Sztuczna inteligencja w brydżu

Piotr BUTRYN[*]

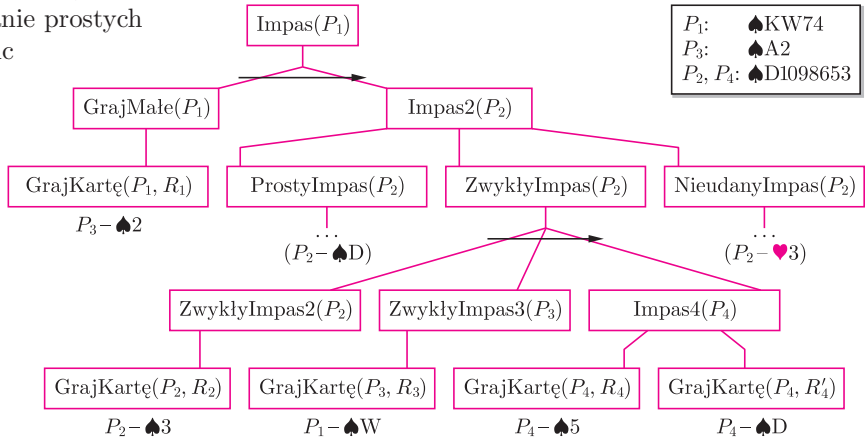
Komputery wyśmienicie grają w warcaby, szachy, backgammona (tryktrak), othello (reversi) – potrafią przeanalizować możliwe dalsze scenariusze gry i zabezpieczyć się przed dobrymi ruchami przeciwnika, dysponują także większą niż człowiek pamięcią. Istnieją jednak gry, np. go, w których jeszcze niedawno komputery nie miały żadnych szans w konfrontacji z najlepszymi zawodnikami, a i z przeciętnymi nie radziły sobie za dobrze.

Z podobną sytuacją mamy do czynienia w przypadku programów grających w brydża. Przeszukanie całego drzewa gry jest niemożliwe ze względu na jego wielkość. W średnim przypadku ma ono około $2 \cdot 10^{24}$ liści, a pesymistycznie nawet $6 \cdot 10^{44}$. Zastosowanie prostych algorytmów, jak alfabeta czy dodanie tablic transpozycji, oczywiście pomaga, ale nie na tyle, aby znaleźć najlepszy ruch w rozsądnym czasie (około minuty). Ograniczenie przeszukiwania do pewnego poziomu jest natomiast trudne do zrealizowania ze względu na brak prostych funkcji, które choć w przybliżeniu mogłyby określić liczbę lew możliwych do wzięcia przez obie strony. Zaczęto więc poszukiwać innych technik mających na celu ograniczenie rozgałęzienia drzewa gry, a co za tym idzie, zmniejszenie liczby możliwych stanów.

Pierwszym przykładem takiego algorytmu jest *planowanie* (ang. *Hierarchical Task Network Planning*, *HTN*), zastosowane w brydżu po raz pierwszy w 1997 roku w programie *Bridge Baron*. Polega ono na podziale głównego problemu na proste zadania, które potrafimy wykonać. W przypadku brydża zadaniem do zaplanowania jest wzięcie określonej liczby lew. Aby to osiągnąć, stosujemy konkretne metody, takie jak ściągnięcie atutów, przebicie karty w dziadku, zaimpasowanie czy zagranie z góry. Z kolei te metody dzielimy na pojedyncze akcje, jakimi są zagrania konkretnych kart. W każdym węźle drzewa gry zamiast zagrania wszystkich możliwych kart sprawdzamy wszystkie metody, jakie możemy w nim zastosować. W zależności od karty ujawnionej przez przeciwnika (a zatem także pewnej nowej informacji o układzie zakrytych kart) wybieramy odpowiednie podmetody mające zastosowanie w danym przypadku. Każda z nich prowadzi nas do innego stanu gry (i innej informacji o układzie kart), w którym to znów próbujemy wszystkich dostępnych metod rozgrywki.

Jedną z najprostszych metod jest impas, czyli zagranie polegające na wzięciu lewy na kartę niższą, dzięki temu, że karta wyższa jest w ręce konkretnego przeciwnika.

