Alcom-FT Technical Report Series
ALCOMFT-TR-03-143

Improvements to the Helpful-Set Algorithm and a New Evaluation Scheme for Graph-Partitioners *

Stefan Schamberger

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

Abstract. Graph partitioning is an important subproblem in many applications. To solve it efficiently, the multilevel strategy in combination with a matching algorithm and a local refinement heuristic has proven to be a powerful method, and several libraries exist providing such an implementation. Due to the large involvement of heuristics, the evaluation of these libraries is usually based on experiments. In this paper we show that single experiments are usually not sufficient to judge the quality of an algorithm, since even results obtained for graphs of and identical structure show high variations. This is still true, even if the applied algorithms do not contain any nondeterminism. We propose a scheme that considers these variations and therefore makes evaluations and comparisons of different implementations more meaningful. We have applied this technique to evaluate the improvements of the Helpful-Set 2-partitioning implementation and present the obtained results.

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 deals with the 2-partitioning problem of unweighted graphs.

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 [6], Jostle [8], Chaco [4] or Party [1] 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

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

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.

Due to speed requirements, all of the three described tasks usually involve some kind of heuristics. In most cases for example, the local refinement process is based on the Fiduccia-Mattheyses method [3], a run-time optimized version of the Kerninghan-Lin (KL) heuristic [7]. Even if approximation algorithms are applied to one of the tasks, the overall calculation can still be seen as a heuristic, since their interaction has not been theoretically investigated yet.

The high involvement of heuristics leads to the fact that partitioning libraries are usually evaluated by performing experiments on a set of selected graphs. These results are seen as an indication of how good the library would perform on similar graphs. However, in [2] it is shown that the calculations of the graph partitioning libraries are not always deterministic. More precisely, it has been shown that different initializations of the random number generator, used in some parts of the implementations, lead to deviations in the solutions quality. Thus, the question arises if a reliable prediction of the partitionings' quality can be made performing only one experiment on each graph, as it has been done in many comparisons so far.

This paper has two goals. First we demonstrate that results calculated by partitioning libraries are not totally predictable, even if the applied algorithms themselves do not include any nondeterminism. We do this by generating graphs of an identical structure and show, that the obtained partitionings vary a lot in their quality. This observation leads to a proposal of a new evaluation scheme. Second, we show that it is possible to further improve the local refinement Helpful-Set algorithm presented in [1]. During the development, our evaluation scheme has thereby shown to be quite useful, helping us to enlarge the reliability of the algorithm. Results of experiments performed with it are presented and finally we also test Party against state-of-the-art partitioning libraries using the new evaluation scheme.

The remaining part of this paper is organized as follows. In the next Section we describe why and how we randomize a graph, and present a new evaluation scheme for graph partitioning libraries. In Section 3, a very brief overview to the Helpful-Set heuristic and its improvements is given. Results obtained with it are presented in Section 4, showing the enhancements of Party and tests against two other state-of-the-art partitioning libraries, Jostle and Metis. In Section 5 we give a brief conclusion.

2 A New Evaluation Scheme

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.

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 and their properties are completely different to those of 'real world' problems. However, a graph can be partly randomized keeping its structure. This is done by permuting its vertices.

Figure 1 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.

To make use out of this observation, we created a new evaluation scheme, based on the described technique. 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, generated by a script. Figure 2 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 weight of the largest partition ($\mathrm{pw}_{\mathrm{max}}$), 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

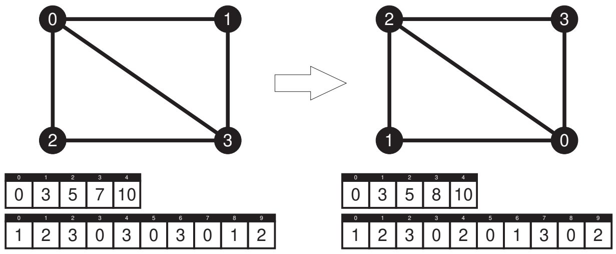

Fig. 1. Graph permutation. The original graph and its data structure (left) is transformed by the permutation $\sigma = (2, 3, 1, 0)$ into the new graph (right).

all 100 runs is displayed by a large solid mark. The right part shows the required resources. Since these do not depend much on the chosen node permutation, only the average time and memory consumption from all runs for this graph is displayed.

