Fiduccia-Mattheyses algorithm. Implementation and modification

Ivan Sladkov, B01-908, MIPT

April 24, 2023

Being an heuristic for solution of a problem of graph partition, Fiduccia-Mattheyses algorithm does not give a completely precise solution, though resulting partition converges to it. Some modifications may be introduced in order to increase convergence speed or precision of partition. In this work original algorithm was implemented and a modification was added to increase speed of algorithm.

1 On algorithm implementation

The original algorithm was implemented with no clustering and with accent on balance. For a formula

$$rW - S_{max} \leq |A| \leq rW + S_{max} \qquad (1)$$

from , which describes soft balancing, coefficient r is taken as 0.5, while maximum divergence between $|A|$ and $|B|$ is $S_{max} = 1$.

Implementation was made in C++. All the containers used in project are based on Standard Template Library. The data structures used are as follows:

- `std::map<int, std::list<int> > gc` – gain bucket

- `std::vector<bool> erased` – shows whether vertex is in bucket or not

- `std::vector<std::pair<int, iterator> > searchSupport` – structure to facilitate and accelerate search

Search through gain buckets is made as $O(1)$.

2 Modification of algorithm

In order to make algorithm converge faster a «cutoff» modification was implemented: if movement of a vertex gives significant cost increase, then further motion is not done. The original idea was to avoid applying of vertices movement, whose gain is negative. Then the threshold was added. So function `FMPass()` is transformed into:

```
function FMpass
(gain_container, partitionment) :

solution_cost =
partitionment.get cost()

while not all vertices locked {

move = best_feasible_move()

solution_cost-=
gain_container.get gain (move)

if (solution_cost > best_cost+threshold)
then break

gain_container.lock_vertex(
move.vertex())

gain_update (move)

partitionment.apply (move)
}

roll back partitionment
to best seen solution

gain_container.unlock_all()
```

3   Comparison

All benchmarks were executed on Apple M2 processor. Result of comparison is shown in table 1.

The reason for introducing this optimization was to check whether reduction of vertices, that are moved in FMPass(), leads to reduction of execution time. It can be seen that such an approach in most cases gives much higher convergence speed; however partition cost gets higher. It can also be seen that relative growth of partition cost w. r. t. edges number is greater than reduction of elapsed time.

As a conclusion, this modification, while improving overall performance, results in a high partition cost, which may be unacceptable in some applications.

References

[1] C.M. Fiduccia and R.M. Mattheyses *A Linear-Time Heuristic for Improving Network Partitions*, General Electric Research and Development Center.

File	Base		Modified		Base/Modified Time ratio	Modified/Base Cost ratio
	Time elapsed	Partition cost	Time elapsed	Partition cost		
ISPD98_ibm01.hgr	2311	358	246	1598	9.39	4.46
ISPD98_ibm02.hgr	2658	520	371	512	7.16	0.98
ISPD98_ibm03.hgr	2824	3909	429	4148	6.58	1.06
ISPD98_ibm04.hgr	5345	1882	308	4096	17.35	2.18
ISPD98_ibm05.hgr	8940	3589	709	6086	12.61	1.70
ISPD98_ibm06.hgr	4653	1724	751	4799	6.20	2.78
ISPD98_ibm07.hgr	7191	3018	1299	7274	5.54	2.41
ISPD98_ibm08.hgr	9761	1382	1360	8776	7.18	6.35
ISPD98_ibm09.hgr	11944	3873	1995	9556	5.99	2.47
ISPD98_ibm10.hgr	12128	3008	1637	13001	7.41	4.32
ISPD98_ibm11.hgr	10075	8693	2313	13463	4.36	1.55
ISPD98_ibm12.hgr	20845	4622	2959	14155	7.04	3.06
ISPD98_ibm13.hgr	17317	3222	3225	16767	5.37	5.20
ISPD98_ibm14.hgr	51533	9833	6173	22643	8.35	2.30
ISPD98_ibm15.hgr	41726	7899	6496	30221	6.42	3.83
ISPD98_ibm16.hgr	39586	5539	12863	32235	3.08	5.82
ISPD98_ibm17.hgr	51904	6354	11020	40035	4.71	6.30
ISPD98_ibm18.hgr	64940	3483	7323	32570	8.87	9.35
					7.42	*3.67*

Table 1: Results of research