Практическая работа
Алгоритм Fiduccia-Mattheysys

Ожерельев Игорь, гр. 718

Описание модификации

Структурное описание

Перед тем как начать описание модифицированного варианта алгоритма, приведу некоторые оптимизации, которые использовались в реализации базового алгоритма. Также следует уделить внимание структуре реализации.

В качестве основного подхода имплементации алгоритма был выбран ООП подход. Среди классов можно выделить следующие: **Partition**, **HyperGraph** и **GainContainer**.

В базовой версии регулярно требуется эффективно подсчитывать количество элементов гипер-ребра для каждой части разбиения. Для этого в **Partition** введены дополнительные массивы, хранящие количество вершин в левой и правой части для каждого ребра соответственно(**left_net_size**, **right_net_size**).

Описанные выше структуры данных позволяют эффективно определять количество вершин каждого ребра в нужной части за время порядка O(1), что в дальнейшем положительно скажется на времени работы как базового, так и улучшенного вариантов. Ниже приведен Листинг 1 с подробным описанием класса Partition.

Листинг 1: Partition

```
class Partition {
private:
    HyperGraph *graph;

    void init_vertices();
    void init_side_sizes();
public:
    char *vertices_part;
    std::uint32_t *left_net_sizes;
    std::uint32_t *right_net_sizes;

    std::size_t solution_cost = 0;
    std::int64_t balance = 0;
    std::size_t imbalance = 0;

    ~Partition();
    Partition(HyperGraph *graph, std::size_t
        imbalance);
    void rollback(const std::vector<std::
       uint32_t>& rollback_vex);
    std::uint32_t get_cost();
    void update(std::uint32_t vex_id);
    HyperGraph *get_graph();
    void store(const std::string& filename);
};
```

Модификация

Перейдем непосредственно к улучшеному варианту. В качестве нового похода была изменена логика функции **GainContainer.feasible_move()**, возвращающая наиболее оптимального кандидата среди вершин на перемещение между частями разбиения. Вспомним, что базовое поведение заключалось в возвращении первой вершины из **bucket** с максимальным значением **gain**. Новая функция **GainContainer.feasible_move_modified()** возвращает вершину не из начала, а из конца **bucket**. На результат работы на первый взгляд это может не повлиять и вовсе, но здесь результат работы упирается в текстовое представление графа, то есть зависимости от порядка записи вершин каждого ребра.

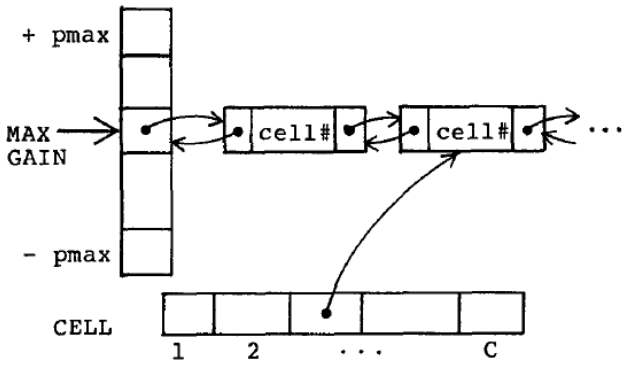

Рис. 1: Устройство контейнера

Сравнение результатов

Для сравнения выберем следующие характерные переменные: величину разреза, время работ и количество эпох алгоритма. Сравнивая результаты, приведенные в таблицах 1 и 2 ниже, можно отметить, что для бенчмарков **ibm16-17** величина разреза существенно улучшилась по сравнению с базовой реализацией, хотя время работы увеличилось из-за большего количества эпох. Также стоит обратить внимание на бенчмарки **dac2012**: для них по величине разреза выигрывает базовый вариант. Однако для **dac2012_12** наиболее выгодным по времени оказался модифицированный алгоритм, несмотря на небольшой проигрыш по величине разреза.

Вывод

Из сравнения можно заключить следующее: для **ibm16-17** результаты работы модифицированного алгоритма оказались лучше; для **dac2012** в целом хуже, чем для базового, кроме **dac2012_12**. При постановки задачи разбиения произвольного графа предпочтительнее использовать оба варианта алгоритма, так как заранее довольно трудно определить - какой способ подходит для успешного разбиения.

Также стоит отметить, что при тестировании предпочтение отдавалось величине разреза, а не балансировке. Поэтому в результатах наблюдается сильная несбалансированность между сторонами разбиения. В современных утилитах для разделения гипер-графов(hmetis, METIS и др.) для избежания этого недостатка применяются более сложные механизмы определения наилучшего кандидата на перемещение среди вершин.

Таблица 1: Результаты базового алгоритма

name	*cut_size*	*time, sec*	*epochs*
*ibm*01	265	0.905	9
*ibm*02	1094	0.994	6
*ibm*03	839	1.954	10
*ibm*04	1509	1.145	5
*ibm*05	2256	0.601	5
*ibm*06	2353	2.724	9
*ibm*07	3145	3.646	9
*ibm*08	1262	5.397	13
*ibm*09	2705	6.573	13
*ibm*10	4355	3.510	5
*ibm*11	3090	8.244	12
*ibm*12	4870	5.846	7
*ibm*13	5193	5.196	6
*ibm*14	8505	8.931	6
*ibm*15	4472	19.756	11
*ibm*16	11965	12.047	6
*ibm*17	12132	16.332	7
*ibm*18	2424	29.199	14
*dac*2012_2	8010	87.456	10
*dac*2012_3	8855	56.672	7
*dac*2012_6	10418	66.611	8
*dac*2012_7	8585	74.906	6
*dac*2012_9	1722	59.072	8
*dac*2012_11	5491	55.413	7
*dac*2012_12	10351	105.627	9
*dac*2012_14	2211	63.289	12
*dac*2012_16	4616	32.720	6
*dac*2012_19	3399	29.777	7

Таблица 2: Результаты модифицированного алгоритма

name	*cut_size*	*time, sec*	*epochs*
*ibm*01	386	0.589	6
*ibm*02	819	1.183	7
*ibm*03	1036	2.875	14
*ibm*04	1662	2.410	10
*ibm*05	1666	1.164	5
*ibm*06	1918	1.950	7
*ibm*07	3135	2.438	6
*ibm*08	2940	3.183	8
*ibm*09	2881	2.501	5
*ibm*10	4065	3.531	5
*ibm*11	3482	7.929	10
*ibm*12	4401	5.432	7
*ibm*13	4460	4.046	5
*ibm*14	7463	10.005	7
*ibm*15	3969	20.547	12
*ibm*16	5669	23.535	12
*ibm*17	7211	29.817	14
*ibm*18	3705	21.261	12
*dac*2012_2	13447	66.899	8
*dac*2012_3	8040	38.774	5
*dac*2012_6	6364	86.846	10
*dac*2012_7	15174	129.88	11
*dac*2012_9	10454	42.227	6
*dac*2012_11	9746	70.130	9
*dac*2012_12	10626	68.646	6
*dac*2012_14	2855	25.293	5
*dac*2012_16	7838	54.843	10
*dac*2012_19	5473	24.636	6