami stosowanymi w omawianych strategiach przeszukiwania są generowanie i testowanie. Ich efektywne wykonywanie wymaga, aby kody obiektów spełniały pewne dodatkowe warunki. Zakłada się, że kod danego obiektu posiada dodatkowe informacje, które definiują wprost podproblem określony przez ten obiekt. Kod, który ma taką własność nazywa się **kodem stanu**. Przestrzeń rozwiązań jest więc przestrzenią stanów. Natomiast graf , który jest określony przez stan początkowy i operatory jest, w tym przypadku, grafem stano przestrzennym. Dalsza część zawierać będzie opis strategii stosowanych przy przeszukiwaniu grafów stano przestrzennych.

Przeszukiwanie grafów stano przestrzennych:

Główną operacją jest generowanie i testowanie. W większości zadań praktycznych, a szczególnie w problemach związanych ze **sztuczna inteligencją**, rozmiary przeszukiwanych zbiorów są tak duże, że niemożliwe jest zapamiętanie wszystkich ich elementów. Dlatego zamiast rozszczepiania należy stosować generowanie. Prezentowane strategie są nadal systematyczne, jednak charakteryzują się mniejszymi wymogami pamięciowymi.

1. Systematyczne przeszukiwanie grafów:

Podstawowym algorytmem stosowanym przy rozwiązywaniu różnego rodzaju zadań wykorzystujących grafy jest właśnie systematyczne przeszukiwanie grafów. Rozpoczyna się ono od węzła początkowego „**p”**. Głównym zadaniem algorytmu jest sprawdzenie wszystkich węzłów grafu. Podczas wykonywania algorytmu zbiór sprawdzanych krawędzi „**S”**, który pierwotnie zawiera tylko węzeł początkowy „**p”**, jest stale powiększany. Systematyczność badania polega na tym, że w kolejnym kroku analizowane są te krawędzie, które wywodzą się z węzłów należących do „**S”**, ich węzły końcowe wpisywane są do „**S”**, a krawędzie oznaczane są jako wykorzystane (sprawdzone). Algorytm kończy się, gdyż żaden węzeł w „**S”** nie ma niewykorzystanych krawędzi.

2. Strategia w głąb:

Jest ona niezależna od zadania związanego z przeszukiwanym grafem i jest oczywiście „ślepa”, tzn. nie wykorzystuje informacji. Najbardziej typowymi reprezentacjami zbioru „**O”** to: lista, stos lub kolejka. Jeżeli zbiór „**O”** jest stosem, to strategia przeszukiwania nosi nazwę **w głąb** (ang. **depth-first**). Nazwa ma podkreślać kolejność przeszukiwania węzłów. Pojedynczą drogę bada się tak długo, aż jej ostatni element (węzeł) nie zostanie uznany za węzeł celu lub końcowy. Przeszukiwanie prowadzi się zawsze od ostatnio sprawdzanego węzła, który ma jeszcze niewykorzystane krawędzie.

Oczywiście strategia w głąb może być nieskuteczna , gdy zastosuje się ją do dużych grafów, a szczególnie dla grafów o nieskończonej głębokości. Z tego względu strategia ta jest rozszerzana o zasadę kontroli ograniczenia głębokości. Powrót następuje wtedy, gdy głębokość badanego węzła przekracza ograniczenie lub węzeł końcowy spełnia własność końcową

3. Strategia z powracaniem (backtracking):

Strategia ta jest modyfikacją algorytmu przeszukiwani w głąb. Zbiór „**O”** ma w tej strategii również postać stosu. Jednak ekspansja badanego węzła jest zastąpiona jego

rozszerzeniem - generowaniem pojedynczego potomka. Dla wybranego węzła generowany jest tylko jeden potomek i jeżeli ten nowy węzeł nie spełnia kryterium celu lub końcowego, to jest dalej rozszerzany (tylko ten jeden potomek). Gdy po kolejnym rozszerzeniu otrzymany węzeł spełnia kryterium końca lub nie można już dla niego wygenerować nowego potomka, wtedy w strategii następuje powrót do najbliższego (dla danego węzła) przodka, dla którego możliwe jest wygenerowanie potomka.

Główną zaletą strategii jest oszczędność pamięci (w stosunku do strategii w głąb). Zamiast zapamiętywanie wszystkich potomków węzła, przechowuje się tylko jeden. Strategia gwarantuje, że po jej zakończeniu (po napotkaniu węzła celu lub przebadaniu wszystkich węzłów listy „**O”**) wszystkie węzły są przetestowane.

4. Strategia w szerz:

W przeciwieństwie do poprzednio opisywanego algorytmu, strategia **w szerz** (ang. **breadth-first**) kolejno badając poziomy grafu o jednakowej głębokości, przyznaje wyższy priorytet węzłom o mniejszej głębokości. Lista „**O”** ma w tej strategii strukturę kolejki i do analizy kierowane są te węzły, które najdłużej przechowywane były w pamięci.

Główną operacją strategii jest ekspansja węzłów. Lista „**O”** o strukturze kolejki gwarantuje, że dla lokalnie skończonego grafu (każdy węzeł ma skończoną liczbę potomków) strategia osiągnie rozwiązanie (węzeł celu), o ile ono istnieje.

5. Niesystematyczna strategia zachłanna (hill-climbing):

Główną operacją strategii jest ekspansja węzłów. Po jej wykonaniu badane są nowe węzły i najbardziej obiecujący z nich wybierany jest do dalszej ekspansji. Strategia zachłanna wykorzystuje lokalną optymalizację i nie są możliwe w niej nawroty żadnego aktualnie badanego węzła. W tym sensie jest ona nieodwracalna i oczywiście nie jest systematyczna.

Postępowanie podejmowane w tej strategii jest podobne do akcji turysty atakującego szczyt wzniesienia. Chcąc jak najszybciej osiągnąć sukces wybiera on aktualnie najlepszy kierunek, chociaż w trakcie dalszej wspinaczki decyzja ta może okazać się błędna i kierunek w danej chwili gorszy może być w perspektywie całej wspinaczki lepszy. Jej wadą jest brak możliwości powrotu do kierunków przeszukiwania, które na pewnym etapie były lokalnie gorsze. Może to oczywiście prowadzić do badania drogi prowadzącej do węzła końcowego nie spełniającego kryterium celu lub do penetrowania drogi nieskończonej.

W tej części projektu chciałbym przedstawić i opisać czym jest schemat i drzewo gry logicznej. Długi czas w konstruowaniu gier głównym wyznacznikiem postępu sztucznej inteligencji był sukces zapewniający powodzenie takich gier. Obecnie rozwiązuje się przeważnie problemy praktyczne (programy gier), to wiedza jaką zdobyto przy tej okazji ma duże znaczenie dla analizy metodologii rozwiązywania złożonych zadań. To właśnie teoria gier umożliwiła oszczędne sprawdzenie pomysłów, intuicji, ekspertyz użytecznych w planowaniu, robotyce, nauce, problemach administracyjnych i technologicznych. Strategie gier są pomocne także przy analizie automatycznego generowania funkcji heurystycznych.

Schemat i drzewo gry:

Ogólnie idea gier logicznych w dużym stopniu pokrywa się z koncepcją typowych gier towarzyskich. Po przyjęciu pewnego stanu początkowego następuje ciąg kolejnych posunięć. W każdym z nich jeden z graczy dokonuje wyboru spośród kilku możliwych. Posunięcia wykonywane przez graczy mogą być także losowe, np. rzut kostką do gry albo tasowanie talii

kart. Typowymi przykładami gier są: szachy - bez posunięć losowych, brydż - przypadek odgrywa pewną rolę oraz ruletka, która jest grą całkowicie losową.

Najważniejszymi elementami schematu gry są:

- istnienie stanu początkowego,
- określenie funkcji wypłaty związanej z każdym stanem końcowym gry (w formie pieniężnej, punktowej, satysfakcji lub prestiżu),
- podział posunięć na losowe lub wykonywane przez poszczególnych graczy,
- charakterystyka randomizacji posunięć losowych,
- podział zbioru posunięć na podzbiory informacyjne, które uwzględniają stopień wiedzy gracza o pozycji gry.

W dalszej części przedstawię metody przeszukiwania użyteczne przy wyznaczaniu strategii rozwiązywania dla gier dwuosobowych z pełną informacją. W takiej grze o posunięciu decyduje wyłącznie gracz, dysponuje on kompletną informacją o stanie rozgrywki. Zakłada się również, że dwóch graczy kolejno i na przemian wykonuje posunięcia. Sukcesem jednego gracza jest porażka drugiego - przeciwnika (ograniczenie funkcji wypłaty). Rozważania ograniczą się do skończonych gier, w których wykonana może być tylko skończona liczba posunięć (może być bardzo duża) i każdy gracz dysponuje skończoną liczbą strategii. Założenia te spełniają takie gry: szachy i warcaby.

Początkiem gry jest pewien stan początkowy, a gra kończy się stanem, który według określonego kryterium może być uznany za wygrany, przegrany lub remisowy (stan wygrany lub przegrany z punktu widzenia jednego z graczy). Do każdego stanu końcowego można przypisać funkcję wypłaty. Przyporządkowuje ona węzłowi końcowemu wektor dwuwymiarowy (jedna współrzędna charakteryzuje wypłatę gracza, a druga przeciwnika). Bez straty ogólności rozważania można ograniczyć do gier dwuosobowych o średniej zerowej (zerowa suma współrzędnych wektora wypłaty). Wykonujący pierwszy ruch (ze stanu początkowego) został nazwany graczem, a drugi przeciwnikiem. Określenie stanów końcowych dokonywane jest z punktu widzenia gracza.

Za cel algorytmu przeszukiwania obrano wyznaczenie strategii gry, czyli konkretnego planu rozgrywki. Chodzi o określenie kolejnych posunięć gracza dla różnych posunięć przeciwnika. Proces przeszukiwania można opisać za pomocą drzewa. Każdy węzeł drzewa odpowiada określonemu stanowi gry. Korzeniem jest początkowy stan gry, jego potomkowie reprezentują stany osiągalne przez gracza w pierwszym ruchu. Z kolei ich potomkowie to stany możliwe do osiągnięcia przez przeciwnika itd. Natomiast węzły końcowe, liście drzewa, reprezentują stany określane jako: wygrana, przegrana lub remis dla gracza. Każde przejście z węzła początkowego do końcowego jest jedną rozgrywką (opisuje ją). Drzewo gry jest reprezentacją wszystkich możliwych do uzyskania rozgrywek, a przeszukiwanie można interpretować jako proces budowy takiego drzewa.

Łatwo można zauważyć podobieństwo pomiędzy drzewami gier a drzewami **AND/OR**. Odpowiednikiem węzłów **OR** są stany gry na poziomie gracza. Aby wyznaczyć strategię wygrywającą wystarczy znaleźć strategię wygrywającą, która rozpoczyna się chociaż od jednego z potomków węzła **OR**. Natomiast stany na poziomie przeciwnika są odpowiednikami węzłów **AND**. Do drzewo strategii wygrywającej powinni należeć wszyscy potomkowie tych węzłów.

Liście drzewa gry mają przypisane trzy stany: wygrany, przegrany lub remisowy (ze względu na współrzędną gracza w wektorze wypłat). Dlatego każdy węzeł drzewa można opisać etykietą: **WYGRANY**, **PRZEGRANY**, **REMISOWY**. Etykietowanie węzłów jest oparte na zasadzie etykietowania węzłów grafów **AND/OR**. Jeżeli „**w**” jest niekońcowym węzłem określającym stan osiągnięty przez gracza, to **STAN(w)** jest określany następująco:

KRZYSZTOF SIUZDAK **WROCŁAW 18.01.99r.**
IV rok IO
specjalność : bazy danych
nr indeksu : 76855

$$\textbf{STAN(w)} = \begin{cases} \textbf{WYGRANY}, \text{ gdy jakiś następnik „}\textbf{w}\text{” ma stan } \textbf{WYGRANY} \\ \textbf{PRZEGRANY}, \text{ gdy każdy następnik „}\textbf{w}\text{” ma stan } \textbf{PRZEGRANY} \\ \textbf{REMISOWY}, \text{ gdy jakiś następnik „}\textbf{w}\text{” ma stan } \textbf{REMISOWY}, \text{ a żaden nie ma stanu } \textbf{WYGRANY}. \end{cases}$$

Jeżeli niekońcowym węzeł „**w**” odpowiada pozycji przeciwnika, to **STAN(w)** wyznaczany jest według następującej procedury:

$$\textbf{STAN(w)} = \begin{cases} \textbf{WYGRANY}, \text{ gdy każdy następnik „}\textbf{w}\text{” ma stan } \textbf{WYGRANY} \\ \textbf{PRZEGRANY}, \text{ gdy jeden z następników „}\textbf{w}\text{” ma stan } \textbf{PRZEGRANY} \\ \textbf{REMISOWY}, \text{ gdy jeden z następników „}\textbf{w}\text{” ma stan } \textbf{REMISOWY}, \text{ a żaden nie ma stanu } \textbf{PRZEGRANY}. \end{cases}$$

Proces przekazywania wektorów wypłat można opisać w sposób analogiczny do przedstawionych procedur wyznaczania stanów węzłów. Należy jedynie uwzględnić następujące przyporządkowanie (dla gry dwuosobowej o sumie zerowej):

$$\textbf{E(w)} = \begin{cases} (m, -m), & \text{gdy } \textbf{STAN(w) = WYGRANY}, \\ (0, 0), & \text{gdy } \textbf{STAN(w) = REMISOWY,} \\ (-n, n), & \text{gdy } \textbf{STAN(w) = PRZEGRANY.} \end{cases}$$

gdzie m, n > 0 wyznaczają wielkość wypłaty osiąganej z wierzchołka „**w**”.

Podobnie jak w przypadku grafów **AND/OR** rozwiązanie drzewa gry następuje w momencie określenia stanu korzenia (węzła początkowego) jako **WYGRANY**, **PRZEGRANY**, **REMISOWY** lub przyporządkowania mu wektora wypłaty **E(.)**. Z każdą etykieta węzła początkowego związana jest strategia gry - drzewo rozwiązania - gwarantująca osiągnięcie przez gracza wyniku nie gorszego niż wartość etykiety korzenia.

