

Практическая работа

Алгоритм Fiduccia-Mattheyses

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 логи-

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.

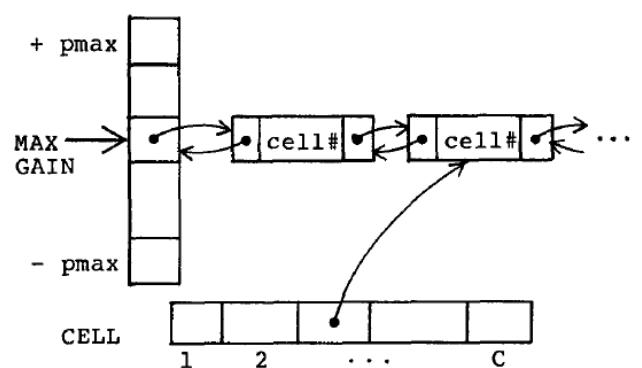


Рис. 1: Устройство контейнера

Сравнение результатов

Для сравнения выберем следующие характерные переменные: величину разреза, время работы и количество эпох алгоритма. Время, приведенное в таблицах - время работы самого алгоритма, без учета на чтение и запись файлов. Сравнивая результаты, представленные в таблицах 1 и 2 ниже, можно отметить, что влияние модификации в целом имеет негативный характер как по величине разреза, так и по количеству эпох, а следовательно времени.

Таблица 1: Результаты базового алгоритма

<i>name</i>	<i>cut_size</i>	<i>time, sec</i>	<i>epochs</i>
<i>ibm01</i>	384	0.959	16
<i>ibm02</i>	943	0.891	10
<i>ibm03</i>	1514	3.358	28
<i>ibm04</i>	2113	1.734	13
<i>ibm05</i>	3410	4.722	38
<i>ibm06</i>	1437	2.19	15
<i>ibm07</i>	2232	3.758	17
<i>ibm08</i>	1662	6.654	27
<i>ibm09</i>	4560	6.112	22
<i>ibm10</i>	2703	5.729	15
<i>ibm11</i>	7277	9.271	25
<i>ibm12</i>	4174	7.478	19
<i>ibm13</i>	2861	7.424	16
<i>ibm14</i>	7929	21.172	28
<i>ibm15</i>	8299	12.399	13
<i>ibm16</i>	5419	27.216	26
<i>ibm17</i>	5610	20.124	18
<i>ibm18</i>	3566	36.795	36
<i>dac2012_2</i>	32863	110	21
<i>dac2012_3</i>	20583	48.698	12
<i>dac2012_6</i>	21035	59.055	13
<i>dac2012_7</i>	56707	95.606	15
<i>dac2012_9</i>	13859	79.794	17
<i>dac2012_11</i>	15021	90.139	18
<i>dac2012_12</i>	29125	120.459	20
<i>dac2012_14</i>	14756	62.359	21
<i>dac2012_16</i>	24852	79.794	26
<i>dac2012_19</i>	10582	38.790	17

Таблица 2: Результаты модифицированного алгоритма

<i>name</i>	<i>cut_size</i>	<i>time, sec</i>	<i>epochs</i>
<i>ibm01</i>	2121	0.755	14
<i>ibm02</i>	1271	0.960	11
<i>ibm03</i>	4430	2.488	24
<i>ibm04</i>	5207	3.318	25
<i>ibm05</i>	6998	2.389	20
<i>ibm06</i>	5816	3.693	27
<i>ibm07</i>	8311	5.235	25
<i>ibm08</i>	9256	5.412	25
<i>ibm09</i>	10211	7.239	27
<i>ibm10</i>	13242	15.162	41
<i>ibm11</i>	14547	17.596	47
<i>ibm12</i>	15638	7.444	20
<i>ibm13</i>	17461	7.633	17
<i>ibm14</i>	24620	36.313	48
<i>ibm15</i>	32833	20.415	22
<i>ibm16</i>	36520	26.082	27
<i>ibm17</i>	41673	77.802	79
<i>ibm18</i>	35252	32.739	35
<i>dac2012_2</i>	49026	44.763	10
<i>dac2012_3</i>	22145	40.376	10
<i>dac2012_6</i>	22160	77.471	17
<i>dac2012_7</i>	57976	135.228	20
<i>dac2012_9</i>	30606	32.801	8
<i>dac2012_11</i>	25755	93.541	23
<i>dac2012_12</i>	37898	234.257	33
<i>dac2012_14</i>	23900	85.849	31
<i>dac2012_16</i>	34152	38.405	12
<i>dac2012_19</i>	19384	36.858	16