Marcin Zięba

Inteligencja obliczeniowa

Raport 3

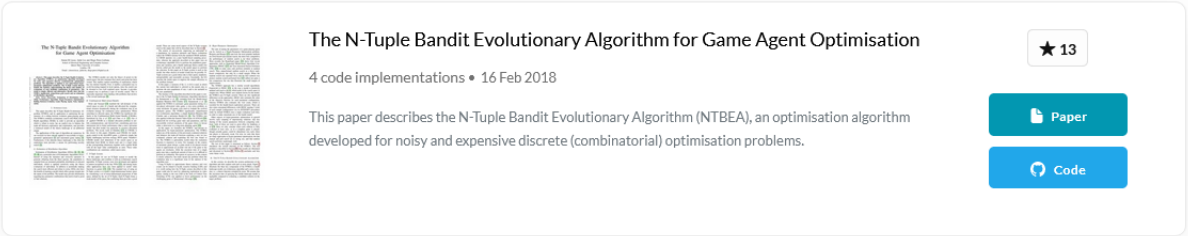
Temat: **Implementacja algorytmów ewolucyjnych na GPU z użyciem TensorFlow**

TL;DR; Poszukałem niedawnych prac z dziedziny algorytmów ewolucyjnych, i przejrzałem je w poszukiwaniu odpowiedniego algorytmu do implementacji zgodnie z założeniami projektu. Pięć prac uznałem za ciekawe, z czego dwie za szczególnie preferowane. Ponadto jako dodatek do poprzedniego raportu, dokonałem eksperymentów z uprzednio zaimplementowanym algorytmem mrówkowym na problemie o większym rozmiarze.

Q&A

Gdzie szukałem artykułów?

https://paperswithcode.com/search?q_meta=&q=evolutionary+algorithm
Papers with code to witryna, na której można znaleźć artykuły z szeroko pojętej dziedziny uczenia maszynowego - co ważne, gromadzi ona także linki do repozytoriów z implementacjami. Okazuje się, że znajdują się tam także prace z dziedziny algorytmów ewolucyjnych. Ogólnie przeglądanie prac z poziomu tej strony okazało się najwygodniejszą opcją, ponieważ bardzo łatwo znaleźć tam interesujące mnie informacje.



Jakie artykuły uznałem za godne uwagi?

- “EBIC: an evolutionary-based parallel biclustering algorithm for pattern discover”, 2018

https://paperswithcode.com/paper/ebic-an-evolutionary-based-parallel
Jest to raczej ciężki problem i z dość ciężkim rozwiązaniem. Dostępny oficjalny kod jest dosyć skomplikowany, pisany w większości w C++, z pewnymi modułami w CUDA. Powstała praca, której głównym celem jest adaptacja EBIC w Julii.

EBIC.JL, https://paperswithcode.com/paper/ebic-jl-an-efficient-implementation-of

- “The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation”, 2018

https://paperswithcode.com/paper/the-n-tuple-bandit-evolutionary-algorithm-for
Raczej przystępny algorytm. Nie wydaje się jakoś bardzo skomplikowany. Są dostępne 4 implementacje, ale żadna na GPU. Raczej bezpieczny temat.

- “Black-Box Ripper: Copying black-box models using generative evolutionary algorithms”, 2020

https://paperswithcode.com/paper/black-box-ripper-copying-black-box-models
To co autorzy proponują brzmi trochę jak magia. Musiałbym bardzo solidnie wczytać się w treść artykułu. Problem bazuje na modelach generatywnych, co prawdopodobnie najpierw zakładałoby uprzednio moją dobrą znajomość tej tematyki.

- “Hybrid Self-Attention NEAT: A novel evolutionary approach to improve the NEAT algorithm”, 2021

https://paperswithcode.com/paper/hybrid-self-attention-neat-a-novel
Doświadczenia z innych przedmiotów raczej podpowiadają mi, żeby trzymać się z dala od neuroewolucji, bo to się okropnie długo liczy.

- “CEM-RL: Combining evolutionary and gradient-based methods for policy search”, 2018

https://paperswithcode.com/paper/cem-rl-combining-evolutionary-and-gradient-1
Dosyć długi artykuł może być odstraszający. Dodatkowo częścią algorytmu jest rozwiązanie pochodzące z innej pracy tego samego autora, która tutaj jest uznana za pewnik. Nie mniej jednak ciekawe połączenie uczenia ze wzmocnieniem z metodami gradientowymi. Prawdopodobnie może się niestety długo liczyć. Repozytorium zawiera dosyć dużo kodu. Raczej ryzykowny projekt, jakkolwiek niewątpliwie ciekawy.