In anticipation of Section 4, we now describe how the chart shown in Figure 2 can be interpreted. The yellow, orange and red squares display results obtained using Jostle with 0%, 1% and 3% allowed imabalce, respectively. One can see that the average edge cut computed gets smaller with a higher imbalance allowance. 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. While the results obtained by pmetis, Jostle with no imbalance and the former implementation of Party (green diamonds) show a similar behavior in the edge cut, the new implementation of Party (black circles) does compute less divergent (often identical) solutions here, reaching an average edge cut of 100.9. On the right hand side the resources consumed by the different libraries are displayed. Since the variation here has shown to be negligible, only the average value is presented. In case of the 100x100 grid, Jostle needs longest to compute its results, followed by Metis and Party, 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 with 1% and 3% imbalance allowance would have perfomed worse while Jostle with no imbalance and Party would have come off better, resulting in different conclusions.

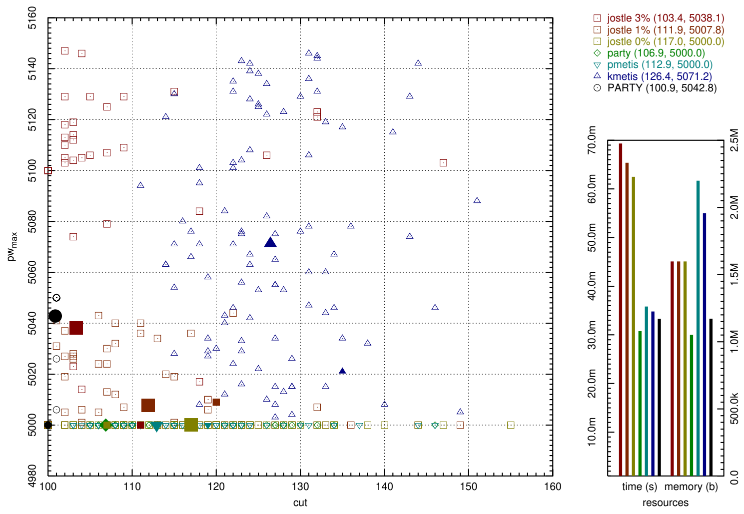

Fig. 2. Results of a 2-partitioning of a 100x100 2D grid

3 Improvements to the Helpful-Set Heuristic

This Section is meant to give an idea about the functioning the Helpful-Set algorithm and the improvements we made to its implementation presented in [1]. 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 [5]. It has been implemented and presented in [1] 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 V_0 or V_1 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 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 3 shows an example of a successful round reducing the edge cut from 9 to 7.

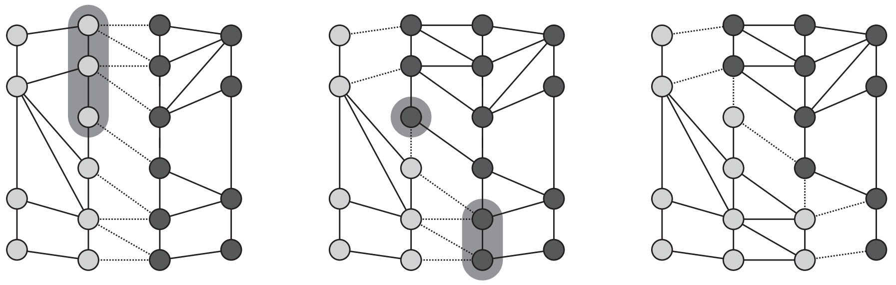

Fig. 3. Moving the marked set (left) reduces the current edge cut of 9 by 2 (thus the set is called 2-helpful), but it destroys the balance by 3. However, by moving the second marked set (middle), this can be fixed (thus this set is called a balancing set), and a balanced solution (right) is found leading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

Due to runtime concerns, the search for Helpful-Sets can only be performed using a simple greedy algorithm always adding the current best vertex to the set. This approach uses the 'bucket' data structures known from the KL improvement [3]. It has turned out however, that the crucial point when implementing the Helpful-Set algorithm is the choice of the helpfulness value l of the sets to be searched for. If l is chosen too small, then promising sets are overlooked, while setting l too large quickly increases the runtime but usually does not discover better sets. The former implementation in Party therefore uses a technique called 'adaptive limitation' [1] to determine l. Setting it to $cut/2$ initially, l is halved or set to the best occurring helpfulness if a l-helpful set can not be found in any of the two sets and doubled if the search has succeeded.

