Marcin Zięba

Inteligencja obliczeniowa

Raport 2

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

TL;DR; Zaimplementowałem prosty algorytm mrówkowy z wykorzystaniem TensorFlow do rozwiązania niewielkiego problemu - szukania najkrótszej ścieżki w grafie. Priorytetami na tym etapie była łatwość debugowania i możliwość łatwej wizualizacji działania algorytmu. Stworzony kod ma charakter proof of concept. Niemniej jednak opisanie algorytmu w TensorFlow w sposób, w jaki jest on kompilowalny przez XLA jest w ogólności wykonalne (niektóre operacje okazały się jednak niewspierane, więc musiały być zaimplementowane nieco mniej wydajnie).

Colab: [Implementacja w TensorFlow i kilka testów]

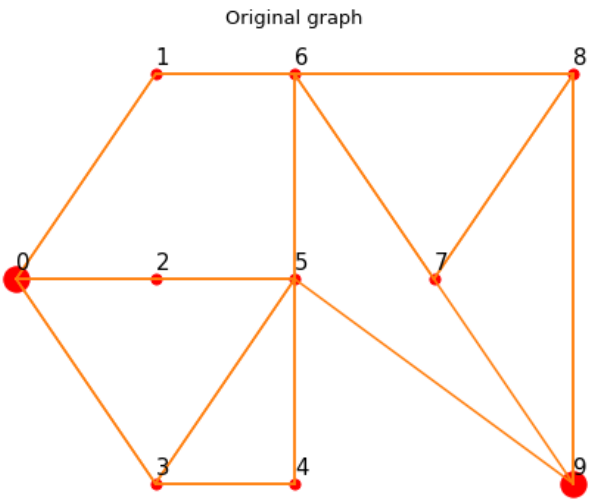
Q&A

Jaki algorytm implementowałem?

Prosty algorytm mrówkowy, nieco dostosowany do potrzeb, jakie wystąpiły przy implementacji. Możliwe jednak, że coś pomieszałem, ale to chyba najbardziej goła implementacja, jak to możliwe.

Jak testowałem poprawność algorytmu?

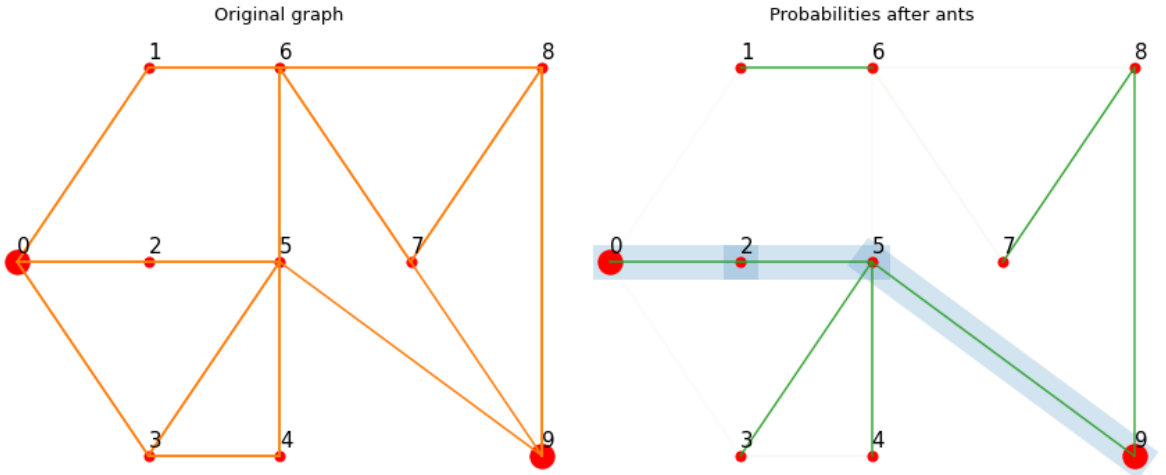
Uruchamiałem go do znajdowania najkrótszej ścieżki w poniższym grafie nieważonym (z wierzchołka 0 do 9), z populacją 9 agentów. Graf wewnętrznie jest przechowywany w formacie COO.

