Heuristic Graph Bisection with Less Restrictive Balance Constraints *

Stefan Schamberger

Fakultät für Elektrotechnik, Informatik und Mathematik
Universität Paderborn
Fürstenallee 11, D-33102 Paderborn
email: schaum@uni-paderborn.de

Abstract

Fast graph partitioning is an important subproblem in many applications. While the classical problem asks all partitions to be of almost the same size, there are some applications that do not need or even do not want such restrictive constraints. This paper shows how the Helpful-Set heuristic implemented in the graph partitioning library Party can be adopted to the less restrictive case.

1. Introduction

The graph partitioning problem is well known and occurs as a subproblem in many important applications. Given a graph, the task is to assign a graph's vertices to one of the equal sized partitions, such that the number of edges connecting vertices of different partitions is minimized. This paper is restricted to the 2-partitioning problem.

Most applications require almost evenly sized partitions. One typical example is solving the load distribution problem occurring during parallel Finite Element Method calculations. However, recently a new approach for on-line routing and data management has been proposed [1]. A subproblem herein is the assembling of a so-called decomposition tree, whose construction requires the computation of a 1/6 min-cut of a graph. If we define an imbalance allowance of i%, $0 \leq i \leq 100$ as being an upper bound to a partition's weight, that may not exceed $\lceil |V|/2 \cdot (100+i)/100 \rceil$, this problem corresponds to finding a 2-partitioning with 66% imbalance allowance. Since is thinkable that the techniques introduced in [1] are as well applicable solving other problems, we assume that there is some demand for finding 2-partitionings with less restrictive balance constraints.

In this paper we demonstrate that the Helpful-Set graph partitioning heuristic can be modified to also work well with less restrictive balancing constraints and present a comparison between the results obtained using Party and two other state-of-the-art partitioning libraries, Jostle and Metis.

The remaining part of this paper is organized as follows. In the next section we briefly present the necessary changes to the original implementation. Section 3 describes our method of evaluating different partitioning libraries, which is then applied in Section 4 comparing results obtained by Jostle, Metis and Party. In Section 5 we give a short conclusion.

*This work was partly supported by the German Science Foundation (DFG) project SFB-376 and by the EU IST Program under contract number IST-1999-14186 (ALCOM-FT).

2. Helpful-Set Modifications

Since the graph partitioning problem often only represents a subproblem, it has to be solved fast and as space-efficient as possible. Due to the size of the graphs, state-of-the-art graph partitioning libraries like Metis [2], Jostle [3], Chaco [4] or Party [5] usually follow the multilevel scheme [4]. Vertices of the graph are contracted and a new level consisting of a smaller graph with a similar structure is generated. This is repeated, until in the lowest level only a small graph, sometimes only with 2 vertices, remains. The partitioning problem is then solved for this small graph and vertices in higher levels are assigned to partitions according to their representatives in lower levels, after a local refinement phase has been applied to further enhance the current solution. This process finally leads to a partitioning of the original graph. Hence, a multilevel algorithm consists of three important tasks: A matching algorithm, deciding which vertices are combined in the next level, a global partitioning algorithm applied in the lowest level (which actually can be omitted if the number of vertices in the lowest level meets the number of desired partitions), and a local refinement algorithm improving the quality of a given partitioning.

In most cases, this important refinement process is based on the Fiduccia-Mattheyses method [6], a run-time optimized version of the Kerninghan-Lin (KL) algorithm [7]. The Helpful-Set (HS) heuristic for local refinement is based on theoretic observations used to find upper bounds for the bisection width of regular graphs [8]. It has been implemented and presented in [5] and serves in the Party library as an alternative to Kerninghan-Lin style heuristics. Just as KL, the Helpful-Set heuristic is based on local search. Beginning with a given initial bisection π, it tries to reduce the edge cut with the help of local rearrangements. However, their choice is the main difference to KL since it does not only migrate single vertices but sets of vertices. The algorithm starts to search for l-helpful sets, that is a subset of nodes from either side decreasing the edge cut by l if moved to the other partition. If such a set is found, it is moved and the algorithm then tries to find a balancing set

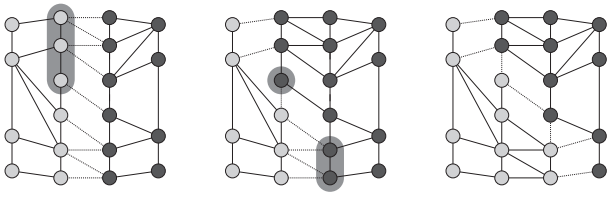

Figure 1. Moving the marked set (left) reduces the current edge cut of 9 by 2 (thus the set is called 2-helpful), but it destroys the balance by 3. However, by moving the second marked set (middle), this can be fixed (thus this set is called a balancing set), and a balanced solution (right) is found leading to an overall edge cut reduction of 2.

that eliminates the caused imbalance but also does not increase the edge cut by more than $l-1$. If such a set can be found it is moved, resulting in an edge cut reduction of at least 1, and the whole process is repeated until no more improvements can be made. Figure 1 shows an example of a successful round reducing the edge cut from 9 to 7 by moving a 2-helpful set.