Dla większości gier drzewo gry składające się ze wszystkich możliwych rozgrywek jest bardzo duże. Na przykład dla warcabów zawiera ono 10 do potęgi 40 węzłów, a dla szachów 10 do potęgi 120 węzłów. W tych warunkach nie jest możliwe zbudowanie drzewa gry i poszukiwanie w nim rozwiązania w praktyce. Tak jak w przypadku opisywanych do tej pory strategii przeszukiwania, w teorii gier preferowane są więc algorytmy generujące węzły z odrzuceniem części z nich (w myśl przyjętych kryteriów. Zbadane drzewo ma znacznie mniejsze wymiary niż pełne drzewo gry. Z powyższego wynika, że najczęściej badane gry są: dwuosobowe, skończone, z pełną informacją, z sumą zerową.

Ogólna koncepcja przeszukiwania heurystycznego

W przedstawionych do tej pory strategiach przeszukiwania najbardziej istotnymi punktami, ze względu na złożoność obliczeniową, były wybór kolejności badania dróg grafu oraz wybór porządku analizy nowo generowanych potomków. Zwłaszcza w przypadku

dużych przestrzeni stanów, wczesne odrzucenia nieobiecujących kierunków przeszukiwania zapewnia duże oszczędności kosztów, a więc w rezultacie szybsze osiąganie rozwiązania.

Strategie przedstawione poprzednim razem miały charakter uniwersalny. Można poprawić ich skuteczność poprzez dopasowanie kierunku przeszukiwania do potrzeb aktualnie rozpatrywanego problemu. W ten sposób następuje zastąpienie strategii ślepych, nie wykorzystujących informacji o zadaniu, strategiami skierowanymi i uwzględniającymi informację o przestrzeni stanów, operatorach i kryteriach celu.

W strategiach ślepych kierunek przeszukiwania zależy wyłącznie od informacji jakie dostarczają już zbadane węzły (stany) grafu i sam proces przeszukiwania. Drugim rodzajem strategii są takie, których analizą zajmę się teraz. Pozwalają one uwzględnić (choćby częściowo) informacje o niezbadanej jeszcze części grafu. W szczególności wiedza o dziedzinie danego problemu może być przydatna przy wyborze najbardziej obiecujących kierunków.

Informacje określające problem nie zawsze są precyzyjne i ściśle sformułowane. Częstym przypadkiem jest, że osoba opisująca dany problem opiera je na własnym doświadczeniu lub na „niesformalizowanych" faktach. Jednak heurystyka powinna wykorzystywać właśnie taki rodzaj informacji. Dlatego też proste uogólnienie i rozszerzenie na inne problemy strategii wykorzystujących informację heurystyczną, charakterystyczną dla innego problemu (zadania), nie jest możliwe.

P. H. Winston, dzięki analogii do strategii zachłannej, sformułował metaforyczny opis idei przeszukiwania heurystycznego. Porównuje on strategię heurystyczną do grupy turystów górskich, którzy dążą do zdobycia szczytu wzgórza. Jednak aby lepiej im to poszło, to oni współpracują ze sobą. Wykorzystują w tym celu kontakt radiowy i posuwają bez ustanku najwyżej położoną podgrupę naprzód, dzieląc się przy tym na mniejsze podgrupy na skrzyżowaniach badanych szlaków. Jest wiadomo jeszcze, że turyści każdej grupy dysponują przyrządami pozwalającymi oceniać wysokości oraz analizować trudności drogi, która pozostała im jeszcze do przebycia. Oczywiście oceny i pomiary mogą być błędne, podobnie jest czasami z informacją heurystyczną.

Teraz postaram się przedstawić ogólne strategie opisujące informację w postaci funkcji numerycznej. Na numeryczną wartość funkcji heurystycznej **f(w)**, dla danego węzła **w**, największy wpływ mają:

- ocena jakości tych węzłów rozwiązania, dla których przodkiem jest węzeł **w**;
- ocena złożoności problemu określanego poprzez węzeł **w**;
- ocena ilości i jakości informacji wnoszonych przez węzeł **w** oraz porównanie ich z informacjami wnoszonymi przez inne węzły wykorzystywane w strategii przeszukiwania.

Wprowadzenie ogólnej funkcji heurystycznej jest niezbędne przy kwalifikowaniu i charakteryzowaniu strategii. Trzeba jednak zwrócić uwagę, że jej praktyczne wykorzystanie wymaga odpowiedniego dopasowania do żądań konkretnego problemu.

Heurystyczna strategia „najpierw najlepszy":

W tej części projektu przedstawię ogólną strategie heurystyczną przeszukiwania **„najpierw najlepszy"** (ang. **best first**). Strategia ta wykorzystuje w sposób naturalny, przy podejmowaniu decyzji o badaniu następnego węzła, informację heurystyczną. Dalszemu rozszerzaniu poddawany jest węzeł „najlepszy" spośród wszystkich węzłów rozpatrywanych do tej pory, niezależnie od ich umiejscowienia w grafie.

W strategii **„najpierw najlepszy"** wartość numeryczna funkcji heurystycznej **f(w)** wyraża ocenę węzła w ze względu na kryteria:

- **zbieżności** - czyli osiągnięcia węzła celu;

- **najmniejszego kosztu drogi** wyznaczanej od węzła początkowego, przez węzeł **w**, do węzła celu;
- **najmniejszej złożoności obliczeniowej** procesu przeszukiwania.

Prezentowana strategia ma charakter uniwersalny. Jedynym ograniczenie jakie zostało przyjęte (nie zmniejszające ogólności) jest to, że węzeł najbardziej obiecujący ma najmniejszą wartość funkcji **f(.)**. Strategia **„najpierw najlepszy”** przedstawiona jest jako procedura **HPBF(p, C, K, f):**

```
procedure HPBF(p: węzeł początkowy;
               C: własność celu;
               K: własność końcowa;
               f: funkcja heurystyczna);
begin
 Rozszerzenie(O, {p});
 while O<>0 do begin
  a:= Wybór_HBF(O);
  {procedura Wybór_HBF(O) wyznacza najlepszy (pod względem wartości funkcji
   heurystycznej f) węzeł z listy O; spełniony jest wzór f(a)=min f(a),
   gdzie w należy do O}
  Usunięcie({a}, O);
  Rozszerzenie(Z, {a});
  G:= Generacja_HBF(a);
  {funkcja Generacja_HBF(a) dokonuje ekspansji węzła a, wszyscy potomkowie
   węzła a tworzą zbiór G}
  for b - wszystkie elementy G do
   if Test(b, C) then begin
        Wyjście_z_sukcesem;
        goto K;
   end;
  for b - wszystkie elementy G do
   if Test(b, K) then begin
        Uporządkowanie_HBF(Z);
        {procedura Uporządkowanie_HBF(Z) usuwa z listy Z przodków węzła b nie
         mającego potomków na liście O}
        goto KK;
   end;
  if b nie należy do O and b nie należy do Z then begin
   Wyznaczenie_f(b);
   {procedura Wyznaczenie_f(b) oblicza nową wartość funkcji heurystycznej
    f dla węzła b}
   Rozszerzenie(O, b);
  end
  else begin
   Wyznaczenie_f(b);
   Modyfikacja_f(b);
   {procedura Modyfikacja_f(b) porównuje nowo obliczoną wartość funkcji
    heurystycznej z poprzednio przypisaną węzłowi b; mniejsza z wartości
    przyporządkowana jest węzłowi b; w ścieżce rozwiązania węzeł b ma tego
```

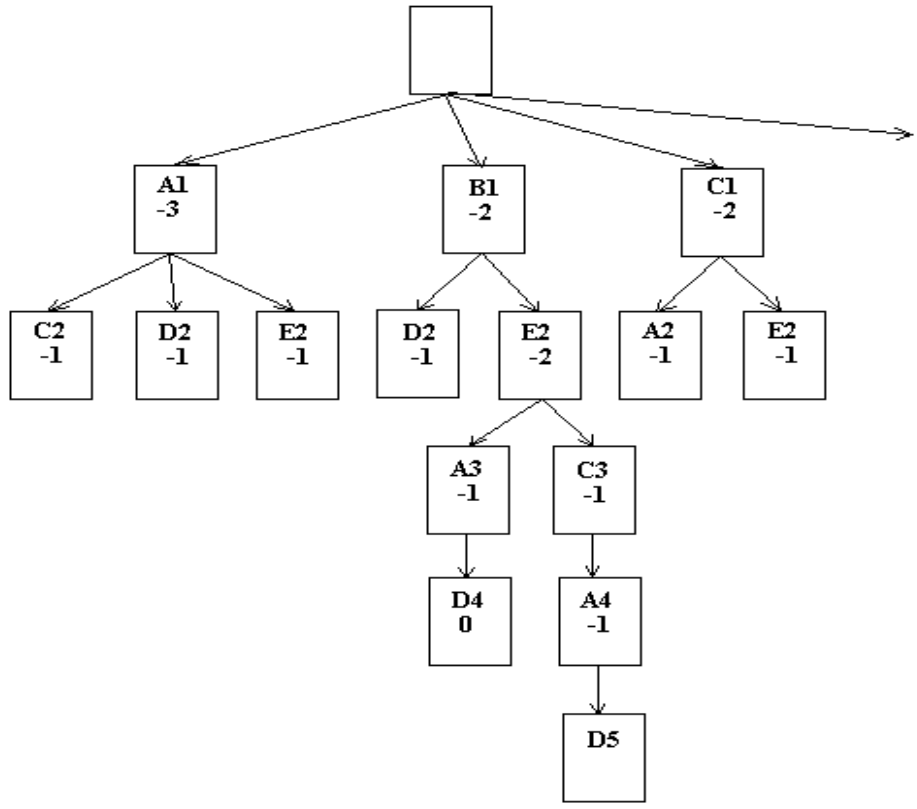
```
      przodka, który zapewnia tą mniejszą wartość f(b)}
     if b należy do Z then begin
          Rozszerzenie(O, {b});
          Usunięcie({b}, Z);
      end;
   end;
KK:end;
  Wyjście_z_niepowodzeniem;
K:end;
```

Działanie strategii **„najpierw najlepszy”** Postaram się pokazać na przykładzie pięciu hetmanów (**Rysunek 1.**).

5				**Q**	
4	**Q**				
3			**Q**		
2					**Q**
1		**Q**			
	A	**B**	**C**	**D**	**E**

Rysunek 1. działanie strategii **„najpierw najlepszy”** dla zadania pięciu hetmanów;

Celem tego zadania jest rozmieszczenie pięciu hetmanów na szachownicy o 25-ciu polach w taki sposób, aby żaden nie atakował innego. Na **rysunku 2.** przedstawiona została część grafu wygenerowanego przez procedurę **HPBF(p, C, K, f)**.



Rysunek 2. górne oznaczenie węzłów charakteryzują pozycje kolejnych hetmanów, dolne liczby to wartości funkcji heurystycznej;

Zaczynając od pustej szachownicy (stanu początkowego **p**) hetmany umieszcza się w kolejnych wierszach o numerach od 1 do 5. Węzły na każdym poziomie drzewa reprezentują możliwe pozycje hetmanów w kolejnych wierszach. Do oceny aktualnej pozycji wykorzystuje się funkcję heurystyczną **f**. Jej wartością bezwzględną jest minimalna liczba nie atakowanych pól w wolnych wierszach (pozycja jest bardziej obiecująca, gdy liczba ta jest większa). W procedurze **HPBF(p, C, K, f)** kolejność węzłów jest ustalana według minimalnych wartości funkcji heurystycznej i dlatego wartości f są liczbami ujemnymi. Na **rysunku 2.** oznaczona jest pozycja reprezentowana przez każdy węzeł oraz wartość funkcji **f** związanej ze stanem zadania określonym przez ścieżkę od **p** do węzła. Przedstawione jest również rozwiązanie końcowe wyznaczone przez **HPBF(p, C, K, f).**

W strategii **„najpierw najlepszy”** najważniejszą operacją, odróżniającą ją od klasycznych algorytmów, jest uporządkowanie listy **O** według kolejności wartości funkcji heurystycznej. Rozszerzenie węzłów wykonywane jest, podobnie jak w strategii **„najpierw głębokość”**, poprzez ekspansję (generowanie wszystkich potomków). Lista **O** zawiera węzły aktualnie badane, natomiast na liście **Z** znajdują się węzły już rozszerzone. Podczas wykonywania procedury Uporządkowanie_HBF(Z) lista Z w każdym momencie przeszukiwania tworzy ścieżkę od węzła początkowego do węzła aktualnie sprawdzanego.

Oprócz opcjonalnie wykonywanej procedury Uporządkowanie_HBF(Z), mającej na celu oszczędność pamięci, w procedurze **HPBF(p, C, K, f)** wykorzystywana jest także Modyfikacja_f(b) i związane z nią przeglądanie list **O**, **Z** oraz przenoszenie węzłów z listy **Z** na listę **O**. Celem tej operacji jest uniknięcie wielokrotnego badania tych samych węzłów. Gwarantuje to oszczędność pamięci, gdyż te same węzły nie są zapisywane wielokrotnie (z różnymi dowiązaniami). Z tego względu w wielu praktycznych zastosowaniach strategii **„najpierw najlepszy”** pomijana jest (kosztem większego zużycia pamięci) kontrola powtarzania się węzłów.

N. J. Nilsson w swojej pracy zaproponował inny sposób uniknięcia powtórnego badania węzłów. Zamiast przenoszenia węzłów z listy **Z** na listę **O** zmieniane są jedynie dowiązania węzłów w grafie (dokładniej w przeszukiwanej części grafu). Jeżeli nowo wyznaczona wartość funkcji **f(b)**, dla węzła potomka węzła **a**, jest mniejsza od wartości poprzedniej , dla węzła **b** jako potomka uprzednio badanego węzła **w**, to odrzucane jest dowiązanie węzłów **w** i **b**. Wymaga to oczywiście również zmiany wartości funkcji heurystycznej **f** dla już badanych potomków węzła **b**.

W strategii **„najpierw najlepszy”** z modyfikacją **N. J. Nilssona** nie wykorzystuje się obliczeń związanych z powtórną ekspansją węzła skierowanego na listę **O** (oraz być może już badanych jego potomków), jednak dzieje się to kosztem wykonywania zmiany dowiązań dla wielu węzłów oraz modyfikacji wartości funkcji heurystycznej.

Modyfikacje strategii „najpierw najlepszy”:

W tej części przedstawię niektóre wyspecjalizowane wersje powyższej strategii. Zaprezentuję najczęściej stosowane metody, w których funkcja heurystyczna jest ograniczona dodatkowymi założeniami. Istnieją ograniczenia, które umożliwiają dokładniejszy opis algorytmów dla szczególnych klas problemów (z wykorzystaniem dodatkowej informacji o kierunkach przeszukiwania). Inny rodzaj ograniczeń pozwala zmniejszać koszty wyznaczenia informacji heurystycznej, poprzez osłabienie określających ja wymagań.

