Modeling Go Game as a Large Decomposable Decision Process

(autoreferat rozprawy doktorskiej)

Łukasz Lew

4 października 2011

Od zarania dziejów ludzie marzyli o budowie maszyn, które byłyby godnymi przeciwnikami ludzi w grach logicznych. To marzenie stało się realne wraz z pojawieniem się komputerów, szczególnie ostatnio, gdy olbrzymia moc obliczeniowa jest w zasięgu ręki. Spektakularnym przykładem gracza komputerowego jest system Deep Blue, który wygrał z szachowym mistrzem świata Garrym Kasparowem. Rozwiązania użyte w szachach to przeszukiwanie drzewa gry do pewnej głębokości i użycie funkcji oceniającej w liściach drzewa do oceny szans wygranej. To podejście przeniosło się na wiele innych gier pozwalając komputerom grać lepiej niż najlepsi ludzie lub wręcz rozwiązać daną grę. Jednym z nielicznych wyjątków jest gra Go.

Gra Go. Drugi rozdział rozprawy przedstawia historię gry Go i jej wielką popularność w Azji. Wyjaśnione są jej zasady oraz podstawowe elementy strategii i taktyki. Podkreślone są te elementy gry, które sprawiają komputerom trudność i odróżniają ją od innych tradycyjnych gier planszowych. Wyjaśnione są terminy *dobry kształt* i *zły kształt* w kontekście Go. Jest to forma *wiedzy* o grze, którą ludzie przyswajają i używają w trakcie gry. Ta wiedza jest także kluczowym elementem algorytmów grających.

Algorytmy grające. Algorytmy grające w Go zawsze były relatywnie słabe. Powodów dla rozbieżności siły gry komputerów i ludzi w grze Go jest kilka. Jednym z nich jest brak znanej, dobrej funkcji oceniającej pozycje w grze. Jest tak, gdyż nie znamy algorytmu, który wydajnie estymowałby szanse wygranej dowolnej pozycji Go.

Badacze i hobbyści tworząc algorytmy grające w Go, starając się odwzorować ludzi sposób myślenia. Rozdział 3.1 przedstawia 30 lat rozwoju "klasycznych" algorytmów Go. Podkreślona jest komplikacja programów tego typu i trudność ich dalszego ulepszania. "Klasyczne" podejście jest przedstawione na przykładzie popularnego programu Gnu Go [BGB05].

Monte Carlo Go. Niedawnym i rewolucyjnym pomysłem poradzenia sobie z problemem braku funkcji oceniającej jest podejście *Monte Carlo Go*. Polega ono przeprowadzenie dla *danej pozycji* bardzo wielu *playoutów* – rozgrywek złożonych z *losowych* ruchów prowadzących do samego końca gry. Dana pozycja jest oceniana na podstawie statystycznej analizy wyników tych rozgrywek. W ciągu kilku ostatnich lat, algorytmy z rodziny Monte Carlo Go znalazły zastosowanie nie tylko w grze Go, ale także w wielu innych grach, jak i w zupełnie odległych od Go dziedzinach [dMRVP09].

Rozdział 3.2 przedstawia niedawną historię odkrycia i rozwoju Monte Carlo Go. Kluczowym elementem rozwoju Monte Carlo Go było stworzenie algorytmu *UCT* [KS06] albo ogólniej *Monte Carlo Tree Search* oraz jego usprawnienia *RAVE* [GS07]. Jest to wariant algorytmu MINIMAX przystosowany do zaszumionej, randomizowanej funkcji oceniającej jaką są wyniki playoutów. Działanie algorytmów MCTS i RAVE jest wyjaśnione w rozdziale 3.3.2.

Wiedza dziedzinowa Go. Drugim kluczowym elementem rewolucji Monte Carlo Go jest wbudowanie w MCTS i w playouty wiedzy dziedzinowej Go [CWvdH$^+$07, GWMT06]. Wiedza ta może być reprezentowana za pomocą wag przypisanych do wielu konfiguracji (kształtów) mogących wystąpić na planszy. Użycie wiedzy w MCTS znacząco zwiększa wydajność przeszukiwania poprzez zawężenie przeszukiwania do obiecującego poddrzewa. Użycie wiedzy do losowania ruchów wewnątrz playoutów znacząco polepsza jakość funkcji oceniającej i ma jeszcze większy efekt na siłę gry Monte Carlo Go. Rozdziały 3.3.2 i 3.4 wyjaśniają znane dobre sposoby implementacji użycia wag w MCTS i playoutach.