Two modifications are required to adopt the original Helpful-Set algorithm to the new situation. The resulting algorithm is shown in Figure 2. First, the weight of the balancing set does not have to match the weight of the helpful set exactly (line 3). Instead, it has just to ensure that the weight constraints are fulfilled. Thus, it can happen that although a helpful set is moved during step 1, no balancing set is needed since the weights are still within their limits. This makes finding improvements easier. Second, to make the algorithm more reliable, an explicit balancing step is appended at the end (line 8), ensuring that balancing also occurs if no helpful set could be found at all.

```
1 while the search for a helpful set S is successful (S ≠ ∅)
2   move S to other partition
3   search for balancing set S̄ with helpfulness at least 1 − H(S)
4   if S̄ ≠ ∅
5     move S̄ to other partition
6   else
7     move S back
8 enforce balance constraints by moving vertices greedily
```

Figure 2. Sketch of the Helpful-Set 2-partitioning Algorithm.

As mentioned, the Helpful-Set algorithm is applied as a local improvement method in every level of the multilevel method. Since many vertices are matched, large vertex weights occur in lower levels of this process. In this case it becomes more difficult to balance the partitions and we therefore introduced a grace variable. This variable determines how much additional imbalance is allowed per level and it is set to 0 in the highest level and to twice the weight of the heaviest vertex in lower levels, introducing more flexibility.

3. Evaluation

We have tested the described algorithm on a large amount of different graphs (we actually used many more graphs than presented in this paper), including the graphs that occur executing the implementation of the routing algorithm from [1]. Unfortunately, the set of the latter is very small, contains only very small instances (ca. 20 – 100 vertices) and thus is not suitable to compare the heuristics. Therefore, we used a set of well known FEM graphs for our comparisons. Most of these graphs have also been used in [3], comparing Jostle and Metis. Since we do not have access to the graphs 'mesh1m', 'oliker' and 'bmw1c', we replaced these two graphs by the 'biplane9', 'grid100', 'stufe10' and 'wave' graphs. Table 1 gives an overview of the test-set and some of the graphs' properties. Most of these graphs are FEM graphs, 2-dimensional as well as 3-dimensional, while some others represent the structure of sparse matrices from different sources. All experiments have been performed on a Pentium III 933 MHz dual processor system with 1 GB of main memory.

Both libraries, Jostle (version 2.3) and Metis (version 4.0.1), use a default imbalance allowance of 3%. While Jostle provides an option to override this setting, both versions of Metis do not offer this choice. However, the variable 'ubfactor' specifies the amount of allowed imbalance internally, and since the source code of Metis is available we were able access this parameter. Nevertheless, the results obtained should be taken with care, since the authors probably did not design the package to work with high imbalance settings.

Using a fixed set of graphs to test a partitioning library includes some drawbacks. Even if not intended, it is easy to adopt an algorithm to the test set, meaning that the results become better for the selected graphs but deteriorate for others. During the development of the Helpful-Set heuristic, we have experienced this several times. To overcome this problem, we first enlarged the test set (we actually used many more graphs than presented in this paper). This makes the undesired adoption more unlikely, but also increases the testing time a lot. As also presented in [9], another approach we found is based on permutation. It should be mentioned, that generating random graphs is no solution to the problem, since their structure is completely different and much more

Table 1. Graphs used in this paper and some of their properties.

Graph	$\|V\|$	$\|E\|$	av. deg.	$\emptyset$	origin
144	144649	1074393	14.86	35	FEM 3D
4elt	15606	45878	5.88	67	FEM 2D
biplane.9	21701	42038	3.87	102	FEM 2D
crack	10240	30380	5.93	84	FEM 2D
dime20	224843	336024	2.99	1105	FEM 2D dual
grid100	10000	19800	3.96	198	FEM 2D
m14b	214765	1679018	15.64	33	FEM 3D
memplus	17758	54196	6.10	10	dig. circuit
ocean	143437	409593	5.71	229	FEM 3D dual
stufe.10	24010	46414	3.87	54	FEM 2D
t60k	60005	89440	2.98	495	FEM 2D dual
vibrobox	12328	165250	26.81	9	vibroacoustic m.
wave	156317	1059331	13.55	38	FEM 3D

predictable than those of 'real world' problems. However, a graph can be partly randomized keeping its structure. This is done by permuting its vertices.

Figure 3 shows an example of how this permutation works. First, a random permutation σ of numbers 0 to $|V|-1$ is determined. σ is then used to map each vertex v of the original graph G to vertex $\sigma(v)$ in the new graph G'. Next, the newly generated sequence of G's vertex numbers is sorted, changing the vertices' order. After that, the edges are adopted to the new graph by transforming an edge (v_i, v_j) into edge $(v_{\sigma(i)}, v_{\sigma(j)})$. Last, the outgoing edges of each vertex are sorted according to their destination's vertex number. It is obvious that the remaining graph G' has exactly the same structure and properties than the original G. However, in our experiments it is shown that the influence of the permutations on the 2-partitioning results is surprisingly high.