W części przedstawiającej ogólną strategię **„najpierw najlepszy”** nie zostały określone sposoby obliczania funkcji heurystycznej. Na skuteczność strategii mają ogromny wpływ rozwiązania przyjęte przy wyborze najlepszych kierunków przeszukiwania. Sposób obliczania funkcji heurystycznej znacząco wpływa również na złożoność obliczeniową całej strategii. Szczególnie ze względu na ten drugi rodzaj uwarunkowań ważne są:

- zapamiętanie obliczeń częściowych - umożliwia to wykorzystanie obliczeń już raz wykonanych przy badaniu nowych węzłów;
- selektywne modyfikowanie wartości funkcji - funkcja heurystyczna obliczana jest tylko dla nowo generowanych potomków danego węzła, a później jest tylko modyfikowana.

Jednym ze składników funkcji heurystycznej jest bardzo często wielkość stanowiąca ocenę jakości drogi rozwiązania związanej z badanym węzłem. Dla węzła w ocena taka opisywana jest numerycznie wagą **W(w)**. Waga **W(w)** reprezentuje koszt oraz jakość drogi (np. koszt lub długość drogi), wiodącej z węzła początkowego **p** poprzez węzeł **w** do węzła spełniającego kryterium celu (lub najlepszego z osiągniętych węzłów celu). W ogólnym przypadku waga jest funkcją zależną od bardzo wielu wielkości w grafie, a między innymi od wag węzłów, kosztów krawędzi, jakości i kosztów poszczególnych węzłów celu.

Ze względów obliczeniowych duże znaczenie mają rekurencyjne funkcje wagowe, w których ocena węzła zależy od oceny potomków. Dokładniej, funkcje wagowe **W(.)** nazywamy rekurencyjną wtedy i tylko wtedy, gdy istnieje taka funkcja kombinatoryczna **F**, że dla każdego węzła **w** (badanego grafu) spełniona jest zależność:

$$W(w) = F\{ L(w); W(w1), W(w2), \dots , W(wk) \},$$

gdzie **w1, w2, ... , wk** są bezpośrednimi potomkami węzła **w**, a wielkość **L(w)** wyraża lokalne własności węzła **w**.

W strategii **„najpierw najlepszy”** za pomocą funkcji heurystycznej podejmowana jest decyzja, który z węzłów listy **O** ma być badany w pierwszej kolejności. Przy decyzji tej uwzględniane są węzły zawarte w drzewie przeszukiwania. Na każdym etapie strategii drzewo przeszukiwania (poddrzewo sprawdzanego grafu) zawiera zbadane do tej pory węzły. Zwroty krawędzi aktualnego drzewa przeszukiwania wyznaczone są przez kolejność wykonywania operacji rozszerzania węzłów lub modyfikacje dowiązań wynikające z przebiegu strategii. Każdy z węzłów **w** listy **O** wyznacza drogę łączącą węzły **p** oraz **w** i zawartą w aktualnym drzewie przeszukiwania. Dzięki zmianom dowiązań wykonywanych w strategii **„najpierw najlepszy”** każdy z takich węzłów w związany jest tylko z jedną drogą od węzła początkowego **p** do węzła **w**. Można więc wyznaczyć wartość funkcji heurystycznej **f(w)** według następującej procedury:

$$f(w) = \begin{cases} h(w) & \text{w na liście O,} \\ \min\limits_{i} F[L(w); f(wi)] & \text{w na liście Z,} \end{cases}$$

gdzie funkcja **h(.)** jest estymacją jakości najlepszego rozwiązania osiągalnego z węzła **w** (im jakość wyższa, tym wartość funkcji **h(.)** mniejsza). Tradycyjnie przyjmuje się, że funkcja heurystyczna **f(.)** reprezentuje koszt przeszukiwania. Jest więc minimalizowana i stąd wynika uzasadnienie przyjętego doboru wartości funkcji **h(.)**. strategia **„najpierw najlepszy”** z zastosowaniem powyższej procedury obliczania funkcji heurystycznej **f(.)** nazywana jest **algorytmem Z**.

Dla potrzeby wyznaczenia rozwiązania optymalnego strategia **„najpierw najlepszy”** musi być odpowiednio przekształcona. Podobnie jak w przypadku algorytmów z poprzedniej części, należy w niej zmienić testy końcowe. Zakończenie nie następuje przy wyznaczeniu

drogi do pierwszego węzła spełniającego kryterium celu, lecz wtedy gdy możemy mieć pewność optymalności (według przyjętej miary). Tak zmodyfikowana strategia **„najpierw najlepszy”** nazywana jest **BF*** (ang. **best first***). W takiej podstawowej wersji strategia **BF*** nie gwarantuje wyznaczenia optymalnego rozwiązania. Zmiany dowiązań węzłów (nieodwracalna operacja wyboru przodków w procedurze Modyfikacja_f) mogą spowodować pominięcie w procesie przeszukiwania drogi do rozwiązania optymalnego.

Strategia łącząca w sobie cechy algorytmów **BF*** oraz **Z** nazywana jest **Z***. Funkcja heurystyczna jest w niej obliczana rekurencyjnie, zgodnie z procedurą określoną dla algorytmu Z. Natomiast testy końcowe są tak zmienione, żeby możliwe było wyznaczenie drogi do rozwiązania optymalnego (tak jak w strategii **BF***).

Strategia A* - heurystyczne przeszukiwanie sieci:

Popularną metodą heurystycznego przeszukiwania jest strategia **A***. Jest to szczególna wyspecjalizowana wersja strategii **Z***, której celem jest wyznaczenie najtańszej drogi w sieci. Funkcja heurystyczna w strategii **A*** jest sumą dwóch składników. Dla węzła **w**, na podstawie informacji heurystycznej, wyznaczana jest estymacja **h(w)** kosztu drogi łączącej **w** z węzłem celu (lub najlepszym węzłem celu). Drugi składnik funkcji heurysxstycznej **g(w)** wyznacza dla węzła **w** koszt drogi łączącej węzeł początkowy **p** z węzłem **w**. Taka budowa funkcji heurystycznej jest jedynym wyróżnikiem strategii **A*** z pośród innych algorytmów klasy **Z*.**

Heurystyczne metody przeszukiwania

Nazwą **A*** określa się często strategię wyznaczającą drogę do pierwszego napotkanego węzła spełniającego kryterium celu [4]. Jej zapis różni się od procedury **HPA*(p, C, K, h, c)** - przedstawionej w Bolcu „Metody wyszukiwania heurystycznego”, brakiem zbioru **2C**. Warto zauważyć, że dla **h**=0 oraz c(**w1**, **w2**)=1, dla wszystkich węzłów **w1**, **w2** szczególnym przypadkiem strategii **A*** staje się algorytm wszerz. Natomiast wybór, funkcji heurystycznej wg zasady f(**p**)=0, f(**b**)=f(**a**)-1, gdy **b** jest bezpośrednim potomkiem węzła **a**, sprowadza strategię **A*** do algorytmu w głąb.

WYBÓR KIERUNKU PRZESZUKIWANIA WG WAŻONEJ FUNKCJI HEURYSTYCZNEJ

W poprzedniej części projektu (tzn. powyżej), przedstawiłem częściowo dla strategii **A***, wpływ estymacji heurystycznej **h** i estymacji kosztu **g** na funkcję heurystyczną **f = h + g**. Elementem wpływu składnika kosztu **g** jest zwiększenie szerokości przeszukiwania. Dla funkcji **f=g**, **A*** redukuje się do strategii jednolitego kosztu (lub wszerz, gdy koszt drogi jest utożsamiony z głębokością jej ostatniego węzła). Natomiast w przypadku **f=h**, algorytm **A*** pomija całkowicie znacznie kosztu lub głębokości pokonanej drogi, wybierając najlepsze kierunki przeszukiwania wyłącznie na podstawie oceny szans dotarcia do węzła celu. Zwiększenie wpływu estymacji heurystycznych na funkcję heurystyczną może przyśpieszyć znalezienie rozwiązania, lecz zawodna estymacja może doprowadzić do niepowodzenia (np. badania ścieżki w grafie lokalnie skończonym).

I . Pohl zaproponował wykorzystanie w strategii **A*** ważonej funkcji heurystycznej $\mathbf{f}_w$ w postaci następującej:

$$\mathbf{f}_w(\mathbf{w}) = (1 - \mathbf{W}) * g(\mathbf{w}) + \mathbf{W} * h(\mathbf{w}) \text{ , gdzie } \mathbf{W} \text{ należy do } [0 ,1].$$

Oczywiste są następujące zależności wiążące wartości wagi **W** i strategie wykorzystujące funkcje $\mathbf{f}_w(.)$:

W = 0 - strategia jednolitego kosztu;
W = 0,5 - strategia **A***;
W = 1 - strategia „**najpierw najlepszy**".

Wybranie innej dowolnej wartości z przedziału [0,1] pozwala na skonstruowanie algorytmów mieszanych łączących własności tych trzech strategii granicznych.

Dla wartości wagi przedziału **W** należącej do [0; 0,5] strategia $\mathbf{A_w}$* z ważoną funkcją heurystyczną wyznacza rozwiązanie optymalne. Jednak dla **W** należącego do [0,5; 1] optymalność wyznaczonego rozwiązania nie jest pewna. W przypadku gdy **W** należy do [0,5; 1] możliwe jest jednak zapewnienie optymalności rozwiązania. Gwarantuje ją zaproponowane przez **I. Pohla** algorytm $\mathbf{A_{Dw}}$* z funkcją heurystyczną ważoną dynamicznie. Waga wiązana z estymacją heurystyczną nie jest stała, lecz zmienia się w trakcie wykonywania algorytmu. Taka koncepcja przeszukiwania pozwala zrealizować funkcja heurystyczna określana wzorem:

$$\mathbf{f}(\mathbf{w}) = \mathbf{g}(\mathbf{w}) + \mathbf{h}(\mathbf{w}) + E * [\ 1 - \mathbf{d}(\mathbf{w}) / \mathbf{D}\ ] * \mathbf{h}(\mathbf{w}),\quad E > 0,$$

gdzie **d**(**w**) oznacza głębokość węzła **w**, a **D** jest oszacownikiem głębokości węzła celu.

Strategia $\mathbf{A_{Dw}}$* z funkcją heurystyczną ważoną dynamicznie jest szczególnie użyteczna dla zadań, w których można dokładnie oszacować głębokość optymalnego węzła celu (szczególnie, gdy głębokość takiego węzła celu jest niewielka). W przypadku **d << D** podany algorytm sprawdza się do algorytmu **A*** wykorzystującego funkcję heurystyczną z wagą.

Dwukierunkowe przeszukiwanie grafów:

W pewnych zadaniach dąży się do wyznaczania pojedynczego, dokładnie zdefiniowanego węzła celu. Jeżeli zmiana stanów może być wykonana w dwóch kierunkach tzn. dopuszczalne jest generowanie potomków każdego węzła (oprócz węzła celu) oraz możliwe jest wyznaczanie rodziców każdego węzła (oprócz węzła początkowego), to przeszukiwanie można również realizować dwukierunkowo. Takie równoczesne przeszukiwanie od węzła początkowego i od węzła celu, zaproponowane przez **I. Pohla** [5], kończy się w momencie dotarcia do węzła osiągalnego z obu kierunków.

Przeszukiwanie dwukierunkowe, w porównaniu z przeszukiwaniem nieukierunkowanym (ślepym), umożliwia znaczną redukcję liczby sprawdzonych stanów. Jeżeli na przykład całkowita złożoność przeszukiwania zależy w sposób wykładniczy od liczby sprawdzonych węzłów **N** (koszt **O(K^N)**), to przeszukiwanie dwukierunkowe może zredukować koszt do poziomu **O(K^(N/2))**. Tak duża redukcja złożoności kompensuje obliczenia wykonane przy sprawdzeniu istnienia węzłów osiągalnych z obu kierunków.

Strategia sprawdzenia dwukierunkowego przedstawiona jest jako procedura **HBD(p, k, hf, hb, c, ce)** (brak przytoczenia treści procedury jest podyktowany tym, że można zobaczyć ją w literaturze [1]). Zmienna **W** określa aktualny kierunek przeszukiwania. Przeszukiwanie od węzła początkowego wykonane jest dla **W** = '**f**', od węzła celu dla **W** = '**b'**, natomiast przy spełnionych kryteriach końcowych zmienna **W** przyjmuje wartość **NIL**.

Procedura **HBD** jest modyfikującą strategii **A***. Funkcja kosztu **C** jest oczywiście wspólna dla obu kierunków, natomiast funkcje heurystyczne $\mathbf{h_f}$ (kierunek od węzła początkowego) oraz $\mathbf{h_b}$ (kierunek od węzła celu) są zazwyczaj różne.

O aktualnym kierunku przeszukiwania decyduje porównanie liczności list $|\mathbf{O_f}|$ oraz $|\mathbf{O_b}|$. Preferowany jest kierunek przeszukiwania, dla którego liczność listy węzłów oczekujących na zbadanie jest mniejsza.

Dodatkowym warunkiem zakończenia procedury **HBD** (z niepowodzeniem) jest przekroczenie zadanego progu „**ce**" przez koszt drogi rozwiązania znalezionej lub przewidywanej. Przy spełnieniu dodatkowych założeń przez funkcję heurystyczną wyjście z niepowodzeniem z procedury **HBD** oznacza, że w danym grafie nie istnieje droga rozwiązania o koszcie mniejszym od „**ce**" (treść procedury do wglądu w literaturze [1]).

PORÓWNANIE:

Porównanie przeszukiwania dwukierunkowego, ze strategiami heurystycznymi jest trudna, ponieważ zależy od jakości zastosowanych heurystyk. **I. Pohl** [5] wykazał, że im jakość heurystyki jest niższa, tym korzyści z zastosowania procedury **HBD** są mniejsze. Redukcja obliczeń w przeszukiwaniu dwukierunkowym jest tym większa, im większe znaczenie przy wyborze kolejności badania węzłów na ich szerokość, (podobnie jak w strategii wszerz preferowana jest szerokość grafu).

Konieczność stosowania dwóch funkcji heurystycznych (z wyjątkiem przypadków, gdy możliwe jest wykorzystanie jednej) oraz zwiększona złożoność obliczeniowa strategii dwukierunkowej powodują, że jest ona stosunkowo rzadko stosowana praktycznie.