The improvements we made can be characterized into three aspects. First, instead of using only one limit value l for both sets, we introduce a separate one for each. This also leads to some changes in the 'adaptive limitation'. Second, we included the possibility of having slight imbalances. This feature has also been implemented in Jostle and Metis, and it has proven to be quite successful since slight imbalances often make better edge cuts possible. Especially in deeper levels of the multilevel approach it is not necessary (often not even possible) to completely balance the two sets. Third, the former implicit balancing mechanism (that is modifying the weight of the balancing sets) has been moved to the end of the algorithm and is performed explicitly, now always ensuring the requested bounds and increasing the reliability of the library.

4 Experimental Results

To evaluate the changes to the HS heuristic described in Section 3, we have implemented it (denoted as 'Party' and tested it against the former implementation (denoted as 'party 2.001') and two other well known graph partitioning libraries, Jostle (version 2.3) and Metis (version 4.0.1). Both of the latter are also based on the multilevel paradigm and apply a Kerninghan-Lin like refinement algorithm.

Throughout this paper we used a set of well known graphs for our comparisons. Most of the graphs have also been used in [8], comparing Jostle and Metis. Since we do not have access to the graphs 'mesh1m', 'oliker' and 'bmw1c', we replaced these two graphs by the 'airfoil1', 'biplane9', 'grid100', 'stufe10' and 'wave' graphs. Table 1 gives an overview of the test set and some of the graph's 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.

As mentioned, a slight imbalance allowance does often lead to much smaller edge cuts. If an imbalance of $i\%$, $0 \leq i \leq 100$ is required, a partition's weight may not exceed $\lceil |V|/2 \cdot (100+i)/100 \rceil$. Jostle accepts an imbalance parameter and we used values of 3% (default), 1% and 0% of allowed imbalance. In the present version, Metis does not allow to adjust this parameter, instead it is set

Graph	$\lvert V\rvert$	$\lvert E\rvert$	min. deg.	av. deg.	max. deg.	diameter	origin
144	144649	1074393	4	14.86	26	35	FEM 3D
4elt	15606	45878	3	5.88	10	67	FEM 2D
airfoil1	4253	12289	3	5.78	9	43	FEM 2D
biplane.9	21701	42038	2	3.87	4	102	FEM 2D
crack	10240	30380	3	5.93	9	84	FEM 2D
dime20	224843	336024	2	2.99	3	1105	FEM 2D dual
grid100	10000	19800	2	3.96	4	198	FEM 2D
m14b	214765	1679018	4	15.64	40	33	FEM 3D
memplus	17758	54196	1	6.10	573	10	digital memory circuit
ocean	143437	409593	1	5.71	6	229	FEM 3D dual
stufe.10	24010	46414	2	3.87	4	54	FEM 2D
t60k	60005	89440	2	2.98	3	495	FEM 2D dual
vibrobox	12328	165250	8	26.81	120	3	vibroacoustic matrix
wave	156317	1059331	3	13.55	44	38	FEM 3D

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

to constant 0% and 3% for the two versions pmetis and kmetis, respectively. While version 2.001 and earlier of the party library cannot handle imbalances, this feature has been added in the new implementation. All experiments have been performed on a Pentium III 933 MHz dual processor system with 1 GB of main memory using one processor.

We now describe the results we obtained by applying the new algorithm to the set of graphs listed in Table 1. First, we compare the improved implementation of the Helpful-Set method with the former one. Due to space limitations, we only present two of the charts and summarize results obtained for all graphs in Table 2. It shows that in all tested cases the new implementation achieves a smaller average edge cut even if no imbalance is allowed, while the time and memory requirements only slightly increase. If imbalance is added, the quality of the results can be further improved in some cases as shown for example for the 't60k' graph in Figure 4. For other graphs however, the opposite can be observed. One example is the 'dime20' graph, which is shown in more detail in Figure 5. There, the average edge cut increases from 185.0 in case of no allowed imbalance to 190.0 with 3% imbalance. We found this quite surprising since one does expect equal or better solutions by increasing the feasible set size. However, with this increase the search for the best solution becomes more difficult, since more, probably worse, local optima exist. On the other hand, Figure 5 also reveals that the cut size of the best solution found during all 100 runs is smaller for 3% imbalance than in the 1% or 0% case. Note, that the decrease would have been much higher if no permutation had been applied. With it, the results are acceptable. Additionally, the solutions found vary more in case of the 't60k' graph than for the 'dime20', indicating that Party has more difficulties in the latter case due to the combination of the graph's structure and the applied heuristics.