Źródła wiedzy. Dobre wagi konfiguracji można próbować dobierać ręcznie metodą prób i błędów [GWMT06] lub znajdować je za pomocą algorytmów typu "Machine Learning" [Cou07]. Dobrze służy temu celowi model Bradley'a – Terry'ego (BT) i algorytm jego optymalizacji Minorization – Maximization (MM) [Hun04]. Ograniczeniem MM są duże wymagania pamięciowe. Komputer z 24 GB pamięci RAM może przetworzyć jedynie 20'000'000 wyborów (pozycji Go z zagranym ruchem), podczas gdy dostępne dane są kilkadziesiąt razy większe.

Pierwszym wynikiem tej rozprawy jest nowa rodzina algorytmów *Laplace Marginal Propagation*. Służy ona do trenowania modeli parametrycznych takich jak BT. Rozdział czwarty, po przedstawieniu podstaw modelowania Bayesowskiego i podstawowych algorytmów trenujących, stosuje LMP do modelu BT. Główną zaletą uzyskanego algorytm treningowego są małe wymagania pamięciowe – dla danych, dla których MM potrzebował 24 GB RAM, LMP potrzebuje jedynie 15 MB. Jego innymi zaletami są prostota implementacji, możliwość pracy na danych strumieniowych o potencjalnie

nieskończonych rozmiarach (takich jak playouty) oraz brak parametrów takich jak współczynnik uczenia (w przeciwieństwie do np. Stochastic Gradient Descent). Rozdział czwarty przedstawia także dowód zbieżności algorytmu.

Rozkładalność na podproblemy. Inną istotną cechą odróżniającą Go od innych gier, a upodobniającą Go do wielu innych problemów decyzyjnych, jest rozkładalność pozycji Go na wiele prawie niezależnych pod-gier. Ludzie grając w Go są w stanie wykorzystać tę cechę, oceniając wartość ruchów w każdej pod-grze niezależnie. Rozdział piąty analizuje finalną pozycję gry pomiędzy człowiekiem i komputerem jako ilustrację typowych słabości programów Go.

Obecnie nie są znane algorytmy typu dziel i rządź, które można byłoby zastosować do gry w Go. Częściowo jest to spowodowane samą charakterystyką gry Go, częściowo koniecznością zintegrowania takich algorytmów z podejściem typu Monte Carlo, a częściowo faktem iż pod-gry Go są często tylko częściowo niezależne.

Obecnie najlepsze algorytmy grające w Go, nie wykorzystują pod-gier i rozpatrują każdą konfigurację planszy jako niezależny punkt w globalnej przestrzeni stanów. Wynikiem tego jest kombinatoryczna eksplozja liczby stanów gry i wielkości drzewa gry. Z tego powodu wybór dobrego kolejnego ruchu na podstawie przeszukania drzewa gry jest bardzo trudny. Ten problem jest łatwo dostrzec w fakcie, że ludzie radzą sobie równie dobrze na dużej planszy 19×19, jak i na planszy małej 9×9, podczas gdy algorytmy grają dużo słabiej na dużej planszy niż na małej. Jest tak, gdyż na małej planszy niezależne pod-gry należą do rzadkości.

Adaptatywne playouty. Nadzieją na pokonanie tej słabości algorytmów są tak zwane *adaptatywne playouty*. Jest to algorytm zbliżony ideą do podejścia "dziel i rządź". Miałby on dostosowywać się podobnie jak algorytm MCTS do aktualnej pozycji. Dostosowanie powinno być lokalne – zmiana pozycji w jednej części planszy nie powinna wpływać na dotychczas zebraną z playoutów wiedzę w innej, odległej pod-grze.

Naturalnym podejściem do adaptatywnych playoutów jest algorytm modyfikujący wagi konfiguracji w sposób analogiczny do tego jak MCTS wykorzystuje wyniki playoutów. W prywatnej korespondencji autor ustalił że wielu badaczy próbowało zaimplementować ten algorytm i w każdym przypadku wyniki były niezadowalające. Nikt jednak nie zdołał ustalić przyczyn tej porażki.

Rozdział 5.1 definiuje bardzo prostą wyidealizowaną grę składającą się z trzech niezależnych bardzo małych pod-gier. Mimo że globalna liczba stanów wynosi jedynie $3 \times 3 \times 5 = 45$ to ten adaptatywny algorytm nie potrafi jej rozwiązać. Rozdział 5.2 wskazuje na fundamentalny problem uniemożliwiający działanie algorytmu. Jest to drugi wynik referowanej rozprawy doktorskiej.