Heurystyczne przeszukiwanie grafów AND/OR:

W przypadku grafów **AND/OR** możliwe jest również wykorzystanie heurystyki przy wyborze najlepszych kierunków przeszukiwania. Można także sformułować ogólny (analogiczny do przedstawionego przy omawianiu algorytmu „**najpierw najlepszy**")opis strategii przeszukiwania grafów **AND/OR,** w którym
heurystyka jest wyrażona w postaci funkcji numerycznej.

Heurystyka w uogólnionej dla grafów **AND/OR** strategii „**najpierw najlepszy**") powinna rozstrzygnąć, który z potencjalnych grafów rozwiązania ma być badany oraz w jakiej kolejności sprawdzane mają być węzły tego wybranego grafu. Innymi słowy przeszukiwanie można podzielić na dwa etapy:

1. wybranie bazy rozwiązania, tzn. podgrafu **GO** zawartego w dotychczas przeszukanej części badanego grafu i spełniającego własności:
   - **GO** zawiera węzeł początkowy **p**;
   - jeżeli w grafie **GO** jest rozszerzony węzeł typu **AND**, to wszyscy jego potomkowie są w **GO**;
   - jeżeli w grafie **GO** jest rozszerzony węzeł typu **OR**, to dokładnie jeden jego potomek jest w **GO**;
   - żaden z węzłów grafu **GO** nie ma etykiety „**nierozwiązany**";
2. przeszukiwanie wybranej bazy rozwiązania.

Baza rozwiązania może zawierać wiele węzłów jeszcze nierozszerzonych. Równocześnie pojedynczy węzeł może należeć do wielu baz rozwiązania.

W procedurze **HBDF(p, C, K, f1, f2)** (uogólnionej strategii „**najpierw najlepszy**") wykorzystane są dwie funkcje heurystyczne **f1** oraz **f2**. Funkcja **f1** wskazuje najbardziej obiecujący graf - bazę rozwiązania (uwzględniając trudności poszukiwania oraz jakość ewentualnego rozwiązania znalezionego w wybranym grafie), natomiast funkcja **f2** określa, który z węzłów wybranej bazy rozwiązania ma być rozszerzany w pierwszej kolejności (ma więc rolę zbliżoną do funkcji heurystycznej strategii „**najpierw najlepszy**") - procedura HBDF również znajduje się w literaturze [1]. W przypadku

Strategia gry

Na ogół gracze podejmują decyzję o planie rozgrywki na kilka posunięć na przód, zazwyczaj dopiero w chwili, gdy muszą wykonać posunięcie. W grach takich jak szachy czy warcaby liczba możliwych posunięć jest tak wielka, że nie da się ich zaplanować z góry. Nie możliwe jest również planowanie działań dla wszystkich możliwych ewentualności. Teoretycznie możemy jednak założyć, że każdy gracz wybiera strategię przed rozpoczęciem gry, czyli zawczasu decyduje o posunięciach w każdym przypadku.

Gry dwuosobowe, skończone, z kompletną informacją mają układ strategii w równowadze. Innymi słowy, żaden z graczy nie ma powodu do zmiany swej strategii, gdy tylko przeciwnik nie zmieni swojej. Każdy z graczy w zasadzie pragnie zastosować taką strategię, która doprowadzi do układu równowagi.

Strategia wygrywająca dla gracza wyznaczona jest przez drzewo rozwiązania, którego wszystkie liście mają etykietę **WYGRANY**. Gracz wykonując posunięcie zgodnie ze strategią wygrywającą osiągnie sukces niezależnie od posunięć przeciwnika. Oczywiście analogicznie można opisać strategię wygrywającą dla przeciwnika.

Wyznaczenie strategii wygrywającej zgodnie z podaną definicją wymagałoby analizy całego drzewka gry. Jak już zaznaczyliśmy, w większości gier praktycznych rozmiary drzewa są tak duże, że nie jest możliwe jako systematyczne zbadanie. W niniejszym rozdziale przedstawiono algorytmy przeszukiwania drzew gier szacujące jakość węzłów na podstawie wybranych cech charakteryzujących pozycje gry. Pewne cechy mogą świadczyć o większej randze pozycji (większej szansie nadania węzłowi etykiety **WYGRANY**), inne o mniejszej. Cechy te możemy przedstawić w postaci funkcji wartościującej wierzchołki **e(.)**. Oczywiście osiągnięcia najlepszego wyniku gry możliwe jest dla bardzo dokładnych funkcji wartościujących, w których szacowana ranga pozycji jest prawie taka sama jak rzeczywista.

Sposobem na lepsze oszacowanie rangi pozycji jest wykonanie pewnej rozgrywki. Generowane jest pewne poddrzewo drzewa gry. Za pomocą funkcji wartościującej **e(.)** szacowana jest ranga pozycji reprezentowanych przez węzły na najgłębszym poziomie poddrzewa. Wykorzystując oszacowania wyznaczamy spodziewaną rangę przodków, aż do wyznaczenia takiego oszacowania dla korzenia wygenerowanego poddrzewa.

Przy przyjętych założeniach można przedstawić ogólną postać procedury wyznaczającej strategię gry. Procedura **STRATEGIA(p, WARUNEK_KOŃCA_G, WARUNEK_KOŃCA_P)** jest ogólnym zapisem algorytmów przeszukiwania dla gier:

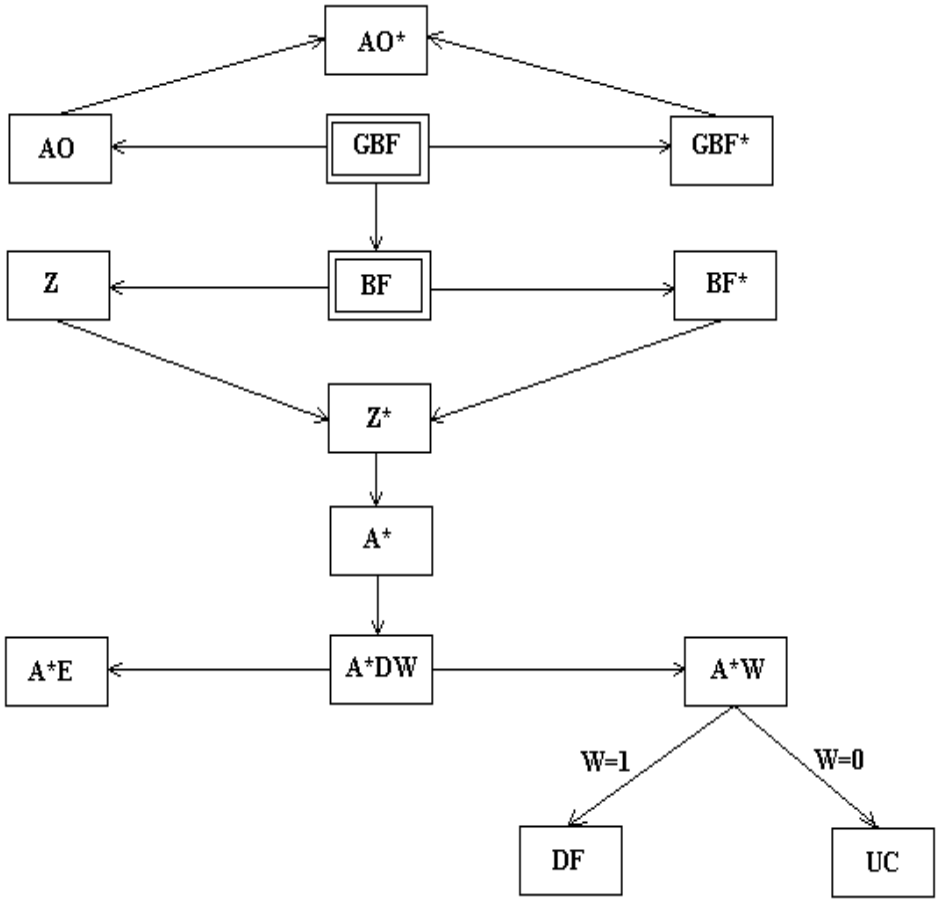
```
procedure STRATEGIA (  p: stan początkowy;
                       WARUNEK_KOŃCA_S : funkcja dla strategii;
                       WARUNEK_KOŃCA_P : funkcja dla posunięcia);
begin
 RORSZERZENIE(D, p);
 { w procesie przeszukiwania generowane jest drzewo gry D }
 while WARUNEK_KOŃCA_S do begin
   v := WYBÓR_WĘZŁA(D);
   { funkcja WYBÓR_WĘZŁA(D) wyznacza stan gry (węzeł) na podstawie gracza
     w celu analizy przed wykonaniem posunięcia (dopisaniem do drzewa D) }
   while WARUNEK_KOŃCA_P do begin
     w := WYBÓR_POTOMKA(v);
     { funkcja WYBÓR_POTOMKA(v) wyznacza stany gry powstałe w wyniku posunięcia v
}
     e(w) := RANGA(w);
```

```
    { funkcja RANGA(w) wartościuje węzeł „w” uwzględniając dostępne informacje
związane        z przodkami węzła „w” (w drzewie D), w zależności od wartości e(w) }
   end;
   W := GENERACJA(w);
   { funkcja GENERACJA(w) wyznacza zbiór stanów W powstałych w wyniku możliwych
     posunięć przeciwnika w reakcji na stan gry "w" }
   ROZSZERZENIE(D, W);
   { funkcja ROZSZERZENIE(D, W) dopisuje posunięcia przeciwnika do drzewa gry D }
  end;
end;
```

Parametrami przedstawionej procedury są stan początkowy gry p oraz dwie funkcje wyznaczające warunki zakończenia poszukiwania strategii (**WARUNEK_KOŃCA_S**) i poszukiwania przesunięcia (**WARUNEK_KOŃCA_P**). Zgodnie z poprzednimi uwagami, do drzewa gry **D** dołączane są węzły (reprezentujące stan gry) uznawane za najlepsze z pozycji gracza oraz wszystkie (lub wszystkie zbadane) węzły reprezentujące stany po posunięciu przeciwnika.

ZALEŻNOŚCI POMIĘDZY STRATEGIAMI PRZESZUKIWANIA GRAFÓW

Na rysunku poniżej (**Rysunek 3.**) przedstawione zostały najważniejsze strategie heurystycznego przeszukiwania i hierarchiczne zależności pomiędzy nimi.



Rysunek 3. Hierarchiczne zależności pomiędzy podstawowymi strategiami przeszukiwania; **GBF** - uogólniona strategia **„najpierw najlepszy”**, **BF** - strategia **„najpierw najlepszy”**, **A*DW** - algorytm **A*** z ważeniem dynamicznym, **A*W** - ważony algorytm **A***, **DF** - strategia w głąb, **UC** - strategia jednolitego kosztu.

Najbardziej ogólnymi są oczywiście „meta-heurystyczne” strategie **„najpierw najlepszy”** - **BF** (ang. **best first**) oraz uogólniona strategia **„najpierw najlepszy”** - **GBF** (ang. **general best first**). Strategia **GBF** dotyczy grafów **AND / OR**, a **BF** grafów stanoprzestrzennych. Istotną różnicą jest jednak sposób badania, przez obie strategie, aktualnego grafu **G**. Strategia **GBF** poszukuje rozwiązania w kolejnych podgrafach, natomiast **BF** konstruuje drzewo rozwiązania. W strategii **BF** podejmowane są, w przypadku napotkania węzła z kilkoma bezpośrednimi przodkami, nieodwracalne decyzje wyboru jednego z rodziców.

Przy rozwiązywaniu problemów optymalizacyjnych stosuje się strategię **GBF*** (grafy **AND / OR**) oraz **BF*** (grafy stanoprzestrzenne). Występujący w strategii **BF*** nieodwracalny sposób wyboru rodziców powoduje, że nie zawsze możliwe jest wyznaczenie rozwiązania optymalnego. Natomiast przy pewnych dodatkowych warunkach dla funkcji heurystycznej strategii **GBF*** wyznacza rozwiązanie optymalne (również dla grafów stanoprzestrzennych).

Rekurencyjne obliczanie funkcji heurystycznych wykorzystywane jest w strategiach **Z**, **AO** oraz **Z***, **AO*** (wyznaczających rozwiązanie optymalne). Ze względów obliczeniowych są to strategie najczęściej stosowane w praktyce.

Strategię przeszukiwania grafów można scharakteryzować w zależności od następujących parametrów:

- stopień możliwości powrotów w strategii,
- liczba alternatyw uwzględnianych przy każdej decyzji.

Przykładowo przestrzeń wszystkich strategii przeszukiwania grafów stanoprzestrzennych w zależności do tych dwóch parametrów tworzy trójkąt, którego wierzchołki wyznaczają strategie:

- **zachłanna** - decyzje nieodwracalne i uwzględnianie tylko ostatnich możliwości wyboru;
- **z powracaniem** - decyzje odwołalne i uwzględnianie tylko ostatnich możliwości wyboru;
- **„najpierw najlepszy”** - decyzje odwołalne i uwzględnianie wszystkich możliwości alternatywnych.

Każda inna strategia przeszukiwania grafów stanoprzestrzennych może być zlokalizowana w zdefiniowanym trójkącie (dla grafów **AND / OR** można przeprowadzić podobne rozważania).

Ze względu na oszczędność obliczeń i wymagań pamięciowych często konstruowane są strategie mieszane, łączące cechy strategii heurystycznych i klasycznych. Przykładowo, popularne są kombinacje strategii **„najpierw najlepszy”** oraz **z powracaniem** [4], [6], zapewniające zmniejszenie wymagań pamięciowych. Możliwe są różne rodzaje takich połączeń. Strategia **„najpierw najlepszy”** może być zastosowana na początku lub na końcu albo lokalnie na pewnych etapach globalnej strategii z powracaniem.

Inny rodzaj kombinacji dwóch strategii zaproponował **Ibaraki** [3]. Polega ona na ograniczeniu liczby węzłów oczekujących na rozszerzenie, elementów listy **O**. Przeszukiwanie przebiega według schematu strategii **„najpierw najlepszy”**, przy czym długość listy **O** jest ograniczona do „n” elementów (dla n=1 powstaje strategia w głąb).

Możliwość zmniejszenia liczby obliczeń w stosunku do oryginalnej strategii **„najpierw najlepszy”** pojawia się przy połączeniu jej ze strategią zachłanną. Nieodwracalność tej drugiej pozwala zredukować liczbę analizowanych alternatyw. Możliwe są oczywiście kombinacje strategii podobne do poprzednio opisywanych. Jednak najbardziej

popularne jest wykorzystywanie strategii **„najpierw najlepszy”**, aż do przekroczenia pewnych progów pamięciowych lub obliczeniowych, a następnie strategii zachłannej. Przełączanie tych dwóch strategii może również następować wielokrotnie.

