Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji
Katedra Informatyki

Autoreferat rozprawy doktorskiej

Algorytmy stadne w optymalizacji problemu przepływowego szeregowania zadań

Wiesław Popielarski

Promotor:
prof. dr hab. inż Bogusław Filipowicz

Kraków 2014

Spis treści

Problem szeregowania zadań

Szeregowanie zadań jest z pewnoscią jednym z ciekawszych problemów kombinatorycznych. Składają się na to trzy przyczyny.

Po pierwsze ta klasa problemów jest bardzo bogada różnorodnością nietrywialnych modeli, gdzie każdy z nich wymaga innego sposobu rozwiązania. I tak okazuje się, że już w grupie modeli szeregowych można wyodrębnić kilkanaście podproblemów określonych tylko i wyłącznie rodzajem funkcji celu. Do tego dochodzą bardziej ogólne podziały wynikające z charakterystyki przetwarzania zadań, przykładowo, ich wzajemne zależności albo możliwość ich wywłaszczania. Kolejny podział związany jest z czasem przetwarzania zadań, czy jest on deterministyczny, czy też jest opisany przez zmienną losową. Jeżeli jeszcze nałożymy możliwość przetwarzania na więcej niż jednej maszynie, to możemy mieć wyobrażenie rzeczywistej różnorodności problemów, które pomimo podobieństwa są nieredukowalne w rozumieniu znajdowania rozwiązania. Książkowym przykładem jest problem *flow shop*, dla którego istnieje wielomianowy algorytm znaleziony przez dra S. H. Johnsona w przypadku dwóch maszyn, jednakże dla trzech i więcej problem należy już do klasy problemów NP.

Drugą przyczyną zainteresowania jest właśnie trudność tych problemów. Dzięki temu, że zdecydowana większość modeli należy do klasy NP, ich badacze zmuszeni są do ciągłego poszukiwania coraz lepszych (wydajniejszych oraz dokładniejszych) rozwiązań. Co więcej, przynależność do klasy NP powoduje, że trudno jest znaleźć ogólny algorytm znajdujący rozwiązanie optymalne (albo jemu wystarczająco bliskie), w akceptowalnym czasie (a więc nie rozpatrujemy w tym miejscu przeszukiwania wyczerpującego - backtrackingu, który jest algorytmem stosującym się do wszystkich problemów kombinatorycznych), dający się zastosować do większego zbioru problemów. Taki algorytm przypominałby swoim działaniem algorytm *simplex* w rozumieniu, że wejściem jest model określony przez pewien zbiór ograniczeń i funkcję celu, natomiast sam sposób poszukiwania rozwiązania jest od tego modelu niezależny.

W końcu trzecią przyczyną zainteresowania, być może nawet najważniej-

szą, jest wielka praktyczność szeregowania zadań. W każdej dziedzinie życia można pośrednio lub bezpośrednio spotkać się z problemem, który sprowadza się do optymalizacji szeregowania zadań. Począwszy od ciągów technologicznych będących powodem, dla którego w ogóle zaczęto myśleć o optymalizacji szeregowania zadań, organizację pracy, na systemach operacyjnych skończywszy.

Inspiracje naturą

W poszukiwaniu ogólnego, dającego się łatwo adaptować do wymagań narzucanych przez poszczególne modele, sposobu rozwiązywania problemu szeregowania zadań, badacze zatrzymali się również nad algorytmami odkrytymi w przyrodzie. Wszytkie one charakteryzują się tym, że występują w nich dwa kroki, mianowicie krok eksploracji oraz krok eksploatacji. Oczywiscie te kroki implementowane są w różny sposób. I tak w przypadku algorytmów genetycznych mamy mutację i krzyżowanie, w przypadku symulowanego wyżarzania tunelowanie i schładzanie, natomiast w przypadku algorytów stadnych (*swarm intelligence*) jednostka albo decyduje się na skauting, albo podąża za którymś z liderów, którzy w poprzednim kroku odkryli rokujace rozwiązanie.

W tym miejscu dochodzimy do drugiego spostrzeżenia. W każdym z powyższych kroków losowany jest wybór dalszego postępowania. I tak w przypadku algorytmu genetycznego losowany jest gen, który mutuje albo taki który będzie się krzyżować. Z kolei tunelowanie jest symulacją efektu kwantowego, pozwalającego na opuszczenie lokalnego optimum. A w przypadku algorytmów stadnych jednostka decyduje (poprzez losowanie) czy chce eksploatować sąsiedztwo bieżącego wyboru (może to czynić deterministycznie), czy dalej eksplorować przestrzeń rozwiązań, czy też w końcu przyłączyć się do eksploatacji jednego z rokujących (potencjalnych) optimów. Stąd też kostatacja, że procesy zachodzące w przyrodzie są generalnie procesami stochastycznymi.

W ten oto sposób natura daje nam generyczne narzędzie do przeszukiwania dużych przestrzeni rozwiązań w inny od backtrackingu sposób, który wprawdzie nie gwarantuje, że znaleziona wartość będzie globalnym optimum,

ale z drugiej strony wartość ta będzie wystarczająco dobra ze stochastycznego punktu widzenia. Algorytm można zapisać poniższym pseudokodem:

procedure SEARCH($best, pos$)
 if $stopconditionis$ **true**
  then return ($best$)
 return (MAX(EXPLORESPACE(pos), EXPLOITPROXIMITY(pos)))

Natomiast *exploreSpace* oraz *exploitProximity* zdefiniowane są następująco:

procedure EXPLORESPACE(pos)
 $newPosition \leftarrow$ GETNEXTRANDOM(pos)
 $newBest \leftarrow$ COMPUTEGAIN($newPosition$)
 return (SEARCH($newBest, newPosition$))

procedure EXPLOITPROXIMITY(pos)
 $newPosition \leftarrow$ GETNEXTNEIGHBOR(pos)
 $newBest \leftarrow$ COMPUTEGAIN($newPosition$)
 return (SEARCH($newBest, newPosition$))