Model liniowy. Drugi popularnym podejściem do adaptatywnych playoutów jest przekazywanie wiedzy "w poprzek" drzewa pomiędzy podobnymi stanami. Jest to na ogół osiągane poprzez użycie modelu liniowego do składowania wiedzy o grze. Istnieje bardzo duża ilość badań nad połączeniem algorytmów z rodziny Monte Carlo (lub ogólniej Temporal Difference Learning) z modelem liniowym [BBK96]. Istnieją także rozbudowane badania nad zastosowaniem modelu liniowego go Go [CGJB$^+$08, GS07].

Referowana rozprawa argumentuje, że stosowanie model liniowego do Go jest skazane na porażkę. Rozdział 5.3 rozprawy przeprowadza eksperyment na tej samej prostej grze pokazujący, że model liniowy nie zbiega nawet na tak prostym przykładzie. Rozdział 5.4 analizuje ten wynik i wskazuje na fundamentalny problem modelu liniowego uniemożliwiający reprezentację dobrych strategii w grach składających się z pod-gier. Jest to trzeci wynik referowanej rozprawy.

Dwa ostatnie wyniki oraz brak pozytywnych wyników innych badaczy wykazują trudność stworzenia poprawnego adaptatywnego lokalnego algorytmu.

Algorytm oparty na teorii gier kombinatorycznych. Rozdziały 5.5 i 5.6 wprowadzają do *teorii gier kombinatorycznych* i *termografów* [BCG82] – modelu, który w poprawny sposób modeluje niezależne pod-gry. Model ten jest zbyt skomplikowany by można go było bezpośrednio zastosować do pod-gier powstających w Go (z wyłączeniem końcówek). Rozdział 5.7 wprowadza *trójkątną aproksymacje termografu*, która pozwala na znaczne uproszczenie obliczeń.

Trójkątna aproksymacja pozwala na stworzenie algorytmu Monte-Carlo analogicznego do MCTS, obliczającego termografy dla każdej z pod-gier. Eksperyment wykazuje bardzo szybką zbieżność i pozwala znaleźć strategie bliskie optymalnym.

Trójkątna aproksymacja i korzystający z niej algorytm są czwartym wynikiem rozprawy.

Libego. Ostatnim lecz nie mniej ważnym wynikiem jest biblioteka programistyczna Libego. Jest to oprogramowanie pozwalające na przeprowadzanie eksperymentów w komputerowym Go. Zostało ono opublikowane na otwartej licencji GPL i zostało użyte przez wielu niezależnych badaczy i hobbystów do przeprowadzania niezależnych badań nad algorytmami Go. Rozdział szósty opisuje funkcjonalność Libego oraz opublikowane wyniki uzyskane z pomocą Libego.

Literatura

[BBK96] Steven J. Bradtke, Andrew G. Barto, and Pack Kaelbling. Linear least-squares algorithms for temporal difference learning. In *Machine Learning*, pages 22–33, 1996.

[BCG82] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways*, volume I and II. Academic Press, 1982.

[BGB05] Daniel Bump, Farneback Gunnar, and Arend Bayer. GnuGo, 2005.

[CGJB+08] Louis Chatriot, Sylvain Gelly, Hoock Jean-Baptiste, Julien Perez, Arpad Rimmel, and Olivier Teytaud. Combining expert, offline, transient and online knowledge in monte-carlo exploration, 2008.

[Cou07] Rémi Coulom. Computing Elo ratings of move patterns in the game of Go. *ICGA Journal*, 30(4):198–208, December 2007.

[CWvdH+07] Guillaume Chaslot, Mark Winands, Jaap H. van den Herik, Jos Uiterwijk, and Bruno Bouzy. Progressive strategies for Monte-Carlo tree search. In *Joint Conference on Information Sciences, Salt Lake City 2007, Heuristic Search and Computer Game Playing Session*, 2007.

[dMRVP09] Frédéric de Mesmay, Arpad Rimmel, Yevgen Voronenko, and Markus Püschel. Bandit-based optimization on graphs with application to library performance tuning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 729–736, New York, NY, USA, 2009. ACM.

[GS07] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *International Conference on Machine Learning, ICML 2007*, 2007.

[GWMT06] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA, France, November 2006.

[Hun04] R. Hunter. MM algorithms for generalized Bradley-Terry models. *The Annals of Statistics*, 32:2004, 2004.

[KS06] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML-06*, 2006.