For each graph from our test set, we performed 100 runs. The first run consists in partitioning the original, unchanged graph, while for all following runs the graphs vertices are permuted. We are aware that this scheme does not solve the evaluation problem completely since an algorithm can still be tailored for special graph types. But at least it makes this more difficult for single test graphs that now represent a class of graphs. Last but not least, the method delivers some data about the variation of the solutions' quality. One can now determine if an algorithm does always find solutions of about the same quality or if they highly differ to each other, supplying a measurement for reliability. Thus we believe that the permutation makes comparisons more meaningful.

All results obtained in the experiments are displayed in a chart. Figure 4 gives an example obtained 2-partitioning the 100x100 grid, which we have chosen because optimal solutions (with edge cut 100) are known. The left part shows the bisection quality. The edge cut is shown on the x-axis while the y-axis displays the size of the largest partition (pw_{max}), that is the balance. Every mark of each type represents the result of one of 100 runs applying one of the libraries, respectively. Furthermore, the first run with the genuine, unchanged graph is represented by a solid mark, while the average of all 100 runs is displayed by a large solid mark. The right part shows the necessary resources. Since for the same graph these do not differ much in each run, only the

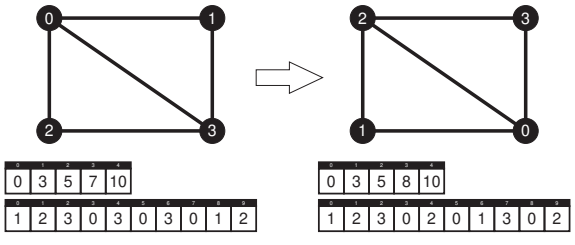

Figure 3. Permuting a graph's vertices. The original graph and its data structure (left) is transformed by the permutation $\sigma = (2,3,1,0)$ into the new graph (right).

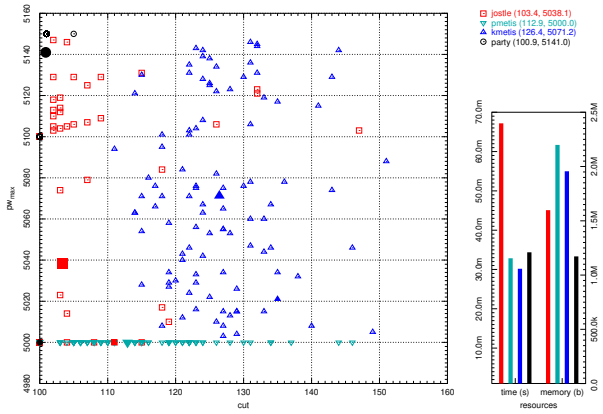

Figure 4. 2-partitioning a 100x100 grid with 3% imbalance

average time and memory consumption from all 100 runs is displayed.

In anticipation of Section 4, we now describe how the chart shown in Figure 4 can be interpreted. The red squares display results obtained using Jostle. The average solution for 3% allowed imbalance is of a good quality (103.4), and the figure reveals that there is some variation due to the randomization. This variation is even higher for Metis, which results are displayed with the upper light and lower dark blue triangles for pmetis and kmetis, respectively. Especially for kmetis achieving an average edge cut of 126.4 it is very high, both for the edge cut and for the balance. In this case, Party (black circles) does compute less diverge solutions, reaching an average edge cut of 100.9. On the right hand side the resources consumed by the different libraries are displayed. Since here the variation has shown to be small, only the average value is presented. In case of the 100x100 grid, Jostle needs longest to compute its results, followed by Party and Metis, while Metis needs the largest amount of memory, followed by Jostle and Party.

Another important point that can be observed is that if the libraries had been compared with only using the original graph, their rating would have been different. Metis (kmetis and pmetis) and Jostle would have performed worse while Party would have come off better, resulting in different conclusions.

4. Results

This section presents the results obtained applying different graph partitioning libraries. The libraries included are Jostle and Metis, where the latter one provides two different versions.

It is obvious that by loosen the balance restrictions of the graph bisection problem, partitionings with less cut edges become possible. However, this decrease is often not continuous. Looking at the 100x100 grid for example, the optimal bisection width is 100. This is still the case, if slight imbalances are allowed. Only if one partition may contain