ANALIZA HEURYSTYCZNYCH STRATEGII PRZESZUKIWANIA

W niniejszej części przedstawione zostaną podstawowe elementy teoretycznego badania strategii przeszukiwania. Przykładową analizę przeprowadzę dla strategii **A*.** Jest to najpopularniejsza strategia heurystycznego przeszukiwania. Ograniczenie rozważań teoretycznych do tej strategii (**A*)** pozwoli również na dokładniejsze przedstawienie zasad analizy heurystycznych metod przeszukiwania. Dodatkowo okaże się, że **A*** ma pewne własności strategii optymalnej.

TEORETYCZNE WŁASNOŚCI STRATEGII PRZESZUKIWANIA.

Nie wszystkie własności strategii przeszukiwania można badać w sposób ogólny. Zależą one między innymi od przeszukiwanego grafu (warunki zadania) oraz jakości zastosowanych metod (funkcje heurystyczne). Przy ich badaniu można zastosować metody probabilistyczne lub kombinatoryczne. Obiektem zainteresowania tego fragmentu są własności stałe w określonych klasach strategii przeszukiwania.

Jedną z najbardziej pożądanych cech algorytmów przeszukiwania jest zbieżność. Strategia przeszukiwania jest zbieżna, gdy gwarantuje wyznaczenie drogi lub grafu rozwiązania, o ile taka istnieje, albo informacji o braku rozwiązania (w przypadku skończonych przestrzeni przeszukiwania). W przypadku nieskończonych przestrzeni przeszukiwania zbieżna strategia musi podawać informację o braku rozwiązania.

Celem wielu zadań jest wyznaczenie rozwiązania zapewniającego spełnienie warunków optymalności lub quasi-optymalności. Strategie gwarantujące osiągnięcie rozwiązania optymalnego (o ile rozwiązanie istnieje) są nazywane dopuszczalnymi (ang. admissible).

Przy porównywaniu różnych strategii wykorzystywane jest pojęcie przewagi - dominacji (ang. dominance) strategii. Strategia $\mathbf{S_1}$ ma przewagę nad strategią $\mathbf{S_2}$, gdy każdy węzeł badany przez strategię $\mathbf{S_1}$ jest również zbadany przez $\mathbf{S_2}$ (strategia $\mathbf{S_2}$ analizuje co najmniej tyle samo węzłów co strategia $\mathbf{S_1}$). Strategia jest nazywana optymalną w pewnej klasie, gdy ma przewagę nad wszystkimi strategiami tej klasy.

Przy badaniu efektywności strategii przeszukiwania duże znaczenie odgrywa ich złożoność. Z praktycznego punktu widzenia na złożoność składają się dwa zasadnicze składniki:

- koszt wykonania znalezionego rozwiązania (związany ze spełnianiem przez strategię warunku dopuszczalności);
- koszt wyznaczenia rozwiązania, na który składają się koszty generowania i sprawdzania węzłów oraz wykrywania niepożądanych kierunków przeszukiwania.

W wielu zadaniach praktycznych oba składniki kosztu i są równoważne. Jakość zastosowanych strategii sterowanie decyduje jednak o wielkości drugiego składników.

Funkcje heurystyczne wpływają oczywiście na każdą z wymienionych cech strategii przeszukiwania - zbieżność, dopuszczalność i złożoność. Głównym zadanie heurystyki jest poprawa jakości rozwiązania i zmniejszenie złożoności jego poszukiwania dzięki ograniczaniu algorytmowych kierunków przeszukiwania badanych strategii. Główną cechą „dobrej” heurystyki jest wykorzystanie przez nią informacji specyficznych dla danego problemu (dopasowanie strategii do konkretnego zadania), a to oczywiście powoduje

trudności w ogólnej analizie własności heurystycznych. Możliwe jest jednak wyodrębnienie pewnych cech ogólnych tych funkcji, pozwalających na klasyfikację i charakterystykę.

Ostatnim (choć nie najmniej ważnym problemem) omawianym w tej części są zasady tworzenia heurystyk, w tym również automatycznego ich generowania.

Przegląd algorytmów heurystycznych wykorzystywanych w grach logicznych

Problemy można rozwiązywać rutynowo bądź twórczo. Gdy problemy rozwiązywane są rutynowo, to rozwiązujący zna całe rozwiązanie lub jego istotne elementy. Gdy problemy rozwiązywane są twórczo, rozwiązanie nie jest znane ani całe, ani jego istotne szczegóły. W początkowym okresie powstawania heurystyki jako nauki wiązano ja głównie z procesami poszukiwania rozwiązań przez intuicję, kontrastującą z logicznym myśleniem. Później przyjęto jednak inne podejście. Zauważono, że służą one (heurystyki) do poszukiwania pomysłów, umiejętności wykrywania nowych faktów i relacji między faktami, dzięki którym dochodzi się do nowych prawd.

J. Rudniański, cytując **A. Molesa**, określa heurystykę jako „tę część wiedzy, która dotyczy odkryć i procesu dokonywania odkryć". W tym ujęciu zajmuję się ona procesem, który prowadzi do nowej wiedzy. Przy tym rozróżnia on „wiedzę osiągniętą" (którą heurystyka się nie zajmuje) i „wiedzę w trakcie powstawania (...), która jest działaniem, procesem, błądzeniem w polu swobody i owa idea pełnej niezależności, wolności, gry jest istotna dla heurystyki". Dalej wskazuje, wiedza osiągnięta tworzy mur książek i faktów, będących zaporą dla wiedzy powstającej i dodaje „badacz czuje się wolny, wolnością niewiedzy, a jeśli obserwator, psycholog lub heurystyk mogą ***a posteriori*** odkryć pewne prawa jego postępowania, prawa natury statystycznej, to jednostka w danej sytuacji nie wie nic". W procesach twórczych płodność heurystyczna zależy od ***intuicji odkrywcy***, choć istotne znaczenie ma też zdolność wytwarzania dobrej komunikacji pomiędzy świadomością i podświadomością, aby świadomość maksymalnie korzystała z usług podświadomości. Dotyczy to zarówno produktów materialnych, jak i działań.

Zagadnienie prawidłowości struktury nowych działań w nowych sytuacjach i zachowania się różnych systemów biologicznych (a zwłaszcza człowieka) podczas rozwiązywania problemów jest rozumienie heurystyki jako nauki. Rozwiązania uwzględniające modelowanie cybernetyczne i informatyczne procesów myślenia, prowadząc do heurystyki informatycznej jako metody rozwiązywania problemów wspomaganej komputerem, tworzą podwaliny sztucznej inteligencji. **Z. Martyniak** podkreślając, że heurystyka jest młodą nauką dodaje, iż nazywa się ją również inwentyką i innowatyką, chociaż następnie tymi terminami nazywa dwa spośród trzech nurtów.

Heurystyka wg **Z. Martyniaka** zajmuje się procesami twórczego myślenia oraz wykorzystaniem komputerów. ***Inwentyka*** zajmuje się opracowaniem, kodyfikacją i strategią zastosowania metod twórczego rozwiązywania problemów. ***Innowatyka*** natomiast zajmuje się wdrażaniem „produktów" twórczego myślenia. Podział heurystyki zaproponowany przez Z. Martyniaka nie obejmuje wszystkich zagadnień składających się na tę naukę, także podział na nurty jest ostry, a więc poszczególne nurty zachodzą na siebie, przy czym nie pominął on zagadnienia kształcenia i rozwijania twórczego rozwiązywania problemów.

Heurystyka rozumiana jako ogół działań związanych z twórczym rozwiązywaniem problemów ma pięć działów:

1. **Podstawy heurystyki** - obejmują badanie i rozwijanie procesów twórczego myślenia ludzi, mechanizmy powstawania barier ograniczających te procesy oraz wskazywanie sposobów ich pokonywania.

2. **Heurystyka informatyczna** - obejmuje twórcze rozwiązywanie problemów wsparte komputerem i ma wiele poddziałów, do których należy zaliczyć:
    - budowę i organizację programów heurystycznych, tj. metody poszukiwania rozwiązań w przestrzeni rozwiązywania zadania;
    - dowodzenia twierdzeń w systemach formalnych, planujących działania, podejmujących decyzję;
    - tworzenie struktury bazy danych w pamięci maszyny, tj. określanie i wyszukiwanie informacji niezbędnej do rozwiązania zadania zgodnie z jego klasą, dzielenie i sortowanie informacji zgodnie z jej strukturą lub procedurą;
    - tworzenie uniwersalnych systemów rozwiązywania zadań danej klasy i różnych klas;
    - skomplikowane języki programowania, umożliwiające programowanie automatyczne i interpretujące, a także zdolne do rozwiązywania problemów jedynie na podstawie zbioru informacji, które wystarczą do określenia sposobów rozwiązywania zadań danej klasy;
    - modelowanie wnioskowania: dedukcyjne, indukcyjne, statystyczne, przez analogię oraz modelowanie wpływu różnych czynników psychicznych;
    - tworzenie i weryfikowanie modeli informatycznych dla dowolnych problemów wyrażonych w języku naturalnym;
    - gry komputerowe i kierownicze;
    - uczenie się, uogólnianie i klasyfikowanie;
    - rozpoznawanie postaci (wrażeń odbieranych wzrokiem, słuchem, a także cech obiektu lub sytuacji technologicznej);
    - programowanie robotów;
    - badania psychologiczne, lingwistyczne, filozoficzne, medyczne na potrzeby tworzenia programów heurystycznych analizujących i diagnozujących;
    - współpracę ludzi i komputerów.
3. **Inwentyka** - obejmuje opracowywanie metod i technik heurystycznych oraz ich modyfikacje do problemów i do różnych wariantów organizacyjnych stosowania metod rozwiązywania problemów (np. dostosowanie do zespołów zadaniowych), a także kodyfikację metod.
4. **Innowatyka** - obejmuje: przygotowanie organizacji (przedsiębiorstwa, firmy, jednostki administracyjnej, itp.) do wdrożenia metody, jej wprowadzenie do praktyki i sposoby utrzymywania nowo wprowadzonego rozwiązania; badanie oporów we wdrażaniu i przygotowanie sposobów ich przezwyciężenia; analizowanie zachowań i postaw w okresie przygotowania, wdrażania i utrzymania metody po wdrożeniu; badanie i określanie roli i zadań kadry kierowniczej we wdrażaniu i utrzymaniu metody po wdrożeniu; badanie i projektowanie rozwiązań organizacyjnych z zakresu sterowania i nadzorowania programów, przedsięwzięć oraz pracy zespołów zadaniowych; niezbędne modyfikacje metody jako konsekwencja współpracy z doradcą organizacyjnym; badanie trudności oraz symulację przy współpracy z doradcą.
5. **Dydaktyka heurystyczna** - obejmuje zagadnienia kształcenia i doskonalenia kadry kierowniczej i specjalistycznej w użytkowaniu metod twórczego rozwiązywania problemów; przygotowywanie kursów, seminariów itp.; dobieranie właściwych form szkolenia i doskonalenia dla skutecznego nauczania metod twórczego rozwiązywania problemów; poszukiwanie i badanie nowych form szkolenia, a szczególnie uwzględniających zdobywanie wiedzy różnymi metodami (naukową oraz intuicyjnie), aby

potem tak uzyskaną wiedzę stopić w całość w świadomości jednostki, przy czym scalenie to musi wynikać z przemyślanego i zaplanowanego toku szkolenia.

Po tym wstępie chciałbym przedstawić najczęściej wykorzystywane metody (algorytmy) heurystyczne stosowane w grach logicznych.

Algorytm minimaksowy

Grę dwuosobową, skończoną, o sumie zerowej można sprowadzić do macierzy **A** o liczbie wierszy równej liczbie strategii gracza oraz tylu kolumnach, ile jest strategii przeciwnika. Elementami **a(ij)** macierzy gry **A** są wartości wypłaty gracza **E(.)** (pierwsza współrzędna wektora wpłat).Wybór przez gracza **i-tej**, a przez przeciwnika **j-tej** strategii wyznacza wypłatę odpowiadającą elementowi **a(ij).** Rozważamy gry z pełną informacją i ze strategiami w równowadze. W macierzy gry oznacza to istnienie elementu równocześnie największego w kolumnie i najmniejszego w wierszu (punkt siodłowy).

Wyznaczając ogólną strategię gry z punktu widzenia gracza musimy założyć, że przeciwnik będzie wykonywał najlepsze z możliwych posunięć. Z tym założeniem wiążą się pojęcia „dolnej wygranej" oraz „górnej przegranej". „Dolną wygraną" **vG** wyznacza wzór:

$$\mathbf{vG = \max_{i} \min_{j} a(ij),}$$

co oznacza, że gracz w pierwszej kolejności poszukuje minimalnej wartości przegranej w strategiach przeciwnika (kolumny macierzy **A**), a następnie wartość tę maksymalizuje ustalając swoją strategię. „Górna przegrana" **Vp** określana jest jako wynik analogicznego rozumowania przeprowadzonego z punktu widzenia przeciwnika i wyraża się wzorem:

$$\mathbf{Vp = \max_{i} \min_{j} aij,}$$

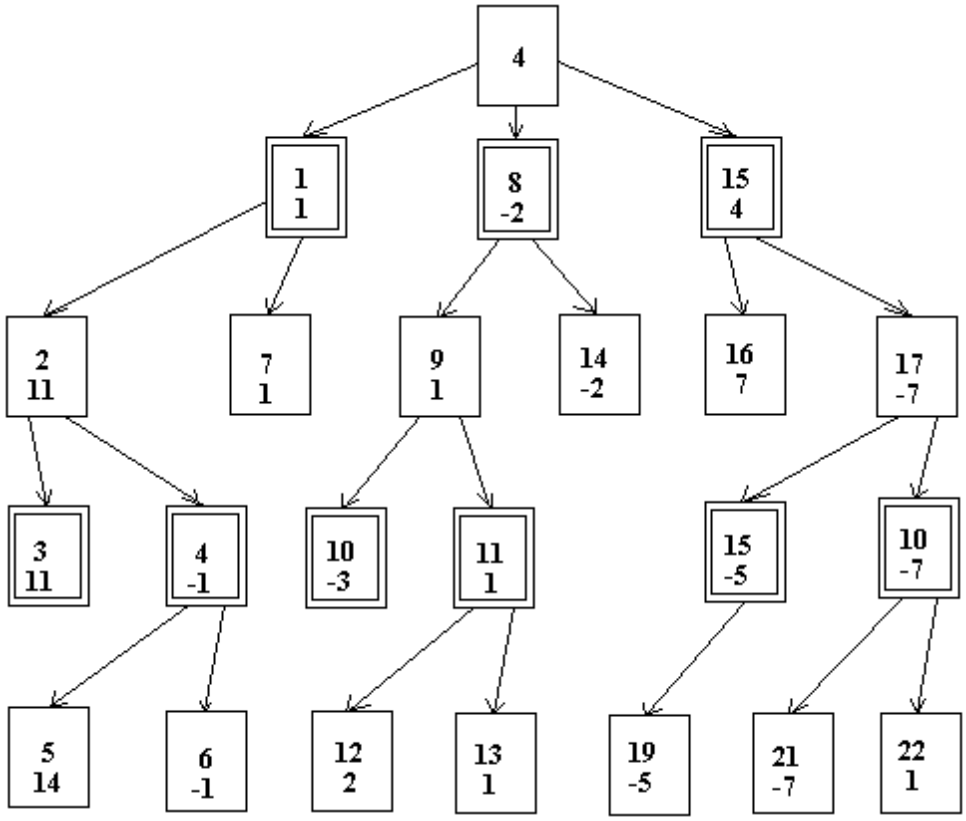
Kolejnym z ważniejszych w teorii gier jest **tw. o minimaksie** (prawdziwe przy założeniach znaczenie ogólniejszych od przyjętych w mniejszej części). Tezą tw. o minimaksie jest równość: **Vg = Vp.**