Rozważmy kolor, w którym w ręce mamy A2 (as i dwójka), a w dziadku (kartach partnera wyłożonych na stole) KW74. Zagranie dwójki i dołożenie waleta to właśnie impas, który pozwoli na wzięcie lewy, jeśli damę ma przeciwnik zagrywający jako drugi. Należy jednak zauważyć, że czasami karta dołożona jako druga może nas odwieść od dołożenia waleta. Stanie się tak wtedy, gdy przeciwnik dołoży damę lub kartę innego koloru. W pierwszym przypadku dołożymy, oczywiście, króla, w drugim wiemy natomiast, że damę ma gracz zagrywający jako czwarty, i w związku z tym nasz plan także może ulec modyfikacji. Opisaną sytuację przedstawia rysunek 1.



Rys. 1. Część drzewa gry odpowiadająca metodzie „Impas” (parametr metody oznacza zagrywającego gracza). W zależności od karty zagranej przez przeciwnika (gracz P_2) przechodzimy do odpowiedniej podmetody. Najczęściej (tzn. gdy przeciwnik dołoży pika, ale nie damę) będzie to „ZwykłyImpas”, polegający na zagraniu waleta i wzięciu na niego lewy lub nie, w zależności od zagrania (R_4, R'_4) drugiego przeciwnika (P_4).

To, jaką decyzję podejmiemy, zależy od wartości oczekiwanej danych zagrań. Tę wartość obliczamy bardzo prosto – jeśli to my zagrywamy kartę, to wartością stanu gry jest maksimum z wartości poszczególnych metod, natomiast jeśli jest ruch przeciwnika, to jest to średnia ważona wartości synów tego węzła, gdzie wagami są prawdopodobieństwa zaistnienia danych sytuacji.

Załóżmy, że w opisanym przykładzie (A2 do KW74) zagranie asa prowadzi do wyniku +600 (3BA, 9 lew), natomiast zagranie na impas do +630 (3BA, 10 lew), jeżeli impas się uda, lub −100 (3BA, 8 lew) w przeciwnym przypadku. Jeśli drugi gracz dołoży małą kartę (prawdopodobieństwo tego to 0,9854), to impas uda się w połowie przypadków, więc wartość oczekiwana tego zagrania to $\frac{1}{2} \cdot 630 + \frac{1}{2} \cdot (-100) = 265$. Gdy jednak nasz przeciwnik dołoży damę lub kartę innego koloru (każda z tych sytuacji wydarzy się z prawdopodobieństwem 0,0078), położymy króla i nasz wynik to odpowiednio +630 i +600. Zatem użycie metody „Impas” ma wartość $0{,}9854 \cdot 265 + 0{,}0078 \cdot 630 + 0{,}0078 \cdot 600 = 270{,}73$. Porównując oba wyniki, wybieramy zagranie asa zamiast impasu.

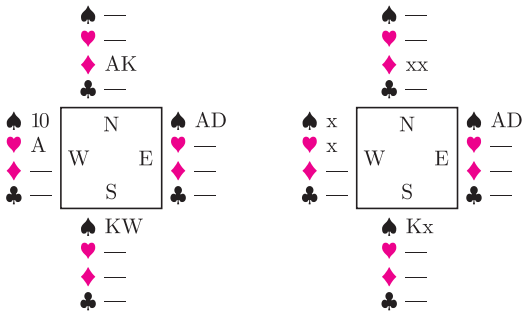
[*]student, Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski

Okazuje się, że takie podejście pozwala na istotne zmniejszenie drzewa gry. Faktycznie, w naszym przypadku drugi gracz może zagrać na trzy różne sposoby (mała karta, dama, karta nie do koloru). Jeśli natomiast rozpatrywalibyśmy wszystkie możliwe przypadki, to mógłby on zagrać jedną z 26 kart znajdujących się w zakrytych rękach przeciwników. Algorytm zastosowany w programie *Bridge Baron 8* korzystał z 91 metod i około 400 podmetod. Mimo dużej liczby możliwych strategii drzewo gry średnio zawierało teraz tylko 26 000 liści, a pesymistycznie 305 000, czyli 10^{39} razy mniej niż w przypadku pełnego drzewa gry. Dzięki temu *Bridge Baron* stał się najlepszym programem grającym w brydża, co potwierdziły rozegrane w 1997 roku Mistrzostwa Świata Programów Komputerowych w Brydżu.