Pułapki natury

W latach 90-tych David Wolpert oraz William G. Macready sformułowali hipotezę *No Free Lunch*, która mówi, że prawdopodobnie nie istnieje specjalizowany algorytm stochastyczny, który dla wszystkich problemów byłby lepszy od spaceru losowego. Inaczej mówiąc każde dwa dowolne algorytmy stochastyczne rozwiązują równie dobrze prawie wszystkie problemy. To tłumaczy obserwację, że dla pewnych przestrzeni algorytm pszczeli zachowuje się lepiej od kukułki i vice versa. Być może wytłumaczeniem tego faktu są rezultaty badań otrzymane przez Wolperta i Macready'ego dotyczące optymalizacji koewolucyjnej, z których wynika, że w układach, w których sposób przeszukiwania wpływa na wartosci funkcji celu darmowe obiady są jednak możliwe. W takim razie można postawić pytanie o to jakimi cechami charakteryzuje się przeszukiwana przestrzeń, która byłaby podatna na szczególny

(na przykład pszczeli lub kukułki) algorytm?
Niestety odpowiedź na powyższe pytanie nie jest znana. Od połowy lat 2000-nych Marco Tomassini, Sebastien Verel oraz Gabriela Ochoa prowadzą badania nad własnościami krajobrazów kombinatorycznych (*combinatorial landscapes*), w ramach których chcą znaleźć odpowiedzi m.in. o dopuszczalny maksymalny gradient funkcji celu zdefiniowanej na danym krajobrazie oraz czy istnieje taki izomorfizm danego krajobrazu na inny, w którym można by taki maksymalny gradient okreslić. Jak dotąd nie osiągnięto zadowalających rezultatów (w moim przekonaniu powinien istnieć jakis związek pomiędzy funkcją celu a warunkiem Lipschitza).

Istnieje jeszcze jedna pułapka w stosowaniu algorytmów stochastycznych w szczególnosci stadnych. Jak to zostało powiedziane, wartości znajdowane przez tego rodzaju algorytmy są wystarczająco dobre ze **stochastycznego** punktu widzenia. W swojej pracy posłużyłem się nierównością Chernoffa oraz metodą sprzężonych łańcuchów Markowa w celu wyznaczenia całkowitej liczby kroków algorytmów. W założeniu podejście takie powinno skutkować tym, że

$$\lim_{n\to\infty} P\{|a_n - opt| > \epsilon\} = 0.$$

Jeżeli jednak popatrzymy dokładniej na nierówność Chernoffa, zauważymy, że *opt* jest w rzeczywistości wartością oczekiwaną. Z drugiej strony poszukiwana wartość optymalna jest wartością skrajną, która będzie zawsze niemniejsza od wartości oczekiwanej. Dzięki temu znaleziona wartość oczekiwana może posłużyć jako kryterium dalszego badania sąsiedztwa danego punktu przestrzeni rozwiązań (oznaczając ją jako obszar rozwiązań niedopuszczalnych).

Implementacja

Do badania wybrano dwa algorytmy stadne: pszczeli oraz kukułki, z tego względu, że mają one różne charakterystyki. Po pierwsze, różnią się zastosowaną funkcją gęstości prawdopodobieństwa (odpowiednio rozkład eksponencjalny i Levy'ego). Po drugie, sposób wyznaczania sąsiedztwa jest też różny

(założenia odnośnie sąsiedztwa są przynajmniej w częsci omówione w [1]).
Jako język programowania zastosowano *Scalę*, natomiast badanie wykonywano na IDE *Typesafe*. Wybór *Scali* jest umotywowany tym, że mocno wspiera programowanie wielowątkowe oparte o model *aktorów*, który z kolei został zaadaptowany z *Erlanga*, dzięki czemu możliwa była redukcja złożności problemu w sensie prawa Amdahla.
Obecna architektura nie pozwala jednak na pełne wykorzystanie środowiska obliczeń rozproszonych, ponieważ zarówno w algorytmie pszczelim jak i kukułki istnieją obiekty pośredniczące w wymianie wiadomości pomiędzy instancjami pszczół i kukułek (odpowiednio *hive* oraz *forest*). Najlepszym rozwiązaniem tego problemu byłaby replikacja tych obiektów.

Konkluzja

Mimo że badano algorytmy w kontekscie modelu *flow shop*, to przeniesienie ich na model *job shop*, który jest większy jedynie o czynnik n, nie byłoby trudne (złożoność *flow shop* to $O(m!)$, a *job shop* to $O(m!n)$). W przypadku *open shop* przestrzeń rozwiązań jest nieporównywalnie większa ($O(n!m!)$) .

Wracając do sposobu wyznaczania liczby kroków za pomocą nierówności Chernoffa i spostrzeżenia, że dotyczy ona wartości oczekiwanej a nie optymalnej można by się zastanowić nad modyfikacją funkcji celu tak, aby nigdzie nie była ona mniejsza od wyznaczonej aktualnie wartości oczekiwanej. Następnie rozważyć jak takie rozwiązanie wiąże się z przekształceniami krajobrazu (niekoniecznie izomorficznymi) i czy nie można by w ten sposób otrzymać optymalizacji koewolucyjnej, w której darmowe obiady są możliwe.

Bibliografia

[1] W.Popielarski, *The neighborhood choice in* $F_m|prmu|C_{\max}$ *model*, http://mechatronika.polsl.pl/owd/pdf2013/025.pdf