Dowód twierdzenia przeprowadzonego na wiele sposobów, a przy naszych dodatkowych założeniach staje się elementarny (istnienie punktów siłowych). Tw. o minimaksie gwarantuje dla każdej dwuosobowej gry o sumie zerowej istnienie strategii optymalnej . Dowody tw. mają charakter egzystencjonalny i nie rozstrzygają problemu jej wyznaczenia. Jednak przy przyjętych założeniach, wyznaczenie strategii optymalnej sprowadza się do skonstruowania planu rozgrywki prowadzonej do punktu siodłowego. Przy wyznaczaniu strategii reprezentacja macierzowa jest nieprzydatne. Gry w macierzy reprezentowane są tylko przez stany końcowe, co uniemożliwia skonstruowanie pełnego drzewa gry.

Celem niniejszego paragrafu jest przedstawionych dwóch wersji funkcji wartościującej wierzchołki drzewa gry. Ze względu na inne zasady wartościowania wierzchołków na poziomach gracza i przeciwnika wprowadzono funkcję **POZIOMU (W)** kreśloną następująco:

$$\mathbf{POZIOMU\ (W) = \begin{cases} G\text{ węzeł na poziomie gracza} \\ P\text{ węzeł na poziomie przeciwnika} \\ L\text{ węzeł jest liściem.} \end{cases}}$$

Rysunek 4. Kolejność analizy węzłów w algorytmie minimaksowania; pojedyncza ramka - węzły na poziomie gracza, podwójna - na poziomie przeciwnika, górne liczby oznaczają numer operacji analizy węzła, dolne - wartości przyporządkowane węzłom przez algorytm.



Nie zawsze możliwe jest przechowywanie w pamięci informacji o wszystkich wygenerowanych potomkach. Użyteczna jest wtedy funkcja wartościująca działająca podobnie jak strategia z powracaniem.

W [1] można obejrzeć dokładnie zapis takiego algorytmu w postaci funkcji RANGA_MINIMAX_B(w). Obie przedstawione funkcje wygenerują cale drzewo gry w celu nadania wartości korzeniowi, węzłowi „w" nie zostanie nadana wartość dopóty, dopóki nie zostaną nadane etykiety wszystkim jego potomkom. Algorytm minimaksowy ma więc znaczenie głównie teoretyczne, a praktyczne zastosowanie może znaleźć dla małych drzew gry.

Algorytm „rozwiązywania"

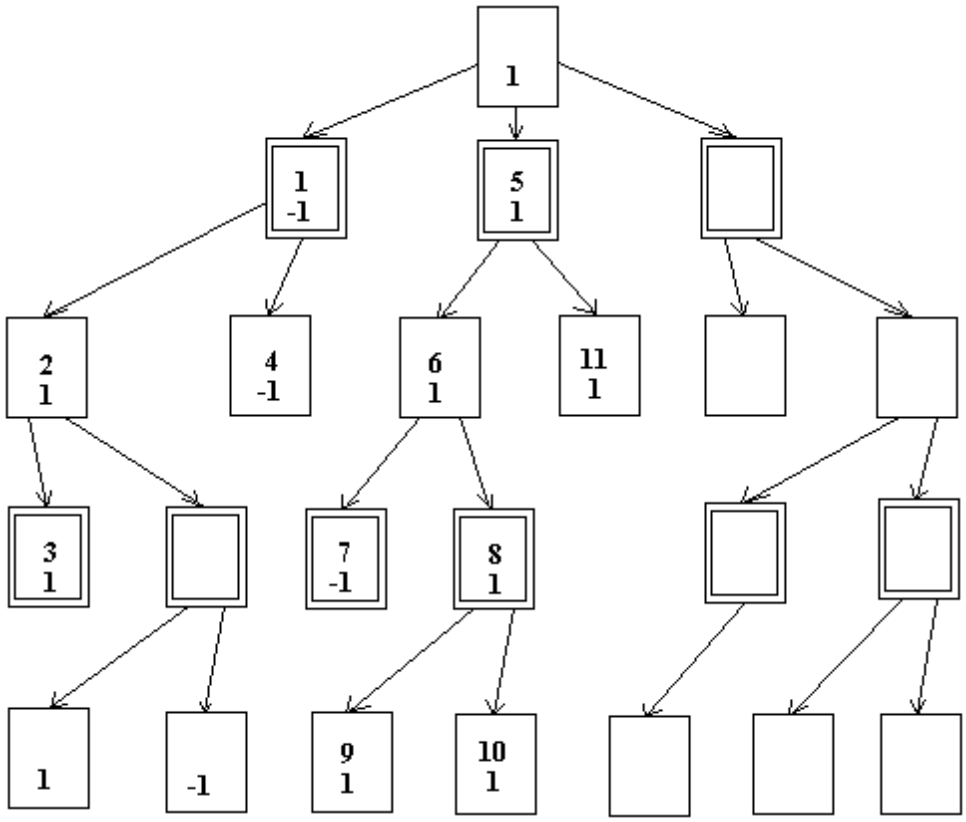
Prezentowany algorytm „rozwiązywania" (ang. slove) jest najprostszą metodą wyznaczenia drzewa rozwiązania bez badania wszystkich wierzchołków. Zastosowanie algorytmu jest jednak ograniczone przez dodatkowe założenia. Dopuszczalne są jedynie dwa stany liścia (a w rezultacie również przodków liści) **WYGRANY** albo **PRZEGRANY**. Inaczej mówiąc funkcja wartościująca jest określona następująco:

$$e(w) = \begin{cases} 1 & \text{STAN}(w) = \text{WYGRANY}, \\ -1 & \text{STAN}(w) = \text{PRZEGRANY}. \end{cases}$$

Takie uproszczenie analizowanych gier pozwala na ograniczenie poszukiwań stanów. Na poziomie gracza węzłowi można przypisać wartość **f(w) = 1** w chwili osiągnięcia pierwszego bezpośredniego potomka **„w”** spełniającego **f(w) = 1** (bez analizowania pozostałych potomków). Na poziomie przeciwnika węzłowi **„w”** przyporządkowuje się wartość f**(w) = -1**, gdy znajdzie się potomek **„w”** spełniający **f(w) = -1** (również nie trzeba analizować pozostałych potomków **„w”**).

Algorytm **„rozwiązywania”** według przestawionych zasad ogranicza liczbę badanych węzłów. Prezentowana funkcja RANGA_SLOVE(w) wyznacza funkcję wartościującą tego algorytmu (poniżej rysunku):

```
function RANGA_SLOVE(w : sprawdzany węzeł) : wartość węzła;
begin
 if POZIOM(w) = L then
   RANGA_SLOVE := E(w);
   { dla liści funkcja przyjmuje wartość funkcji E(.) }
 else begin
   z := GENERACJA(w);
   {funkcja GENERACJA(w) wyznacza potomka węzła "w" wzdłuż nieoznaczonej krawędzi }
   OZNACZENIE(w, z);
   R_S := RANGA_SLOVE(z);
        { wyznaczana jest "bieżąca wartość" wierzchołka "w" R_S, która następnie będzie
        modyfikowana }
   if POZIOM(w) = G then
    while(R_S <> 1) and (not TEST_KRAWEDZI(w)) do begin
        { zakończenie instrukcji while następuje po nadaniu wierzchołkowi "w" wartości 1 lub
        oznaczeniu jako wykorzystanych wszystkich krawędzi wywodzących się z węzła "w" }
        z := GENERACJA(w);
        OZNACZENIE(w, z);
        R_S := RANGA_SLOVE(z);
    end;
    if POZIOM(w) = P then begin
        while (R_S <> -1) and (not TEST_KRAWEDZI(w)) do begin
        { zakończenie instrukcji while następuje po nadaniu wierzchołkowi "w" wartości -1 lub
        oznaczeniu jako wykorzystanych wszystkich krawędzi wywodzących się z węzła "w" }
          z := GENERACJA(w);
          OZNACZENIE(w, z);
        end;
    RANGA_SLOVE := R_S;
    { wartość funkcji nadawana jest w przypadku spełnienia jednego z kryteriów zakończenia lub
        po sprawdzeniu wszystkich potomków węzła "w" }
 end;
end;
```



Rysunek 5. Kolejność analizy węzłów w algorytmie „rozwiązywania"; pojedyncza ramka oznacza węzły na poziomie gracza, podwójna - węzły na poziomie przeciwnika, górna liczba oznacza numer operacji analizy węzła, dolne - wartości przyporządkowane przez algorytm.

Działanie procedury RANGA_SLOVE(w) zilustrowane jest na rysunku 2. Przedstawiona jest kolejność badania węzłów oraz ich wartościowanie (-1 lub 1).

Algorytm „rozwiązywania" jest odpowiednikiem strategii z powracaniem. Liczba obliczeń funkcji wartościującej zależy w sposób bezpośredni od kolejności generowania następników. W praktyce algorytm „rozwiązywania" jest najczęściej strategią ślepą. Porządek sprawdzania węzłów, wyznaczony przez funkcję GENERACJA(w), zależy od przyjętych a priori rozwiązań algorytmicznych, a nie od aktualnej oceny drzewa gry.

Algorytm cięć a-b

Algorytm cięć a-b (gdzie **a** - **alfa**, **b** - **beta**)jest również modyfikacją algorytmu minimaksowego (ze schematem **strategii z powracaniem**). Dotyczy jednak gier ogólniejszych (spełniających założenia z części „Schemat i drzewo gry") z dowolnymi funkcjami wartościującymi. W algorytmie cięć alfa i beta węzły nie wpływające na wartość przypisaną ich przodkom są eliminowane z dalszej analizy. Zaniechanie badań węzła **„w"** następuje wtedy, gdy przyporządkowana mu wartość przekroczy pewne granice liczbowe. Takie cięcie pozwala na wyeliminowanie z dalszej analizy również wszystkich potomków węzła **„w"**. Liczbowe granice cięć są ustalane dynamicznie w sposób następujący:

- **alfa** - granica cięć dla wierzchołka **„w"** na poziomie przeciwnika minus największa aktualna wartość wszystkich przodków **„w"** na poziomie gracza; zakończenie analizy węzła **„w"** nastąpi w chwili, gdy aktualna wartość **„w"** stanie się mniejsza lub równa **alfa**;

- **beta** - granica cięć dla wierzchołka **„w”** na poziomie przeciwnika minus najmniejsza aktualnie przypisana wartość wszystkich przodków **„w”** na poziomie przeciwnika; zakończenie analizy węzła **„w”** nastąpi w chwili, gdy aktualna wartość przypisana **„w”** stanie się większa lub równa **beta**.

Z definicji granic **cięć alfa-beta** wynika, że w czasie wykonania algorytmu liczby **alfa** i **beta** są poprawiane (**alfa** zwiększane, a **beta** zmniejszane).

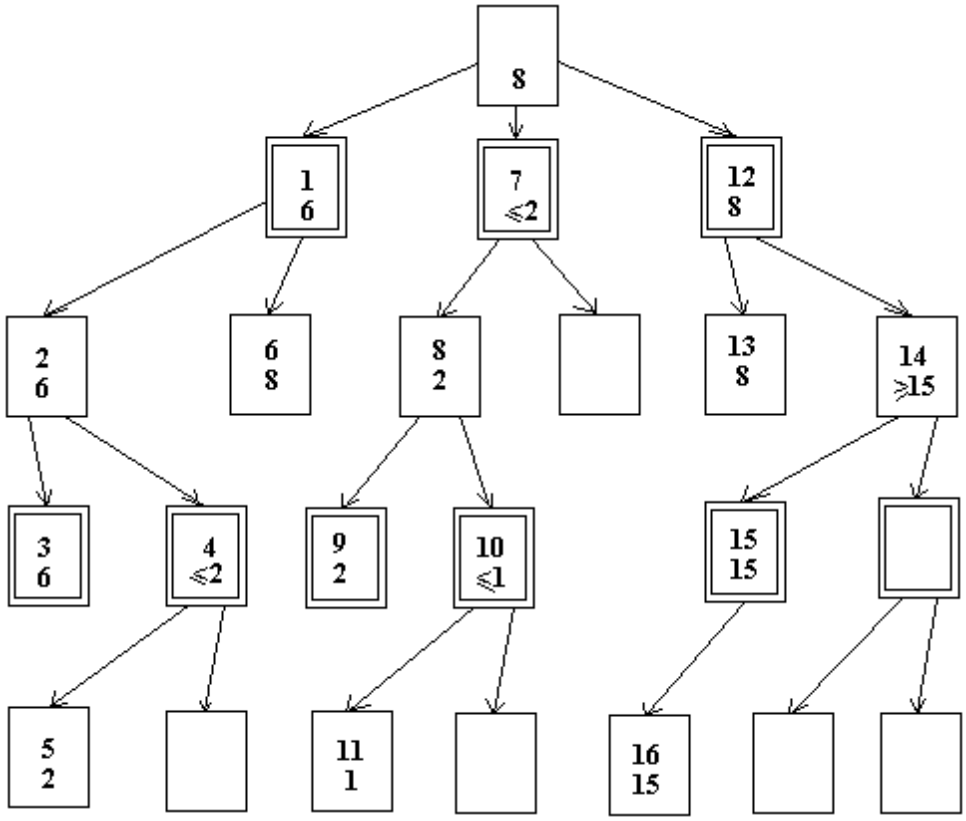
Rekurencyjna funkcja RANGA_a_b(w, a, b) jest realizacją funkcji wartościującej **algorytmu cięć alfa-beta**:

```
function RANGA_a_b( w: sprawdzany węzeł;
                    a: granica ciec na poziomie przeciwnika;
                    b: granica ciec na poziomie gracza) : wartość węzła;
begin
 if POZIOM(w) = L then
   RANGA_a_b := E(w);
   { dla liści funkcja przyjmuje wartość funkcji E(.) }
 else begin
   z := GENERACJA(w);
   { funkcja GENERACJA(w) wyznacza potomka węzła "w" wzdłuż nieoznaczonej krawędzi }
   OZNACZENIE(w, z);
   if POZIOM(w) = G then begin
     a := max(a_b (z, a, b));
     while(a >= b) and (not TEST_KRAWEDZI(w)) do begin
          { zakończenie instrukcji while następuje po wyznaczeniu wierzchołka spełniającego
nierówność      a>=b lub oznaczeniu jako wykorzystanych wszystkich krawędzi wywodzących się z
węzła "w" }
          z := GENERACJA(w);
          OZNACZENIE(w, z);
          a := max( a, RANGA_a_b(z, a, b));
     end;
     if a >= b then RRRANGA_a_b := b
     else RANGA_a_b := a;
   end;
   if POZIOM(w) = P then begin
     while (a < b) and (not TEST_KRAWEDZI(w)) do begin
          { zakończenie instrukcji while następuje po wyznaczeniu wierzchołka spełniającego
nierówność      a >= b lub oznaczeniu jako wykorzystanych wszystkich krawędzi wywodzących się z
węzła "w"       }
          z := GENERACJA(w);
          OZNACZENIE(w, z);
     end;
     b := min(b, RANGA_a_b(z, a, b))
   end;
   if a >= b then RANGA_a_b := a
   else RANGA_a_b := b;
 end;
end;
```