Dominacja *Bridge Barona* nie trwała jednak długo, gdyż zaledwie rok później zdetronizował go oparty na zupełnie nowym pomyśle program Matthew Ginsberga *GIB* (*Ginsberg's Intelligent Bridgeplayer*, zwany także *Goren In Box* od nazwiska znanego amerykańskiego brydżysty). W celu znalezienia optymalnego zagrania losuje on wiele (około 100) możliwych układów rąk przeciwników i dla każdego z nich sprawdza, jaki jest najlepszy ruch, analizując rozdanie w widne karty (tzn. widząc karty przeciwników). Następnie wybiera to zagranie, które średnio okazywało się najkorzystniejsze. O ile sama idea jest prosta, to problemem, przed którym stanął Ginsberg, była analiza w widne karty w czasie pozwalającym na jej stokrotne powtórzenie. I tu właśnie z pomocą przychodzi algorytm *Partition Search*.

W rozmowach o rozdaniach brydżyści, mówiąc o swoich kartach, pomijają wysokości niektórych kart (blotek). Karta ♠AD732 ♥K92 ♦D98 ♣72 to dla nich „piąty as dama, trzeci król, trzecia dama i dwie blotki", czyli ♠ADxxx ♥Kxx ♦Dxx ♣xx. To podejście jest uzasadnione, gdyż rzeczywiście to, czy w kolorze mamy asa czy króla, jest dużo ważniejsze niż to, czy mamy dwójkę czy czwórkę. Właśnie ten pomysł jest wykorzystywany przez algorytm *Partition Search*. Podobne przypadki są w nim utożsamiane i analizowane jednocześnie, a nie rozpatrywane pojedynczo.

Rozważmy rozkład kart widoczny na rysunku 2, odpowiadający sytuacji, w której do końca rozdania pozostały dwie lewy. Zauważmy, że jedyne kara w rozdaniu posiada zawodnik N, więc nie ma znaczenia, że są to as i król, gdyż nawet gdyby to były dwójka i trójka, to żaden z przeciwników nie posiadałby wyższej karty w tym kolorze. Z podobną sytuacją mamy do czynienia w przypadku kierów gracza W. Jeśli chodzi o kolor pikowy, to sytuacja jest trochę bardziej skomplikowana, ale tutaj także wysokości niektórych kart nie są istotne. Zauważmy bowiem, że E w każdej z dwóch pozostałych lew zagra „co najmniej" damę pik, a więc ani 10, ani walet nie mogą wziąć żadnej lewy. W ten sposób uzyskaliśmy rozkład widoczny po prawej stronie rysunku 2. Ponieważ iksy w kierach i karach możemy zastąpić dowolnymi kartami, a w pikach kartami niższymi od damy, więc tak naprawdę ten rozkład odpowiada $13 \cdot \binom{13}{2} \cdot 10 \cdot 9 = 91\,260$ różnym rozkładom.



Rys. 2. Oryginalne rozdanie i jego „uproszczona" wersja.