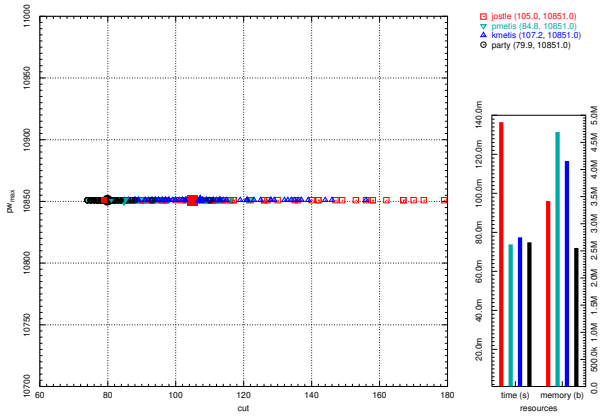

Figure 5. 2-partitioning 'biplane9' allowing no imbalance

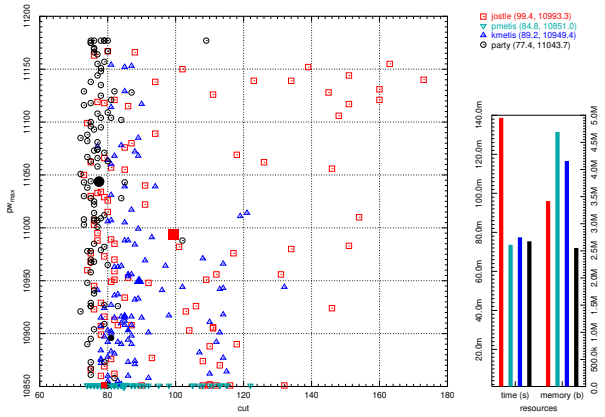

Figure 6. 2-partitioning 'biplane9' allowing 3% imbalance

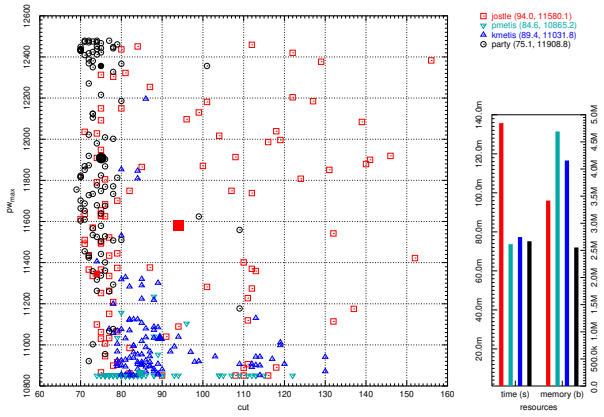

Figure 7. 2-partitioning 'biplane9' allowing 15% imbalance

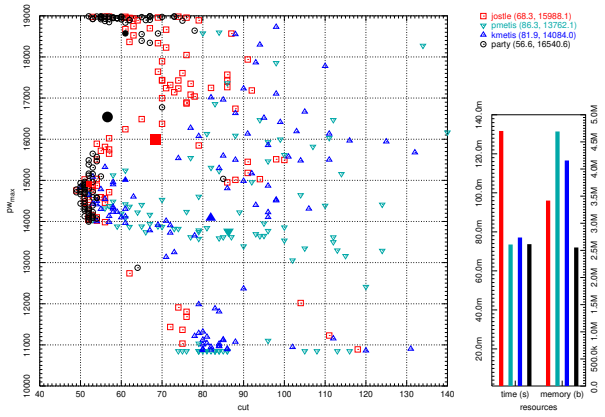

Figure 8. 2-partitioning 'biplane9' allowing 75% imbalance

$49 \cdot 50 = 2450$ vertices or less (which corresponds to more than 50% imbalance), the edge cut can be decreased to 99 or less. Furthermore, there can be some local minima, meaning that increasing as well decreasing a certain imbalance value leads to better results. For example, this fact can be observed in case of the 'biplane9' graph described next.

Figures 5 through 8 show the results obtained. If no imbalance is allowed, (Figure 5), Party and pmetis find the best partitionings while Jostle and kmetis determine worse solutions. For 3% imbalance allowance, which is the default parameter for Metis and Jostle, the results are given in Figure 6. Party, Jostle and kmetis utilize the allowed imbalance, while pmetis still only computes perfectly balanced partitionings. In this figure it becomes obvious again that the influence of the vertex permutations is quite large. Metis and especially Jostle produce some bad solutions resulting in a larger average edge cut. The same can be observed if 15% imbalance is allowed (Figure 7). While Metis tends to produce rather balanced solutions, Party does the opposite and often computes partitionings close to the maximum imbalance allowance. Jostle finds some good solutions, but is again highly affected by the permutation. Figure 8 contains the results for 75% imbalance allowance. Party and Jostle find good solutions while Metis often cannot make use of the weaker balance constraints*. Furthermore, one can observe that for a maximum partition weight of around 17500 (which equals an imbalance allowance of about 60%) the solutions found have a larger edge cut than those above or below this value. This indicates that due to the structure of the 'biplane9' not better solution exists in this region, meaning the existence of local minima.

From Figures 5 through 8 one can also see that in case of the 'biplane9' graph Metis and Party need about the same time to calculate the solutions while Jostle consumes almost twice this amount. Furthermore, it is interesting to see that the results and conclusions would have been different for

*This could be due to the design of Metis, since the imbalance feature was not implemented by the authors themselves, but added by us. Especially for pmetis the imbalance allowance has to be very large to prevent the calculation of perfectly balanced partitionings. Hence, the results computed by Metis with other than 0% (pmetis) and 3% (kmetis) imbalance allowance should be interpreted with care.