Działanie procedury RANGA_a_b(w, a, b) ilustruje rysunek 3. Zaprezentowana jest kolejność badania węzłów.

Parametry fikcji RANGA_a_b(w, a, b) spełniają nierówność **alfa < beta**. Dla węzła początkowego **„p”** funkcję można wywołać następująco:

e(p) := RANGA_a_b(p, -nieskończoność, +nieskończoność).



Rysunek 6. Kolejność analizy węzłów w **algorytmie cięć alfa-beta**; pojedyncza ramka oznacza węzły na poziomie gracza, podwójna - węzły na poziomie przeciwnika, górne liczby oznaczają numer operacji analizy węzła, dolne - wartości przyporządkowane węzłom przez algorytm.

W wielu pracach prezentowane są analizy zasad odcinania węzłów w **algorytmie cięć alfa-beta**. Okazuje się, że pomijane są jedynie te węzły, które nie maja wpływu na wartościowanie węzła początkowego (korzenia drzewa gry) **„p”.**

Niech $F_w(x)$ będzie funkcją wpływu węzła na korzeń **„p”.** Wartość $F_w(x)$ można obliczać stopniowo przy założeniu, że jest ona przenoszona wzdłuż drogi łączącej węzły **„p”** oraz **„w”.** Przyjmijmy, że funkcja $f_{v,w}(x)$ określa bezpośredni wpływ wierzchołka na jego przodka **„v”:**

$$RANGA(v = f_{v,w}(RANGA(w)) .$$

Prawdziwa jest zależność: $$F_w(x) = F_v(f_{v,w}(x)),$$

a dla drogi łączącej węzły **„p”** oraz **„w”** (droga jest ciągiem węzłów $w_1, w_2, ..., w_n$, w którym $w_1 = p, w_n = w$)

$$F_w(x) = f_p(f_{p,w2}(... (f_{wn-1,w}(x)) ...))).$$

Jeżeli **„v”** jest wierzchołkiem na poziomie gracza, to funkcja wpływu jest określona następująco:

KRZYSZTOF SIUZDAK **WROCŁAW 18.01.99r.**
IV rok IO
specjalność : bazy danych
nr indeksu : 76855

$$f_{v,w} = f^{+}_{\alpha} = \max(\alpha, x)$$

natomiast dla węzła **„v”** na poziomie przeciwnika spełniona jest równość:

$$f_{v,w} = f^{-}_{\beta} = \min(\beta, x)$$

Dla **alfa < beta** (warunek spełniony w **algorytmie cięć alfa-beta**) funkcje f^{+}_{α} oraz f^{-}_{β} są przemienne. Natomiast założenia funkcji f^{+}_{α} oraz f^{-}_{β} są funkcjami tego samego typu:

$$f^{+}_{\alpha_1}(f^{+}_{\alpha_3}(x) = f^{+}_{\alpha_3}(x), \text{ gdzie } \alpha_3 = \min(\alpha_1, \alpha_2),$$

$$f^{-}_{\beta_1}(f^{-}_{\beta_3}(x) = f^{-}_{\beta_3}(x), \text{ gdzie } \beta_3 = \min(\beta_1, \beta_2).$$

Wykorzystując te spostrzeżenia można funkcję $\mathbf{F_w(w)}$ zapisać w następujący sposób:

$$F_w(x) = f_{B(w)}(f_{A(w)}(x)), \quad A(w) = \max_j(\alpha_j), \quad B(w) = \min_i(\beta_i),$$

gdzie α_j, β_i związane są z węzłami odpowiednio w_j oraz w_i.

Jeżeli możemy analizować pełne drzewo gry, to liczby **A(w), B(w)** można opisać inaczej, o ile znany jest porządek generowania węzłów przez **algorytm cięć alfa-beta**. Dla danego węzła **„w”** liczba **A(w)** jest największą wartością, przypisaną wg zasady minimaksowej, spośród wszystkich wygenerowanych przed **„w”** potomków przodka węzła **„w”** na poziomie gracza. Analogicznie liczba **B(w)** jest najmniejszą wartością, przypisana wg zasady minimaksowej, spośród wszystkich wygenerowanych przed **„w”** potomków przodka węzła **„w”** na poziomie przeciwnika.

W zależności od liczb **A(w), B(w)** można sformułować warunek konieczny i dostateczny wygenerowania węzła **„w”.** Węzeł **„w”** będzie wygenerowany przez **algorytm cięć alfa-beta** wtedy i tylko wtedy, gdy spełniona jest nierówność: **A(w) < B(w).**

Notacje algorytmów wyznaczania strategii gier

Przyjęta w tej części zasada opisu strategii gry z punktu widzenia jednego gracza jest użyteczna przy uzasadnieniu i opisie algorytmów przeszukiwania. Notacja tego rodzaju nosi nazwę **MIN-MAX**. Węzły na poziomie gracza są nazywane **MAX**, a na poziomie przeciwnika **MIN**. Stosowany jest również opis strategii gry z punktu widzenia gracza wykonującego właśnie posunięcie i taka postać nosi nazwę **NEG-MAX**. Opis strategii w postaci **NEG-MAX** jest bardziej jednolity. Przedstawiając takie zapisy ogólnej funkcji wartościującej oraz funkcji wartościującej **algorytm cięć alfa-beta** utrzymujemy w mocy wszystkie wprowadzone założenia. W szczególności przyjmujemy, najlepsze posunięcie zapewnia największą wartość funkcji na końcu rozrywki, przy najlepszych możliwych posunięciach przeciwnika. Jeżeli **e(w)** jest najlepszą (z punktu widzenia gracza wykonującego posunięcie) wartością osiągalną ze stanu gry reprezentowanego przez węzeł **„w”,** to wartości dla tego gracza po wykonaniu ruchu do **„w”** wynoszą **-e(w):**

e(w) = <
| E(w), gdy POZIOM(w) = L
| max(-e(w_i)), gdy w_i są potomkami węzła „w”.

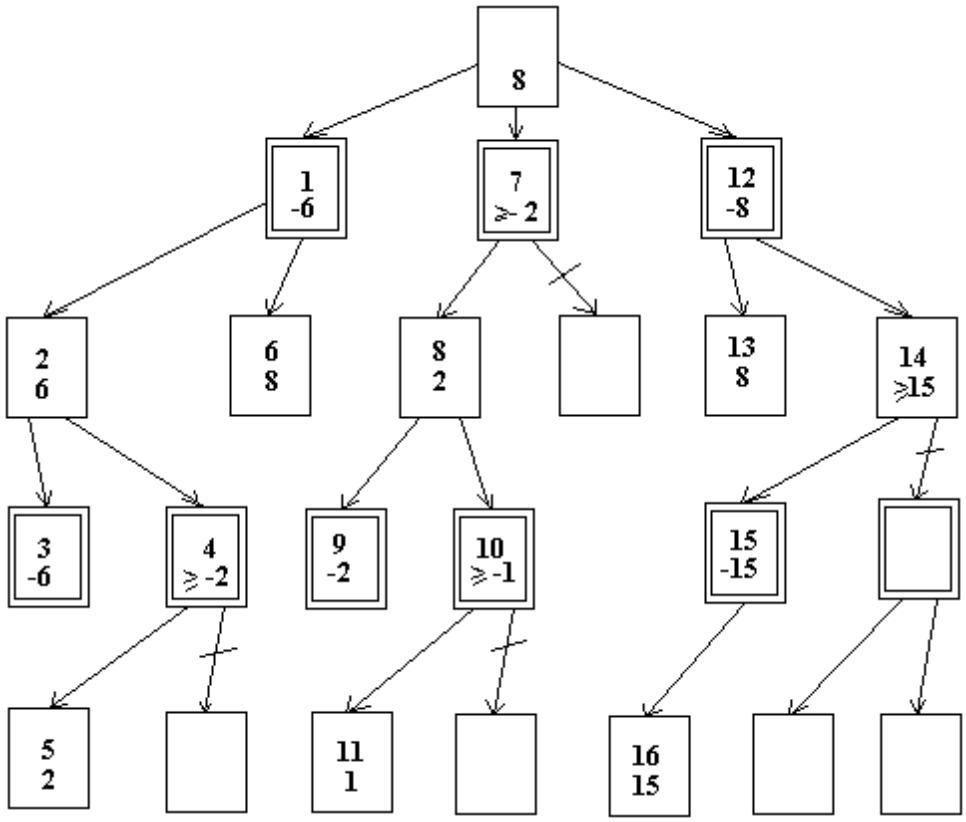
KRZYSZTOF SIUZDAK **WROCŁAW 18.01.99r.**
IV rok IO
specjalność : bazy danych
nr indeksu : 76855

Zapis ogólnej funkcji rekurencyjnej wartościującej R_NEGMAX(w) jest następujący:

```
function RNEGMAX( w: sprawdzany węzeł ) : wartość węzła;
begin
  if POZIOM(w) = L then
    R_NEGMAX := E(w);
    { dla liści funkcja przyjmuje wartość funkcji E(.) }
  else begin
    z := GENERACJA(w);
    { funkcja GENERACJA(w) wyznacza potomka węzła "w" wzdłuż nieoznaczonej krawędzi }
    R_NEGMAX := max(-R_NEGMAX(z))      dla z należących do Z;
    { wartość funkcji dla węzłów, które nie są liśćmi }
  end;
end;
```

Podobnie upraszcza się opis algorytmu cięć alfa-beta, przedstawiony jako funkcja R_NEGMAX_a_b(w, a, b):

```
function RNEGMAX_a_b( w: sprawdzany węzeł;
                      a: granica cięć na poziomie przeciwnika;
                      b: granica cięć na poziomie gracza) : wartość węzła;
begin
  if POZIOM(w) = L then
    RNEGMAX_a_b := E(w);
    { dla liści funkcja przyjmuje wartość funkcji E(.) }
  else begin
    z := GENERACJA(w);
    { funkcja GENERACJA(w) wyznacza potomka węzła "w" wzdłuż nieoznaczonej krawędzi }
    OZNACZENIE(w, z);
    a := max(a, RANGA_a_b(z, a, b));
    while(a < b) and (not TEST_KRAWEDZI(w)) do begin
      { zakończenie instrukcji while następuje po wyznaczeniu wierzchołka spełniającego nierówność
          a>=b lub oznaczeniu jako wykorzystanych wszystkich krawędzi wywodzących się z węzła
"w" }
      z := GENERACJA(w);
      OZNACZENIE(w, z);
      a := max( a, -RANGA_a_b(z, -b, -a));
    end;
    if a >= b then RRRANGA_a_b := a
    else RANGA_a_b := b;
  end;
end;
```



Rysunek 7. ilustruje działanie strategii cięć alfa-beta w konwencji zapisu NEG_MAX. Kolejność analizy węzłów w algorytmie cięć alfa-beta (w notacji NEG_MAX); pojedyncza ramka oznacza węzły na poziomie gracza, podwójna - węzły na poziomie przeciwnika, górne liczby oznaczają numer operacji analizy węzła, dolne - wartości przyporządkowane węzłom przez algorytm.

Zastosowanie notacji **NEG_MAX** znacznie upraszcza opis algorytmów dla gier, jednak ocena rozgrywki z punktu widzenia gracza wydaje się łatwiejsza do analizowania. Z tego względu w wielu pracach stosowana jest notacja **NEG_MAX** i w dalszym ciągu także będzie wykorzystywana.

Algorytm „rozpoznawania”

Algorytm **„rozpoznawania”** (ang. **scout**) został zaproponowany przez **J. Pearla** [ ]. Początkowo był wykorzystywany do teoretycznej analizy drzew gier. Okazało się jednak, że może mieć również zastosowanie w praktycznej konstrukcji drzew rozwiązania.

Główną przesłanką algorytmu **„rozpoznawania”** jest obserwacja, że duża część obliczeń, związanych z wartościowaniem węzłów, poświęcona jest testowaniu nierówności. We wszystkich przedstawionych algorytmach gier decydujące znaczenie mają operacje wyznaczania wartości maksymalnych i minimalnych w zbiorach liczb przyporządkowanych węzłom. Realizowano to poprzez obliczanie wartości węzłów, a następnie ich uporządkowanie.

W algorytmie **„rozpoznawania”** wykonywane są testy, których celem jest wyeliminowanie węzłów nie wpływających na aktualnie obliczane wartości minimalne lub maksymalne. Efektywność algorytmu jest tym większa, im mniejszy jest koszt testowania

węzłów w stosunku do kosztu przypisywania im wartości. Celem testów jest rozstrzygnięcie czy węzeł powinien być dokładnie analizowany, czy też nie. Funkcja testowania może mieć więc schemat podobny do algorytmu **„rozwiązywania”,** ponieważ węzłowi przyporządkowywane są tylko dwie wartości.