Aby wyjaśnić zasadę działania algorytmu *Partition Search*, wprowadźmy dwie nowe definicje. Rozważmy pewien zbiór pozycji (układów kart) S. Jeśli z danej pozycji p możemy przejść do pewnej pozycji ze zbioru S, wtedy p nazwiemy pozycją *osiągającą* S. Natomiast jeśli z pozycji p możemy przejść tylko do pozycji należących do zbioru S, to p nazwiemy pozycją *przechodzącą* do S. Idea algorytmu *Partition Search* polega nie tylko na obliczaniu wartości danej pozycji, ale także na znalezieniu pewnego zbioru pozycji R, które mają taką samą wartość. Dzięki temu, gdy kiedyś znowu trafimy na pozycję ze zbioru R, to nie będziemy musieli dla niej obliczać wartości od nowa, lecz skorzystamy z wartości wyznaczonej poprzednio. Załóżmy teraz, że celem jest znalezienie tych dwóch niewiadomych (tzn. wartości i zbioru pozycji o tej samej wartości) dla pewnej pozycji p. Jeśli ruch wykonuje gracz maksymalizujący wynik (rozgrywający), to po prostu sprawdzana jest wartość we wszystkich pozycjach, do których może on teraz przejść, i wybierana jest najkorzystniejsza. Pozostaje pytanie, jak znaleźć zbiór R. Ponieważ dla każdego z rozpatrywanych następników p oprócz jego wartości dostaliśmy także odpowiedni zbiór pozycji R, to możemy wyznaczyć dwa nowe zbiory: S_{all} i $S_{\max}$ – pierwszy zawiera wszystkie otrzymane pozycje, a drugi te, które odpowiadają wartości maksymalnej. Kiedy jakaś inna pozycja będzie miała taką samą wartość jak p? Stanie się tak na pewno, gdy będzie ona osiągała zbiór $S_{\max}$ (aby otrzymać co najmniej taką samą wartość) oraz przechodziła do zbioru S_{all} (aby nie uzyskać wartości lepszej). Zatem szukanym zbiorem jest przecięcie zbioru pozycji osiągających $S_{\max}$ ze zbiorem pozycji przechodzących do zbioru S_{all}. W przypadku gdy w danej pozycji ruch wykonuje gracz minimalizujący wynik, postępujemy analogicznie, tylko wybieramy ten z następników, który ma wartość najmniejszą. Oczywiście, sam problem znajdowania pozycji osiągających dany zbiór i przechodzących do danego zbioru może być trudny, lecz akurat w przypadku brydża znalezienie funkcji dobrze je przybliżających nie jest zbyt skomplikowane.

Skuteczność opisanej metody zależy od tego, jak duże są średnio rozmiary przecięć pozycji osiągających $S_{\max}$ z przechodzącymi do S_{all}. W wyniku analizy rozdań brydżowych okazało się, że zastosowanie algorytmu *Partition Search* (połączonego z alfabetą) zmniejsza średnie rozgałęzienie przeszukiwanego drzewa z r do $r^{3/4}$, gdzie r to średnie rozgałęzienie w przypadku użycia

samej alfabety (jest to zmiana podobna do tej, jaką powoduje użycie alfabety w stosunku do samego minimaksa).

Jakkolwiek *Partition Search* sprawdził się bardzo dobrze, okazało się, niestety, że sama idea losowania rozdań i analizowania ich w widne karty ma pewne wady. Po pierwsze, w brydżu istnieją rozgrywki wywiadowcze, polegające na ściągnięciu kilku swoich lew w celu zdobycia pewnych informacji o zakrytych rękach przeciwników, które ułatwią nam podjęcie decyzji kluczowych dla losów rozdania. *GIB* oczywiście tak nie zagra, jako że on w każdej symulacji te układy zna. Po drugie, problemem jest wyrzucanie wysokich kart – *GIB* zakłada, że przeciwnik trafi wszystkie decyzje (w końcu przecież gra w widne karty) i pozbywa się kart, których położenie wcale nie musi być oczywiste. Po trzecie, *GIB* nie zawsze rozróżnia rozgrywkę lepszą od gorszej. Załóżmy, że grając metodą A, wygramy zawsze, natomiast w metodzie B będziemy musieli zdecydować, na który układ kart przeciwników gramy, lecz decyzję tę odłożymy do następnej lewy. Dla *GIB*-a obie te rozgrywki są skuteczne w stu procentach (znając układ rąk przeciwników, będzie wiedział, na co zagrać), natomiast w rzeczywistości rozgrywka B może mieć tylko 50% szans. Aby radzić sobie z tymi problemami, w kolejnych wersjach *GIB*-a wprowadzono modyfikacje polegające na rozpatrywaniu nie pojedynczych pozycji, lecz par (pozycja, możliwe układy kart przeciwników), i przypisywaniu każdej z możliwych rozgrywek zbioru układów kart, przy których jest ona skuteczna. Szczegółowe informacje na ten temat można znaleźć w artykule [1]. Można tam także poczytać o ciekawym podejściu do licytacji, które wykorzystuje *GIB*, gdy znaczenie poszczególnych odzywek w danej sytuacji nie jest opisane w jego bazie. Generuje on wtedy zbiór rąk swoich i partnera zgodnych z dotychczasową licytacją i na ich podstawie poszczególnym odzywkom przyporządkowuje takie znaczenie, które niesie jak najwięcej informacji przydatnych do znalezienia optymalnego kontraktu.