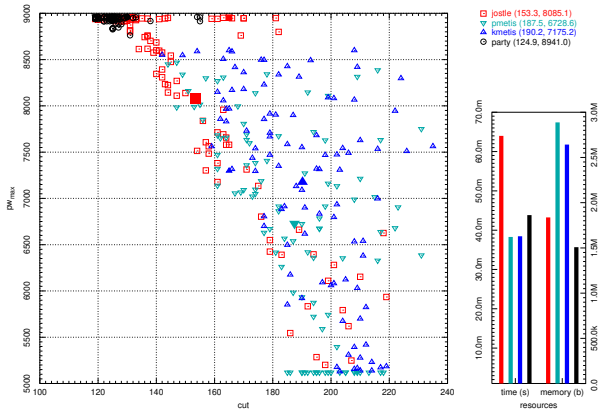

Figure 9. 2-partitioning 'crack' allowing 75% imbalance

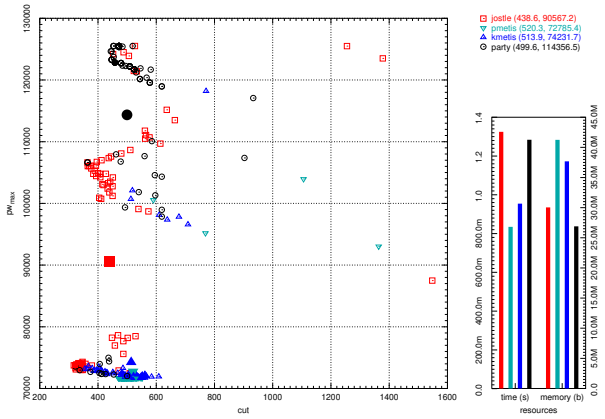

Figure 10. 2-partitioning 'ocean' allowing 75% imbalance

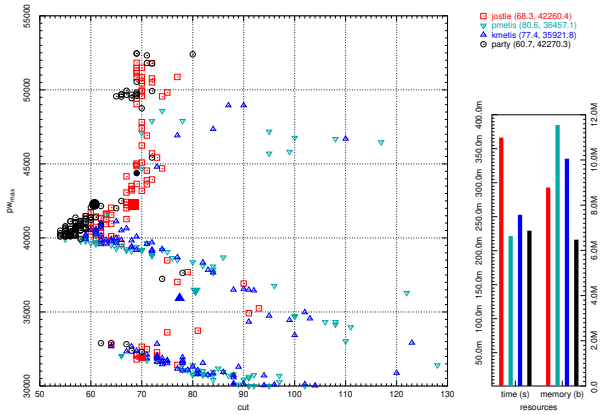

Figure 11. 2-partitioning 't60k' allowing 75% imbalance

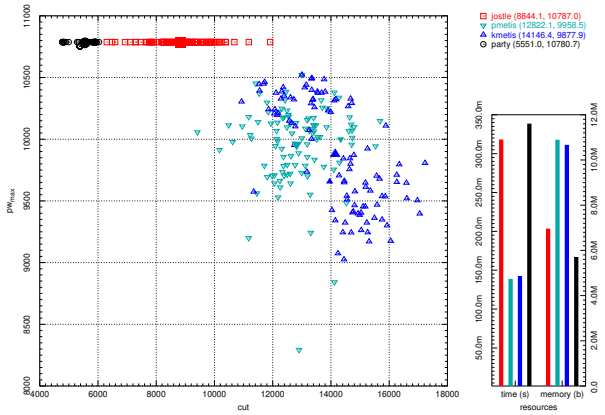

Figure 12. 2-partitioning 'vibrobox' allowing 75% imbalance

all imbalance settings if permutation was excluded and only the genuine graph had been considered for evaluation.

Figures 9, 10, 11 and 12 present four more charts obtained with an imbalance allowance of 75% for the graphs 'crack', 'ocean' 't60k' and 'vibrobox', respectively. Figure 10 reveals that the 'ocean' graph, in contrast to the 'crack' graph from Figure 9, also has some imbalance regions that prevent small edge cuts. Unfortunately, in case of this graph Party is not able to evade one of these local minima and thus produces much worse edge cuts than in case of 15% imbalance allowance. While Metis only produces solutions without large imbalance and therefore benefits of the fact that good solutions exist there, Jostle finds the best average edge cut. The 't60k' graph contains local minima as well, and in this case also Party is able to avoid them. In Figure 12 the results for the 'vibrobox' graph are shown. Here, Party and Jostle need more that twice as much time as Metis, but the chart shows that this pays off and much better solutions, down to 1/3 of the edge cut that Metis computed, can be found.

Due to space limitation, we present a comprehensive overview to all other results in Table 2. It shows the average values obtained for all graphs listed in Table 1 for 0%, 3%, 15% and 75% imbalance allowance, respectively. One can see that the time and memory requirements of all three libraries do not depend much on the imbalance settings. While usually Metis (pmetis or kmetis) is fastest, often followed by Party being faster than Jostle, Party always consumes the smallest and Metis the largest amount of memory.