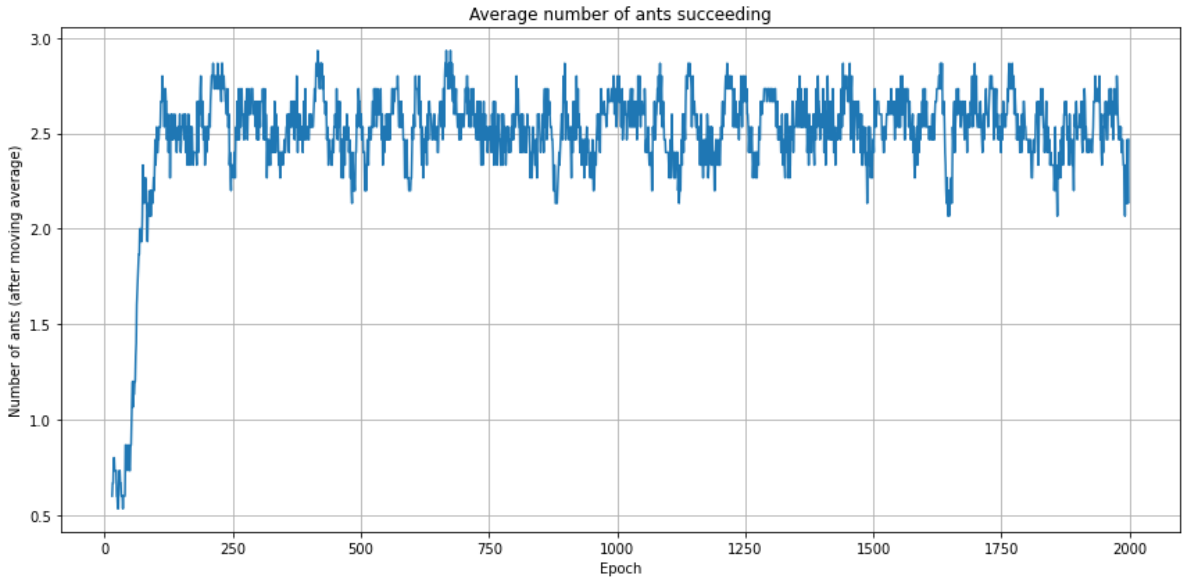


```
N = 10  # number of nodes in graph

# COOrdinate graph representation
row = tf.constant(np.array([0, 0, 0, 1, 1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 8, 9, 9], dtype=np.int32))
col = tf.constant(np.array([1, 2, 3, 0, 6, 0, 5, 0, 4, 5, 3, 5, 2, 3, 4, 6, 9, 1, 5, 7, 8, 6, 8, 6, 7, 9, 7, 8], dtype=np.int32))
hashes = 10 * row + col
```

Najkrótsza ścieżka w tym grafie ma długość 3, więc średnio co krok 3 mrówki powinny osiągać wierzchołek docelowy, jeśli algorytm się zbiega.





W aktualnej implementacji pozostaje nieredukowalny błąd wynikający z mechanizmu narzucającego eksplorację, który obecnie nie jest wygaszany wraz z epokami (a mógłby!).

Generalnie widać zbieżność do oczekiwanej liczby mrówek osiągających sukces.

Jedna z optymalnych ścieżek również jest zazwyczaj poprawnie znajdowana.

W jaki sposób dostosowuję kod pod uruchamianie na GPU?

- Pisanie kodu opartego wyłącznie o operacje TensorFlow.
- Użycie dekoratora tf.function - wygenerowanie lepszego grafu obliczeń do celów optymalizacyjnych (wymaga stosowania się nieco bardziej restrykcyjnych reguł pisania kodu).
- Wyłączenie taśmy monitorującej gradienty - nie jest to nam potrzebne.
- Uruchomienie kompilacji XLA (opcja `jit_compile=True`).

Jak wygląda taki kod?

Wykonanie pojedynczego przejścia wszystkich mrówek na następną pozycję:

```
@tf.function
def traverse_one_step(row, col, ants, threshold):
    thresholds_per_agent = tf.gather(
        tf.where(row == tf.reshape(tf.range(10, dtype=tf.int32), [-1, 1]), threshold, 0),
        indices=ants)

    targets = tf.gather(
        col,
        tf.argmax((thresholds_per_agent - tf.random.uniform([9])[:, tf.newaxis]) > 0, axis=-1))

    return targets
```

