

Практическая работа

Алгоритм Fiduccia-Mattheysesys

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

ка 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_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.

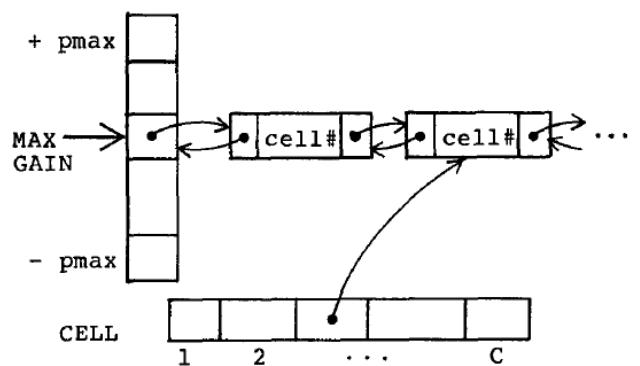


Рис. 1: Устройство контейнера

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 постановке 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 предположение отдавалось величине разреза, а не балансировке. Поэтому в результатах наблюдается сильная несбалансированность между сторонами разбиения. В современных утилитах 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

<i>name</i>	<i>cut_size</i>	<i>time, sec</i>	<i>epochs</i>
<i>ibm01</i>	265	0.905	9
<i>ibm02</i>	1094	0.994	6
<i>ibm03</i>	839	1.954	10
<i>ibm04</i>	1509	1.145	5
<i>ibm05</i>	2256	0.601	5
<i>ibm06</i>	2353	2.724	9
<i>ibm07</i>	3145	3.646	9
<i>ibm08</i>	1262	5.397	13
<i>ibm09</i>	2705	6.573	13
<i>ibm10</i>	4355	3.510	5
<i>ibm11</i>	3090	8.244	12
<i>ibm12</i>	4870	5.846	7
<i>ibm13</i>	5193	5.196	6
<i>ibm14</i>	8505	8.931	6
<i>ibm15</i>	4472	19.756	11
<i>ibm16</i>	11965	12.047	6
<i>ibm17</i>	12132	16.332	7
<i>ibm18</i>	2424	29.199	14
<i>dac2012_2</i>	8010	87.456	10
<i>dac2012_3</i>	8855	56.672	7
<i>dac2012_6</i>	10418	66.611	8
<i>dac2012_7</i>	8585	74.906	6
<i>dac2012_9</i>	1722	59.072	8
<i>dac2012_11</i>	5491	55.413	7
<i>dac2012_12</i>	10351	105.627	9
<i>dac2012_14</i>	2211	63.289	12
<i>dac2012_16</i>	4616	32.720	6
<i>dac2012_19</i>	3399	29.777	7

Таблица 2: Результаты модифицированного алгоритма

<i>name</i>	<i>cut_size</i>	<i>time, sec</i>	<i>epochs</i>
<i>ibm01</i>	386	0.589	6
<i>ibm02</i>	819	1.183	7
<i>ibm03</i>	1036	2.875	14
<i>ibm04</i>	1662	2.410	10
<i>ibm05</i>	1666	1.164	5
<i>ibm06</i>	1918	1.950	7
<i>ibm07</i>	3135	2.438	6
<i>ibm08</i>	2940	3.183	8
<i>ibm09</i>	2881	2.501	5
<i>ibm10</i>	4065	3.531	5
<i>ibm11</i>	3482	7.929	10
<i>ibm12</i>	4401	5.432	7
<i>ibm13</i>	4460	4.046	5
<i>ibm14</i>	7463	10.005	7
<i>ibm15</i>	3969	20.547	12
<i>ibm16</i>	5669	23.535	12
<i>ibm17</i>	7211	29.817	14
<i>ibm18</i>	3705	21.261	12
<i>dac2012_2</i>	13447	66.899	8
<i>dac2012_3</i>	8040	38.774	5
<i>dac2012_6</i>	6364	86.846	10
<i>dac2012_7</i>	15174	129.88	11
<i>dac2012_9</i>	10454	42.227	6
<i>dac2012_11</i>	9746	70.130	9
<i>dac2012_12</i>	10626	68.646	6
<i>dac2012_14</i>	2855	25.293	5
<i>dac2012_16</i>	7838	54.843	10
<i>dac2012_19</i>	5473	24.636	6