Looking at the average edge cut, it usually decreases for all libraries the higher the allowed imbalance is. As already mentioned before, there are a few exceptions to this rule. The most obvious one is the increase of the edgecut computed by Party for the 'ocean' graph. Although a good solution of 361.9 can be found with less than 3% imbalance, only an average of 499.6 cut edges can be achived with the less tight restrictions of 75% imbalance allowance. As mentioned, we think that this can be explained with the strong increase of feasible solutions and the emerging of local minima, that makes the search and therefore finding the best solution more complicated. Nevertheless, apart from

Table 2. Bisection results obtained for the chosen set of graphs with an imbalance allowance of 0%, 3%, 15% and 75%, respectively. Each value represents the average of 100 runs. The best (smallest) value in each of the categories edge cut, time (sec) and memory (kb) are printed bold, while results not within 10% (edge cut) and 25% (time and memory) of this value are displayed gray.

Graph	Jostle			pmetis			kmetis			Party		
	edge cut	time	memory	edge cut	time	memory	edge cut	time	memory	edge cut	time	memory
no imbalance allowed												
144	7047.8	2.257	47072.7	6853.2	**1.319**	73596.3	7711.6	1.519	70080.5	**6730.2**	2.000	**38744.7**
4elt	169.9	0.112	2721.4	161.9	0.060	3969.8	217.6	0.061	3596.9	**148.6**	**0.060**	**1855.9**
biplane9	105.0	0.136	3338.8	84.8	**0.074**	4581.5	107.2	0.077	4060.0	**79.9**	0.075	**2494.1**
crack	205.8	0.064	1794.5	200.1	0.039	2820.0	263.3	**0.039**	2580.5	**192.5**	0.046	**1473.1**
dime20	243.3	1.491	31531.6	191.1	**1.007**	41622.7	215.4	1.161	36157.1	**185.8**	1.285	**23215.9**
grid100	117.0	0.058	1560.4	113.3	0.033	2147.2	155.9	**0.030**	1910.7	**100.2**	0.034	**1144.2**
memplus	6183.7	0.355	5887.4	6587.3	**0.118**	6739.4	6802.6	0.124	6535.7	**5791.4**	0.372	**5774.8**
m14b	4221.7	3.352	72498.3	4058.7	**2.157**	114897.6	4552.8	2.435	109672.5	**3956.8**	3.602	**61310.7**
ocean	564.4	1.359	29339.7	506.4	**0.836**	40283.4	751.2	0.962	36795.2	**496.1**	1.262	**26266.7**
stufe10	64.8	0.163	3712.4	59.4	0.091	5084.7	81.5	0.096	4507.3	**52.0**	**0.083**	**2780.9**
t60k	101.4	0.382	8577.2	97.0	**0.223**	11276.4	113.2	0.253	9820.6	**91.9**	0.234	**6325.5**
vibrobox	11802.8	0.416	6800.3	11799.4	0.157	10640.1	18279.4	**0.145**	10419.6	**11711.7**	0.287	**5576.5**
wave	9404.2	2.402	48204.8	9372.1	**1.367**	74861.3	10404.0	1.565	71069.1	**9104.9**	2.234	**40790.6**
allowed imbalance = 3%												
144	7030.4	2.183	47072.7	6855.9	**1.314**	73596.3	6989.6	1.513	70080.5	**6814.1**	2.318	**38744.7**
4elt	162.8	0.116	2721.4	160.5	**0.060**	3969.8	169.0	0.061	3596.9	**149.4**	0.061	**1855.9**
biplane9	99.4	0.139	3338.8	84.8	**0.074**	4581.5	89.2	0.077	4060.0	**77.4**	0.075	**2494.1**
crack	201.4	0.069	1794.5	199.8	0.039	2820.0	210.8	**0.039**	2580.5	**191.3**	0.046	**1473.1**
dime20	224.3	1.452	31531.6	192.0	**1.006**	41622.7	194.1	1.159	36157.1	**190.0**	1.287	**23215.9**
grid100	103.4	0.067	1560.4	112.9	0.033	2147.2	126.4	**0.030**	1910.7	**100.9**	0.034	**1144.2**
memplus	6092.3	0.327	5887.4	6555.6	**0.118**	6739.4	6583.8	0.125	6535.7	**5642.3**	0.374	**5774.8**
m14b	4203.5	3.294	72498.3	**4056.9**	**2.158**	114897.6	4145.0	2.433	109672.5	4081.8	3.810	**61310.7**
ocean	438.6	1.329	29339.7	504.5	**0.838**	40283.4	512.9	0.960	36795.2	**391.6**	1.277	**26266.7**
stufe10	62.5	0.160	3712.4	59.0	0.089	5084.7	61.7	0.095	4507.3	**51.0**	**0.083**	**2780.9**
t60k	92.8	0.370	8577.2	96.5	**0.225**	11276.4	105.5	0.253	9820.6	**79.4**	0.235	**6325.5**
vibrobox	11821.1	0.391	6800.3	11792.4	0.158	10640.1	11857.4	**0.148**	10419.6	**11730.0**	0.433	**5576.5**
wave	9410.1	2.274	48204.8	9381.0	**1.367**	74861.3	9441.4	1.551	71069.1	**9049.6**	2.471	**40790.6**
allowed imbalance = 15%												
