Przedmiotem tego raportu jest implementacja równoległej wersji algorytmu b-adoratorów. Program wczytuje dane, mapuje wierzchołki grafu i zapisuje je (jako krawędzie) do tablicy sąsiadów. Następnie przygotowuje resztę struktur, sortuje (malejąco) krawędzie w tablicy sąsiadów każdego wierzchołka, a następnie same wierzchołki – względem najcięższej wychodzącej z niego krawędzi. Wtedy przystępuje do działania: dla każdego *metoda_b* z przedziału *[0, limit_b]* wczytuje wartości *b(v)* i wykonuje podany w zadaniu algorytm, po czym wypisuje wynik.

Zastosowane optymalizacje:

- użycie możliwie szybkich struktur danych: *unordered_map* zamiast *map*, *unordered_set* zamiast *set*, użycie *vectora* dla *rev_vertices_map* zamiast *map*, użycie działającego w czasie stałym *unique_integers_vector* zamiast *set*
- Posortowanie wierzchołków względem najcięższej wychodzącej z niego krawędzi
- Posortowanie krawędzi wychodzących z danego wierzchołka: jako, że program wykonuje obliczenia dla wielu różnych zestawów wartości *b(v)*, posortowanie wszystkich krawędzi może być nawet bardziej opłacalne niż częściowe posortowanie, wspomniane w pracy „KHAN, Arif, et al. Efficient approximation algorithms for weighted b-matching."
- Wykonywanie operacji: sortowania krawędzi, inicjalizowania wartości *b(v)*, liczenia wyniku i czyszczenia niektórych struktur danych równolegle – operacje te wydają się być kosztowne, a jednocześnie możliwe w łatwy sposób do zrównoleglenia
- Użycie *atomic_flag* zamiast *mutex*
- Trzymanie danych globalnie zamiast przekazywania ich jako argument funkcji (niestety trochę cierpi na tym jakość kodu)
- Podział operacji w strukturach na te potrzebujące i niepotrzebujące „mutexu" i odpowiednie korzystanie z nich – dzięki temu m.in. ilość podniesień „mutexu" jest mniejsza
- Wyeliminowanie operacji *T[y].erase(x)* m.in. poprzez wykorzystanie tablicy *db*, dzięki czemu struktura *T* nie potrzebuje „mutexu"
- Szybkie oddawanie „mutexów" w funkcji *suit* i przeniesienie podnoszenia „mutexu" struktury *R* poza część kodu, w której podniesiony jest inny „mutex" (a dokładniej *vertex_db_and_R_lock[y]*)
- Pamiętanie, dla danego wierzchołka, indeksu dla ostatnio wyszukanego *arg_max'a* i rozpoczynanie poszukiwania następnego już za nim (jest to możliwe dzięki wcześniej wspomnianemu posortowaniu krawędzi)
- Korzystanie z szybkich operacji przeniesienia, m.in. poprzez zastosowanie funkcji *swap* czy *emplace_back*

Dla zmodyfikowanej wersji grafu „Pennsylvania road network" (ze strony http://snap.stanford.edu/data/roadNet-PA.html), przy czasie wczytywania danych rzędu pół sekundy, blimit wynoszącym 30 i funkcji bvalue o następującej definicji:

```
unsigned int bvalue(unsigned int method, unsigned long node_id) {
    return (3*node_id + 5*method) % 11 + 1;
}
```

program generował wyniki w następujących czasach:

| Ilość wątków: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Czas (w sekundach): | 130,913 | 74,453 | 55,35 | 45,469 | 41,088 | 37,418 | 34,385 | 33,502 |
| Przyspieszenie: | 1,000 | 1,758 | 2,365 | 2,879 | 3,186 | 3,499 | 3,807 | 3,908 |

Testy zostały wykonane na komptuterze z procesorem czterordzeniowym (Intel(R) Core(TM) i7-4720HQ @ 2.60GHz) z włączoną funkcją Hyper-threadingu