Mimo tych wszystkich technik i specjalnych algorytmów komputerom nadal trochę brakuje, by wygrywać z ludźmi. Zia Mahmood – jeden z najlepszych brydżystów na świecie – był gotów kilkanaście lat temu założyć się o milion funtów (!), że żadna drużyna złożona z czterech komputerów nie wygra długiego meczu z drużyną złożoną z czterech graczy wybranych przez Zię. Chętny się nie znalazł, a sam Zia w 1997 r. wycofał się z zakładu. Czyżby zaczął się obawiać, że taki program komputerowy zostanie kiedyś napisany? Komputery są w istocie coraz lepsze – ale czy mogą być aż *tak* dobre?

Literatura

[1] M. Ginsberg, *GIB*: *Imperfect information in a computationally challenging game*, J. Artificial Intelligence Res. 14 (2001).

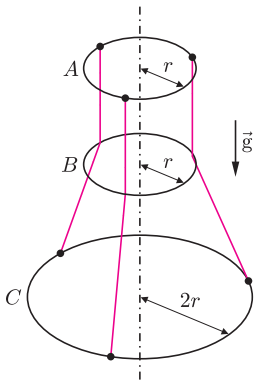
[2] S. J. J. Smith, D. Nau, T. Throop, Journal of Artificial Intelligence Research 14 (2001), AI Magazine 19 (2/2008), 93–105.

Zadania

Redaguje Ewa CZUCHRY

F 767. Jednorodna nić o masie m wisi swobodnie, zaczepiona dwoma końcami na tej samej wysokości. Siła napięcia nici w jej najniższym punkcie wynosi N_0. Znaleźć siły napięcia nici w punktach jej zaczepienia.
Rozwiązanie na str. 17

F 768. Trzy nierozciągliwe nici jednakowej długości są przymocowane w jednakowych odstępach do obręczy A oraz do obręczy C i przechodzą przez wnętrze obręczy B (rys. 1). Promienie obręczy A i B są jednakowe i wynoszą r, a promień obręczy C jest równy $2r$. Wszystkie obręcze wykonane są z takiego samego drutu, układ wisi na obręczy A, utrzymywanej w płaszczyźnie poziomej. Znaleźć odległość między środkami obręczy B i C.
Rozwiązanie na str. 17



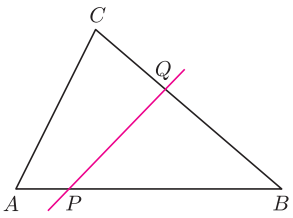
Rys. 1

Redaguje Waldemar POMPE

M 1282. Punkt P leży na boku AB trójkąta ABC. Za pomocą cyrkla i linijki poprowadzić przez punkt P prostą dzielącą trójkąt ABC na dwie figury o równych polach (rys. 2).
Rozwiązanie na str. 17

M 1283. Rozpatrujemy ciąg $101, 10101, 1010101, \ldots$ zapisany w systemie o podstawie $b \geqslant 2$. Udowodnić, że ciąg ten zawiera co najmniej jeden wyraz będący liczbą złożoną.
Rozwiązanie na str. 24

M 1284. Niech A będzie $(3n+1)$-elementowym podzbiorem zbioru $\{1, 2, \ldots, 4n-1, 4n\}$, gdzie n jest liczbą całkowitą dodatnią. Wykazać, że zbiór A zawiera takie trzy elementy $a < b < c$, że $a \,|\, b$ oraz $b \,|\, c$.
Rozwiązanie na str. 17



Rys. 2