144	6990.1	2.174	47072.7	**6850.8**	**1.313**	73596.3	6991.2	1.521	70080.5	6987.8	2.561	**38744.7**
4elt	159.3	0.114	2721.4	162.5	**0.059**	3969.8	167.5	0.061	3596.9	**150.8**	0.060	**1855.9**
biplane9	94.0	0.136	3338.8	84.6	**0.074**	4581.5	89.4	0.077	4060.0	**75.1**	0.075	**2494.1**
crack	199.1	0.066	1794.5	200.2	**0.039**	2820.0	209.2	0.039	2580.5	**189.5**	0.047	**1473.1**
dime20	210.7	1.448	31531.6	195.0	**1.010**	41622.7	193.4	1.154	36157.1	**160.9**	1.285	**23215.9**
grid100	**100.9**	0.071	1560.4	112.7	0.033	2147.2	126.1	**0.031**	1910.7	101.0	0.035	**1144.2**
memplus	5796.3	0.310	5887.4	6569.2	**0.118**	6739.4	6020.6	0.125	6535.7	**5000.2**	0.311	**5774.8**
m14b	4159.7	3.286	72498.3	**4077.4**	**2.157**	114897.6	4107.8	2.389	109672.5	4186.8	3.866	**61310.7**
ocean	422.4	1.324	29339.7	504.0	**0.835**	40283.4	503.2	0.957	36795.2	**421.7**	1.281	**26266.7**
stufe10	58.3	0.149	3712.4	62.0	0.085	5084.7	60.6	0.091	4507.3	**45.2**	**0.082**	**2780.9**
t60k	78.5	0.367	8577.2	95.3	**0.222**	11276.4	89.8	0.253	9820.6	**68.1**	0.231	**6325.5**
vibrobox	**11681.7**	0.371	6800.3	11800.6	0.157	10640.1	11796.1	**0.147**	10419.6	11695.6	0.551	**5576.5**
wave	9410.1	2.271	48204.8	9373.6	**1.368**	74861.3	9456.3	1.550	71069.1	**9363.0**	2.709	**40790.6**
allowed imbalance = 75%												
144	5694.9	2.154	47072.7	8049.1	**1.316**	73596.3	7710.1	1.512	70080.5	**5002.4**	2.319	**38744.7**
4elt	133.9	0.111	2721.4	167.1	**0.057**	3969.8	165.4	0.059	3596.9	**122.1**	0.058	**1855.9**
biplane9	68.3	0.132	3338.8	86.3	**0.073**	4581.5	81.9	0.077	4060.0	**56.6**	0.074	**2494.1**
crack	153.3	0.064	1794.5	187.5	**0.038**	2820.0	190.2	0.038	2580.5	**124.9**	0.044	**1473.1**
dime20	53.1	1.432	31531.6	134.4	**1.004**	41622.7	119.3	1.159	36157.1	**28.7**	1.265	**23215.9**
grid100	81.0	0.064	1561.8	117.5	0.032	2147.2	118.6	**0.030**	1910.7	**74.2**	0.032	**1144.2**
memplus	1960.2	0.263	5887.4	3519.7	**0.110**	6739.4	3600.0	0.118	6535.7	**1561.0**	0.192	**5774.8**
m14b	4063.7	3.286	72498.3	4333.9	**2.154**	114897.6	4316.3	2.417	109672.5	**3709.1**	3.832	**61310.7**
ocean	**438.6**	1.325	29339.7	520.3	**0.835**	40283.4	513.9	0.954	36795.2	499.6	1.284	**26266.7**
stufe10	56.9	0.147	3712.4	62.0	0.083	5084.7	59.3	0.089	4507.3	**46.0**	**0.079**	**2780.9**
t60k	68.3	0.366	8577.2	80.6	**0.222**	11276.4	77.4	0.253	9820.6	**60.7**	0.229	**6325.5**
vibrobox	8844.1	0.318	6800.3	12822.1	**0.139**	10640.1	14146.4	0.143	10419.6	**5551.0**	0.339	**5576.5**
wave	6765.4	2.241	48204.8	11537.7	**1.374**	74861.3	10690.7	1.559	71069.1	**4894.8**	2.434	**40790.6**

the 'ocean' graph, Party performs quite well often calculating good partitionings that outperform those found by Metis and Jostle, which is especially visible for large imbalances. While for high imbalance settings the quality of the partitionings computed by Jostle is better than the one obtained using Metis, this differs for small imbalance allowances where Metis (either pmetis or kmetis) produces slightly better average results.

Table 3 shows the standard deviation of the computed edge cut. It can be seen that although a graph's structure does not change when permuting its vertices, the results are highly affected by it. Thus, we see the deviation as an indicator for the reliability for the algorithm, that provides information in what quality range the computed results can be expected. From the table, one can see that although variations occur for all three libraries, it is often smallest for Party. However, from the results we obtained it is not clear if and how the reliability of the algorithms is influenced by weakening the balancing constraints, as in some cases the variation becomes smaller while the opposite can be observed for other graphs. We think that this factor might depend too much on the graphs' structure and thus no conclusion is possible.