Jakie są preferowane przeze mnie prace?

Preferencja 1 (najbardziej preferowane):
- “The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation”

Preferencja 2:
- “CEM-RL: Combining evolutionary and gradient-based methods for policy search”

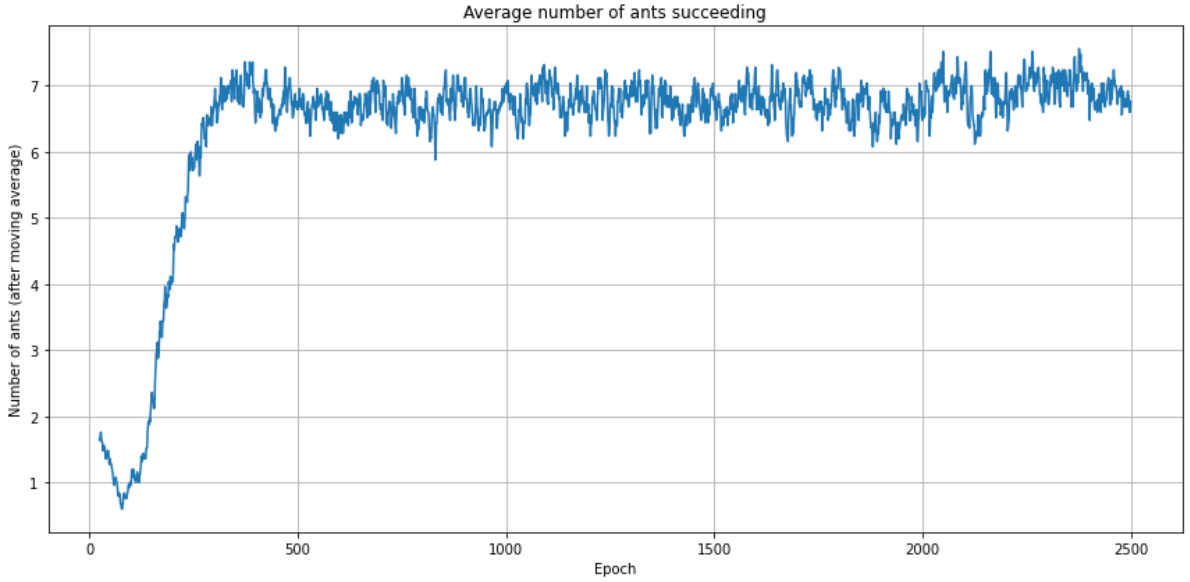
Jakimi problemami się zajmę?

Wezmę mniej więcej te problemy, które tematem przedstawionych prac.

Dodatek do raportu z etapu drugiego

Uruchomiłem przedstawiony na poprzednich zajęciach algorytm mrówkowy na większym (150 wierzchołków zamiast 10) grafie (odpowiednio stworzonym, co nie jest trywialne) i z większą liczbą mrówek (100 zamiast 9), żeby sprawdzić przyśpieszenie jakie oferuje GPU. Z dobrych informacji, algorytm nadal zbiega.

Colab: [Algorytm mrówkowy na większym problemie]



Czasy obliczeń (wyniki z mniejszego eksperymentu zapożyczone z poprzedniego raportu). Wszystko liczone z pełną optymalizacją, tj. wewnątrz tf.function (zredukowany tracing i włączony JIT compile (XLA)), o ile nie zaznaczono inaczej.

	CPU	GPU (standard)
150,000 epok 10 wierzchołków 9 mrówek	11.3856s	18.3746s
5,000 epok 150 wierzchołków 100 mrówek	9.0653s (4.6830s bez XLA na 2500 epok)	1.6399s (12.0171s bez XLA)

Tak jak wcześniej GPU działało około połowę wolniej niż CPU, tak teraz przyśpieszenie jest już 6-krotne.

Niestety okazało się, że któreś elementy implementacji nie działają prawidłowo, i przełączając wyłącznie jedną flagę (jit_compile, czyli kompilację XLA), zmieniają się wyniki algorytmu! Tak nie powinno być. Z włączoną kompilacją XLA algorytm przestaje zbiegać, wykres liczby mrówek osiągających sukces zaczął wyglądać na losowy. Pierwszy podejrzany - generacja liczb losowych. Zamiana generatora ze stateful na stateless (seed zależny od numeru epoki) jednak nie pomogła, pomimo iż testy w mikroskali pokazują, że to działa. Być może bug jest w kompilatorze XLA?

Wygenerowany na potrzeby problemu graf, wraz z przykładową najlepszą ścieżką znalezioną przez mrówki.