Wykonanie jednej epoki:

```
@tf.function
def proceed_one_epoch(row, col, hashes, ants, history, proba, starts, end, sink, source, h_len):
    threshold = tf.math.cumsum(proba) - tf.cast(row, dtype=tf.float32)
    ants = traverse_one_step(row, col, ants, threshold)
    history = tf.tensor_scatter_nd_update(history, tf.transpose(tf.stack([tf.fill([9], end), tf.range(9)])), ants)

    is_at_target = tf.boolean_mask(tf.range(9), ants == sink)
    t = tf.tile(tf.gather(history, is_at_target, axis=-1), [2, 1])[:h_len + end + 1]
    s = tf.gather(tf.where(starts > end, starts, starts + h_len), is_at_target)

    history = tf.tensor_scatter_nd_update(
        history, tf.transpose(tf.stack([tf.fill([9], end), tf.range(9)])), tf.where(ants == sink, source, ants))

    starts = tf.where(ants == sink, end, starts)
    starts = tf.where(ants == source, end, starts)
    ants = tf.where(ants == sink, source, ants)

    edges_to_enhance = tf.boolean_mask(
        tf.reshape(tf.transpose(tf.stack([t[:-1, :], t[1:, :]], axis=-1), [1, 0, 2]), [-1, 2]),
        tf.reshape(tf.transpose((tf.cumsum(tf.ones_like(t), axis=0) - 1)[1:, :] > s), [-1])
    )

    hashed_selected_edges = 10 * edges_to_enhance[:, 0] + edges_to_enhance[:, 1]
    updates = tf.reduce_sum(tf.cast(tf.reshape(hashed_selected_edges, [-1, 1]) == hashes, dtype=tf.int32), axis=0)

    proba *= tf.pow(1.05, tf.cast(updates, dtype=tf.float32))
    proba = tf.maximum(proba, 0.125 * eta)
    proba = proba / tf.gather(
        tf.math.reduce_sum(
            tf.cast(row == tf.reshape(tf.range(10), [-1, 1]), dtype=tf.float32) * proba[tf.newaxis, :], axis=-1),
        row)

    return proba, ants, history, starts, tf.reduce_sum(tf.ones_like(s))
```

Pętla zewnętrzna:

```
@tf.function(reduce_retracing=True, jit_compile=True)
def gpu_run(row, col, hashes, tau, epochs):
    with tf.GradientTape() as t:
        with t.stop_recording():
            ants = tf.zeros([9], dtype=tf.int32)
            starts = tf.zeros_like(ants, dtype=tf.int32)
            h_len = 20
            source = 0
            sink = 9
            history = tf.zeros([h_len, 9], dtype=tf.int32)
            history = tf.tensor_scatter_nd_update(history, tf.transpose(tf.stack([tf.fill([9], 0), tf.range(9)])), ants)

            eta = tf.ones_like(row, dtype=tf.float32)

            proba = tau * eta
            proba = proba / tf.gather(
                tf.math.reduce_sum(
                    tf.cast(row == tf.reshape(tf.range(10), [-1, 1]), dtype=tf.float32) * proba[tf.newaxis, :], axis=-1),
                row)

            step = tf.constant(0, dtype=tf.int32)
            while tf.less(step, epochs):
                proba, ants, history, starts, s = proceed_one_epoch(
                    row, col, hashes, ants, history, proba, starts, step % h_len, sink, source, h_len)
                step += 1

            return proba
```

Jak sobie radzi taka implementacja?

Testy 1:

- CPU i standardowe GPU
- 2000 epok
- 10 wierzchołków, 9 mrówek

	CPU (2000 epok)	GPU (2000 epok)
Tylko przejścia liczone wewnątrz tf.function	20.4354s	-
Epoki liczone wewnątrz tf.function	3.5072s	4.8191s
Wszystko liczone wewnątrz tf.function (bez modyfikatorów)	0.4914s	-
Wszystko liczone wewnątrz tf.function (zredukowany tracing i JIT compile (XLA))	0.1806s	0.3614s

W zasadzie można było się spodziewać, że przy tak małym grafie i takiej liczbie mrówek kod jest w większości sekwencyjny, więc użycie GPU nie daje przewagi. Trzeba by spróbować na większym problemie, ale nie wykonałem na ten moment takiego testu.

Test 2:

- CPU, standardowe GPU, high-end'owe GPU, TPU
- Pełna optymalizacja
- 150,000 epok
- 10 wierzchołków, 9 mrówek

	CPU	GPU	High-end GPU	TPU
Wszystko liczone wewnątrz tf.function (zredukowany tracing i JIT compile (XLA))	11.3856s	18.3746s	22.2890s	13.9803s

Dalsze kroki?

Dotychczasowe doświadczenia pokazały, że prawdopodobnie można spróbować zaimplementować nawet bardziej skomplikowane algorytmy, jak to było ustalone na spotkaniu projektowym.