Table 3. Standard deviation of the computed edge cut during 100 runs with the chosen set of graphs with an imbalance allowance of 0%, 3%, 15% and 75%, respectively.

Graph	Jostle	pmetis	kmetis	Party
no imbalance allowed				
144	536.2	210.3	304.9	77.0
4elt	27.2	23.8	26.6	14.5
biplane9	26.6	10.8	14.8	5.8
crack	15.5	13.9	15.6	7.3
dime20	33.3	18.3	20.4	21.9
grid100	11.0	9.2	10.5	1.1
memplus	208.5	216.6	191.3	115.9
m14b	569.9	132.6	408.6	58.3
ocean	262.7	19.8	39.7	16.4
stufe10	22.0	18.7	21.6	4.7
t60k	12.4	10.4	15.4	4.1
vibrobox	380.7	318.7	592.2	151.6
wave	398.1	346.0	387.8	192.1
allowed imbalance = 3%				
144	541.6	211.2	236.1	151.6
4elt	26.4	23.4	19.6	15.4
biplane9	26.2	10.8	11.7	5.0
crack	14.4	14.2	10.9	6.7
dime20	29.5	20.3	18.2	24.6
grid100	7.7	9.9	7.9	0.6
memplus	206.7	223.2	236.4	114.9
m14b	567.0	131.4	349.5	124.0
ocean	296.2	21.5	49.8	64.6
stufe10	22.0	16.9	18.5	5.6
t60k	14.1	9.8	49.9	4.9
vibrobox	426.2	319.1	367.1	207.9
wave	393.7	344.6	352.7	182.9
allowed imbalance = 15%				
144	557.6	199.9	248.5	299.1
4elt	22.0	23.8	18.0	10.9
biplane9	22.8	10.1	12.5	6.7
crack	10.3	14.8	8.9	6.6
dime20	25.8	21.6	17.1	11.3
grid100	4.3	9.4	7.9	0.8
memplus	217.6	208.5	302.4	97.4
m14b	484.3	223.2	111.7	151.0
ocean	290.4	20.7	54.9	49.6
stufe10	22.6	21.1	19.0	9.6
t60k	17.3	14.7	17.0	7.0
vibrobox	532.4	326.5	365.0	234.7
wave	475.3	351.3	348.6	505.3
allowed imbalance = 75%				
144	823.5	1863.4	1799.9	310.1
4elt	17.5	23.4	24.6	9.9
biplane9	15.0	18.8	18.9	7.1
crack	25.3	20.3	19.6	6.4
dime20	40.9	69.1	61.8	13.9
grid100	8.8	13.6	11.4	2.3
memplus	298.5	386.7	346.0	53.1
m14b	628.6	675.1	1158.0	607.5
ocean	188.2	109.2	67.5	84.3
stufe10	17.8	18.8	16.7	4.9
t60k	6.4	15.7	13.7	6.1
vibrobox	1026.1	1134.8	1425.5	401.5
wave	1851.9	2427.5	2017.8	1167.2

5. Conclusion

We have shown that the Helpful-Set heuristic implemented in Party has been successfully extended to find high quality 2-partitionings of graphs under less restrictive balancing constraints. Applying our permutation based evaluation method, we have shown that for 2-partitioning the Helpful-Set refinement algorithm requires less memory and a comparable amount of time compared to the other state-of-the-art libraries Jostle and Metis, while its reliability and the average computed edge cut is often superior.

References

[1] H. Räcke, "Minimizing congestion in general networks", in *Proceedings of the 43rd Symp. on Foundations of Computer Science*, pp. 43–52, 2002.

[2] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs", *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1999.

[3] C. Walshaw and M. Cross, "Mesh partitioning: A multilevel balancing and refinement algorithm", *SIAM Journal on Scientific Computing*, vol. 22, no. 1, pp. 63–80, 2000.

[4] B. Hendrickson and R. Leland, "A multi-level algorithm for partitioning graphs", *Proceedings of Supercomputing'95*, 1995.

[5] R. Diekmann, B. Monien, and R. Preis, "Using helpful sets to improve graph bisections", *Interconnection Networks and Mapping and Scheduling Parallel Computations*, AMS, vol. 21, pp. 57–73, 1995.

[6] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions", *ACM IEEE Nineteenth Design Automation Conference Proceedings*, pp. 175–181, 1982.

[7] B. W. Kernighan and S. Lin, "An efficient heuristic for partitioning graphs", *Bell Systems Technical Journal*, vol. 49, pp. 291–308, 1970.

[8] J. Hromkovic and B. Monien, "The bisection problem for graphs of degree 4 (configuring transputer systems)", *Proceedings of Mathematical Foundations of Computer Science*, LNCS vol. 520, pp. 211-220, 1991.

[9] S. Schamberger, "Improvements to the Helpful-Set Heuristic and a New Evaluation Scheme for Graphs-Partitioners", to appear, 2003.