Next, we show the results we obtained comparing the new implementation of the Helpful-Set heuristic to the state-of-the-art partitioning libraries Jostle and Metis. Tables 3 and 4 give an overview to the results obtained for all graphs from Table 1. Due to space limitations and better comparability we only include results for Party obtained with 3% imbalance allowance. From Table 3 one can see

Graph	party 2.001			Party									
	imbalance=0%			imbalance=0%		imbalance=1%			imbalance=3%			all
	edge cut	time	memory	edge cut	time	edge cut	time	pw_{max}	edge cut	time	pw_{max}	memory
144	6963.1	**1.949**	45741.8	6730.2	2.000	**6682.9**	2.200	72844.6	6814.1	2.318	74152.6	**38744.7**
4elt	167.6	**0.054**	2008.9	**148.6**	0.060	149.7	0.061	7854.9	149.4	0.061	7925.5	**1855.9**
airfoil1	84.3	**0.011**	537.1	81.5	0.014	78.8	0.014	2139.7	**77.5**	0.014	2174.6	**518.4**
biplane.9	90.5	**0.074**	**2186.9**	79.9	0.075	78.7	0.076	10917.8	**77.4**	0.075	11043.7	2494.1
crack	203.3	**0.035**	**1348.1**	192.5	0.046	192.2	0.046	5155.4	**191.3**	0.046	5221.1	1473.1
dime20	213.4	**1.227**	**18678.6**	**185.8**	1.285	189.5	1.282	113058.4	190.0	1.287	114377.9	23215.9
grid100	106.9	**0.031**	**1027.0**	**100.2**	0.034	100.9	0.034	5042.8	100.9	0.034	5141.0	1144.2
memplus	5858.7	**0.161**	**4620.6**	5791.4	0.372	5744.5	0.384	8965.9	**5642.3**	0.374	9139.8	5774.8
m14b	4127.0	**3.337**	71679.4	**3956.8**	3.602	4004.0	3.748	108236.1	4081.8	3.810	110064.4	**61310.7**
ocean	551.1	**1.204**	**22517.8**	496.1	1.262	505.9	1.287	72142.9	**391.6**	1.277	72932.5	26266.7
stufe.10	62.2	**0.079**	**2435.8**	52.0	0.083	51.3	0.083	12073.2	**51.0**	0.083	12204.9	2780.9
t60k	100.3	0.241	**5152.4**	91.9	**0.234**	88.7	0.235	30218.4	**79.4**	0.235	30836.2	6325.5
vibrobox	11974.8	**0.160**	7320.1	11711.7	0.287	**11691.8**	0.368	6191.4	11730.0	0.433	6269.4	**5576.5**
wave	9445.8	**2.051**	46399.3	9104.9	2.234	9070.8	2.388	78797.0	**9049.6**	2.471	80052.5	**40790.6**

Table 2. Comparison of party 2.001 and Party. The best average result is printed bold while results not within 10% (edge cut) and 25% (time and memory) of this value are shaded gray.

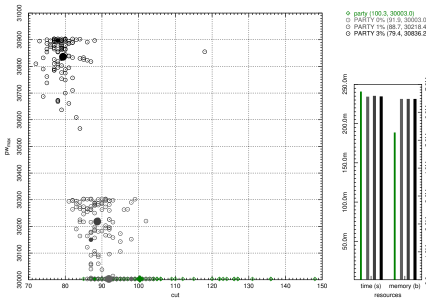

Fig. 4. 2-partitioning of 't60k'

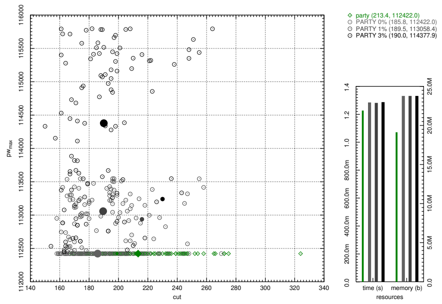

Fig. 5. 2-partitioning of 'dime20'

that Party often computes the best average edge cut, followed by Jostle (allowing 3% imbalance) and pmetis (allowing no imbalance). Furthermore, Party seems to make more use of the freedom granted by the imbalance setting than Jostle or Metis. However, also allowing only 1% imbalance results in a similar picture (omitted here, see Table 2) with Party computing the best average results.

The resource requirements are displayed in Table 4. While Party always consumes least memory, Metis (kmetis and pmetis) are fastest. In some cases, Party can compete with Metis, while in some others it is slightly slower than Jostle. This high difference between the graphs is probably due to problems inside the matching strategy, resulting in constructing many unnecessary levels and causing lots of unsuccessful searches. Hence, this matter requires further work.