Zasadnicza konstrukcja algorytmu **„rozwiązywania”** przypomina minimaksowanie. Wyeliminowanie z obliczeń pewnych węzłów jest wynikiem zastosowania funkcji testowych. Algorytm jest przedstawiony jako rekurencyjna funkcja RANGA_SCOUT(w):

```
function RANGA_SCOUT(w : sprawdzany węzeł) : wartość węzła;
begin
 if POZIOM(w) = L then
  RANGA_SCOUT := E(w);
  { dla liści funkcja przyjmuje wartość funkcji E(.) }
 else begin
  z := GENERACJA(w);
  {funkcja GENERACJA(w) wyznacza potomka węzła "w" wzdłuż nieoznaczonej krawędzi }
  OZNACZENIE(w, z);
  R_SC := RANGA_SCOUT(z);
  { wyznaczana jest "bieżąca wartość" wierzchołka "w" R_SC, która następnie będzie
       modyfikowana }
  if POZIOM(w) = G then
   while (not TEST_KRAWEDZI(w)) do begin
       { zakończenie instrukcji while następuje oznaczeniu jako wykorzystanych wszystkich
       krawędzi wywodzących się z węzła "w" }
       z := GENERACJA(w);
       OZNACZENIE(w, z);
       if TEST_SCOUT(z, R_SC, >) then R_SC := RANGA_SCOUT(z);
       { zapis funkcji TEST_SCOUT(z, R_SC, >) przedstawiony jest poniżej }
   end;
   if POZIOM(w) = P then begin
       while (not TEST_KRAWEDZI(w)) do begin
        { zakończenie instrukcji while następuje po oznaczeniu jako wykorzystanych  wszystkich
             krawędzi wywodzących się z węzła "w" }
        z := GENERACJA(w);
        OZNACZENIE(w, z);
        if TEST_SCOUT(z, R_SC, >) then R_SC := RANGA_SCOUT(z)
        { zapis funkcji TEST_SCOUT(z, R_SC, >) przedstawiony jest poniżej }
       end;
       RANGA_SCOUT := R_SC;
   { wartość funkcji nadawana jest po sprawdzeniu lub przetestowaniu wszystkich potomków  węzła
       "w" }
 end;
end;
```

Strategia **„rozpoznawania”** testuje - przed sprawdzeniem węzłów - ich jakość z punktu widzenia wartościowania. Wykorzystywana jest funkcja TEST_SCOUT(z, m, O) testująca relację RANGA_SCOUT(z)’m:

```
function TEST_SCOUT(w : sprawdzany węzeł;
                    m: testowana wartość;
                    o: testowana relacja) : wartość węzła;
begin
```

```
 if POZIOM(w) = L then begin
  if E(w) o m then TEST_SCOUT := true
  else TEST_SCOUT := false;
 end
 else begin
  z := GENERACJA(w);
  { funkcja GENERACJA(w) wyznacza potomka węzła "w" wzdłuż nieoznaczonej krawędzi }
  T_S := TEST_SCOUT(z, m, o);
  if POZIOM(w) = G then
   while (not TEST_KRAWEDZI(w)) do begin
        { zakończenie instrukcji while następuje oznaczeniu jako wykorzystanych wszystkich
krawędzi        wywodzących się z węzła "w" }
        z := GENERACJA(w);
        OZNACZENIE(w, z);
        T_S := TEST_SCOUT(z, m, o);
   end;
   if POZIOM(w) = P then begin
        while T_S and (not TEST_KRAWEDZI(w)) do begin
         { zakończenie instrukcji while następuje po oznaczeniu jako wykorzystanych wszystkich
              krawędzi wywodzących się z węzła "w" }
         z := GENERACJA(w);
         OZNACZENIE(w, z);
         T_S :=  TEST_SCOUT(z, m, o)
        end;
        if T_S then TEST_SCOUT := true
        else TEST_SCOUT := false;
 end;
end;
```

Teoretyczna analiza algorytmu **„rozpoznawania”** jest stosunkowo łatwa (takie było uzasadnienie jego skonstruowania). Jednak porównywanie go z innymi algorytmami jest już trudne. Ogólnie , w stosunku do **algorytmu cięć alfa-beta** efektywność algorytmu **„rozpoznawania”** zwiększa się gdy rośnie głębokość drzew badanych gier, nawet pomimo dwukrotnego sprawdzania niektórych węzłów (testowania i przypisywania wartości). Szczegółowa analiza nie prowadzi jednak do tak prostych i jednoznacznych wniosków. Rozważania teoretyczne dotyczące zostały przedstawione w pracach [ ], [ ].

Heurystyczny algorytm SSS*

Algorytm SSS* [4] poszukuje optymalnego drzewa rozwiązania. Wyznaczone drzewo jest reprezentacją optymalnej strategii gracza. W porównaniu z poprzednio prezentowanymi algorytmami gier, **SSS*** przypomina metody heurystycznego przeszukiwania. O kolejności badania węzłów nie decyduje ustalony a priori porządek ich generowania. Do analizy wybierane są węzły, które gwarantują zmaksymalizowanie wartości przypisywanej węzłowi początkowemu **„p”.** W tym sensie algorytm **SSS*** przypomina strategię **AO*.**

W algorytmie **SSS*** węzeł **„w”** drzewa gry rozważany jest z punktu widzenia zbioru poddrzew rozwiązania z korzeniem **„w”.** Na podstawie wartości liści takich poddrzew ustala się ich górne ograniczenia. Te górne granice są wykorzystywane do uporządkowania zbioru poddrzew rozwiązań; poddrzewo o największym ograniczeniu analizowane jest w pierwszej kolejności. W każdym poddrzewie drzewa gry wartości przypisywane węzłom przez **algorytm SSS*** tworzą monotoniczny ciąg nierosnący. Zakończenie przeszukiwania następuje w chwili znalezienia pierwszego drzewa rozwiązania. Proces konstrukcji tego drzewa gwarantuje jego zakończenie.

KRZYSZTOF SIUZDAK **WROCŁAW 18.01.99r.**
IV rok IO
specjalność : bazy danych
nr indeksu : 76855

Procedura STRATEGIA_SSS*(p) jest zapisem **algorytmu SSS*.** Nowym elementem, w procedurach dla gier, jest lista węzłów **O** uporządkowana według wartości przypisywanych poszczególnym węzłom (dokładniej poddrzewom). Z każdym węzłem **„w”** związana jest informacja **s(w)** oraz liczba **h(w).** Funkcja **s(w)** jest dwuwartościowa:

$$s(w) = \begin{cases} N & \text{potomkowie „w” nie są wygenerowani,} \\ R & \text{zakończone zostało badanie „w” i jego potomków.} \end{cases}$$

Liczba **h(w)** jest wartością przypisywaną węzłowi **„w”** podczas wykonywania algorytmu.

```
procedure STRATEGIA_SSS* (p: węzeł początkowy);
begin
 ROZSZERZENIE(O, {p});
 s(p) := N;
 h(p) := +nieskończoność;
 while O <> 0 do begin
  w := WYBOR_SSS*(O);
  { funkcja WYBOR_SSS*(O) wyznacza jako "w" węzeł o największej wartości h }
  if (w=p) and (s(w) = R) then begin
   WYJSCIE_Z_DRZEWEM_ROZWIAZANIA;
   { wyznaczone zostało pełne drzewo rozwiązania a wartością wygranej równej h(w) }
   goto K
  end
  else begin
   if s(w) = N then begin
        if POZIOM(w) = G then begin
         z := GENERACJA1_SSS*(w);
         { funkcja RENERACJA_SSS*(w) dokonuje ekspansji węzła "w" }
         for z := wszystkie elementy Z do begin
          s(z) := N;          h(z) h(w);
         end;
         ROZSZERZENIE1_SSS*(O, Z);
         { procedura umieszcza węzły zbioru Z na początku listy O }
        end;
        if POZIOM(w) = L then begin
         s(w) := R;    h(w) := min(E(w), h(w));
         ROZSZERZENIE3_SSS*(O, w);
         { procedura umieszcza węzły w na liście O przed wszystkimi węzłami o mniejszej wartość h;
                w przypadku węzłów o równej wartości h, pierwszym jest wcześniej wygenerowany }
        end;
   end;
   if s(w) = R then begin
        if POZIOM(w) = G then
         if TEST_SSS*(w) then begin
          { funkcja TEST_SSS*(w) przyjmuje wartość true (false w przeciwnym przypadku), gdy
                węzeł "w" ma na liście O brata "w" (węzeł z jednakowymi przodkami) spełniającego
                równość s(w) =N }
          s(w) := N;    h(w) := h(w);
          ROZSZERZENIE2_SSS*(O, w);
          { węzeł "w" umieszczany jest na początku listy O }
         end
         else begin
          z := PRZODEK(w);
```

```
            { funkcja PRZODEK(w) wyznacza bezpośredniego przodka węzła "w" na liście O }
            s(z) := R;    h(z) := h(w);
            ROZSZERZENIE2_SSS*(O, z);
            { węzeł "z" umieszczany jest na początku listy O }
          end;
          if POZIOM(w) = P then begin
            z := PRZODEK(w);
            { z jest bezpośrednim przodkiem węzła "w" }
            s(z) := R;    h(z) := h(w);
            ROZSZERZENIE2_SSS*(O, z);
            { węzeł "z" umieszczany jest na początku listy O }
            UPORZADKOWANIE_SSS*(O);
            { procedura usuwa z listy O wszystkie następniki węzła "z" }
          end;
        end;
    end;
  end;
K:end;
```

W stosunku do poprzednio przedstawionych algorytmów opis **SSS*** jest bardziej złożony. Dla uproszczenia, w opisie procedury użyte zostały nieprecyzyjne (lecz zrozumiałe intuicyjnie) sformułowania „wcześniej wygenerowany”. Bardziej formalny opis możliwy jest przy wykorzystaniu dla reprezentacji węzłów w drzewie gry notacji **Deweya** [4]. Notacja ta wiąże z każdym węzłem informację o porządku jego wygenerowania.

Niewątpliwą zaletą **algorytmu SSS*** jest wyznaczanie jego pierwszego optymalnego drzewa rozwiązania. Jednak skomplikowana struktura **algorytmu SSS*** oraz jego olbrzymie wymagania pamięciowe (nawet w przypadku stosunkowo prostych gier) powodują, że nie jest on jeszcze praktycznie wykorzystywany.

Porównanie algorytmów

Porównania skuteczności strategii przeszukiwania dokonano w Instytucie Informatyki Uniwersytetu Warszawskiego za pomocą symulacji komputerowej. Analizowano wariant gry na planszy 8x8 z liczbami dodatnimi z przedziału <1, 9> i ujemnymi z przedziału <-9, -1>. Przykładowa plansza takiej gry przedstawiona jest na rysunku 8. Widać, że trudno jest ocenić, który z partnerów ma większe szanse wygrania. Gdyby jedna z kolumn zawierała same liczby dodatnie, wtedy gracz miałby pewność wygranej. Gdyby zaś istniał wiersz z liczbami ujemnymi, wtedy przeciwnik miałby pewność wygranej.

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

Rysunek 8.

KRZYSZTOF SIUZDAK **WROCŁAW 18.01.99r.**
IV rok IO
specjalność : bazy danych
nr indeksu : 76855

Przeprowadzono testy dla 100 plansz z losowo generowanymi liczbami. Do generowania liczb wykorzystano standardowy generator (firmy Microsoft), przy czym losowane wartości były dodatnie z założonym prawdopodobieństwem (parametr operacji generowania). Jeżeli to prawdopodobieństwo wynosiło P, to liczby dodatnie losowane były z prawdopodobieństwem 1/9P, a ujemne z prawdopodobieństwem 1/9(1-P).

Analiza dotyczyła liczby wierzchołków końcowych badanych przez trzy algorytmy: **cięć alfa-beta, „rozpoznawania" i SSS*** dla tych samych gier. Plansze generowane były z prawdopodobieństwem P, a wartość P przebiegała od 0,01 do 1 z krokiem 0,01. Dla konkretnej planszy każdy z algorytmów nadawał węzłowi początkowemu tą samą wartość.

W przeprowadzonych testach **algorytm SSS*** był lepszy w 58% przypadków, **algorytm cięć alfa-beta** był lepszy w 37% przypadków, natomiast dla 5% plansz gry osiągnięto wyniki identyczne. Ocenę wynikającą z tego porównania należy jednak zmodyfikować faktami dotyczącymi bardziej złożonej struktury **SSS*** w porównaniu z **algorytmem cięć alfa-beta** oraz większymi wymaganiami pamięciowymi.

Tylko w 5% przypadków **algorytm „rozpoznawania"** charakteryzował się mniejszą liczbą przetestowanych i wartościowanych wierzchołków w porównaniu z liczbą badanych wierzchołków w **algorytmie cięć alfa-beta**. W pozostałych przypadkach w trakcie wykonywania **algorytmu „rozpoznawania"** liście były testowane wielokrotnie. Tak niekorzystny wynik dla **algorytmu „rozpoznawania"** należy jednak zmodyfikować. Wiele opracowań [4] podaje wyniki testów dla innych gier, preferujące **algorytm „rozpoznawania"**.

Przeprowadzone testy potwierdziły praktyczną skuteczność **algorytmu cięć alfa-beta**, która jest wynikiem stosunkowo prostej struktury algorytmu raz małej liczby testowanych wierzchołków. Uzasadnione jest więc powszechne stosowanie **algorytmu cięć alfa-beta** w programach dla gier.

Literatura podstawowa:

1. L. Bolc - „Metody przeszukiwania heurystycznego", tom 1.
2. L. Bolc - „Metody przeszukiwania heurystycznego", tom 2.
3. J. Antoszkiewicz - „Metody heurystyczne. twórcze rozwiązywanie problemów."

Literatura uzupełniająca (wykorzystywana przez autorów powyższych pozycji):

1. T. Ibaraki - „Branch-and-bound procedure and state spase representation of combinatorial optymalization problems.", „Information and Control", 1978, vol.3 nr 41.
2. J. Pearl - "Hueristics: Inteligent search strategis for computer problem solving."
3. I. Pohl - „Bi - directional search", „Machine Inteligence", 1971, nr 6.
4. P. S. Rosenbloom - „A world-championship-level Othello program", „Artificial Inteligence", 1983, vol21, nr 1-2.