For two graphs the results included in Tables 3 and 4 are presented in more detail in Figures 6 and 7. As mentioned earlier, in case of the 'dime20' the edge cut computed by Party does not increase with allowed imbalance, but is still good compared to the other libraries. Again, this is also a good example for a different ranking mentioned in Section 2 that would have been obtained without the permutation approach. Figure 7 is an example where Party performs slowly. In contrast to the 'm14b' where this time is more or less wasted, it turns into account for the 'memplus' graph finding much better edge cuts.

Graph	Jostle 3%		Jostle 1%		Jostle 0%		pmetis		kmetis		PARTY
	edge cut	pw_{max}	edge cut	pw_{max}	edge cut	pw_{max}	edge cut	pw_{max}	edge cut	pw_{max}	edge cut
144	7030.4	73263.4	7040.0	72765.6	7047.8	72325.0	6855.9	72325.0	6989.6	72642.1	**6814.1**
4elt	162.8	7911.7	164.7	7836.7	169.9	7803.0	160.5	7803.0	169.0	7888.2	**149.4**
airfoil1	84.0	2162.2	86.6	2134.4	88.5	2127.0	84.7	2127.0	88.5	2168.9	**77.5**
biplane9	99.4	10993.3	101.1	10903.9	105.0	10851.0	84.8	10851.0	89.2	10949.4	**77.4**
crack	201.4	5185.3	203.7	5137.0	205.8	5120.0	199.8	5120.0	210.8	5188.9	**191.3**
dime20	224.3	113713.1	228.5	112942.7	243.3	112422.0	192.0	112424.9	194.1	112598.3	**190.0**
grid100	103.4	5038.1	111.9	5007.8	117.0	5000.0	112.9	5000.0	126.4	5071.2	**100.9**
memplus	6092.3	9085.6	6140.5	8966.5	6183.7	8879.0	6555.6	8879.2	6583.8	9126.8	**5642.3**
m14b	4203.5	108592.4	4206.3	107916.2	4221.7	107383.0	**4056.9**	107383.0	4145.0	107573.8	4081.8
ocean	438.6	73078.1	500.1	72319.8	564.4	71719.0	504.5	71722.7	512.9	72108.5	**391.6**
stufe10	62.5	12118.0	62.8	12052.3	64.8	12005.0	59.0	12005.6	61.7	12092.0	**51.0**
t60k	92.8	30469.2	97.1	30129.8	101.4	30003.0	96.5	30009.9	105.5	30192.2	**79.4**
vibrobox	11821.1	6320.0	11858.3	6218.7	11802.8	6164.0	11792.4	6164.1	11857.4	6297.5	**11730.0**
wave	9410.1	79262.9	9416.5	78628.5	9404.2	78159.0	9381.0	78159.0	9441.4	78539.7	**9049.6**

Table 3. Quality of 2-partitionings computed by Jostle, Metis and Party (with 3% imbalance allowance). The smallest average edge cut is printed bold while results not within 10% of this value are shaded gray.

Graph	Jostle 3%		Jostle 1%		Jostle 0%		pmetis		kmetis		PARTY	
	time	memory	time	memory	time	memory	time	memory	time	memory	time	memory
144	2.180	47072.7	2.186	47072.7	2.257	47072.7	**1.321**	73596.3	1.515	70080.5	2.332	**38744.7**
4elt	0.115	2721.4	0.113	2721.4	0.113	2721.4	0.060	3969.8	0.062	3596.9	**0.058**	**1855.9**
airfoil1	0.024	739.0	0.024	739.0	0.028	739.0	0.016	1078.6	**0.014**	982.5	0.014	**518.4**
biplane9	0.145	3338.8	0.145	3338.8	0.148	3338.8	0.081	4581.5	0.083	4060.0	**0.072**	**2494.1**
crack	0.071	1794.5	0.069	1794.5	0.070	1794.5	0.043	2820.0	**0.042**	2580.5	0.045	**1473.1**
dime20	1.462	31531.6	1.463	31531.6	1.504	31531.6	**1.011**	41622.7	1.169	36157.1	1.292	**23215.9**
grid100	0.069	1560.4	0.065	1560.4	0.062	1560.4	0.036	2147.2	0.035	1910.7	**0.033**	**1144.2**
memplus	0.325	5887.4	0.340	5887.4	0.356	5887.4	**0.118**	6739.4	0.125	6535.7	0.372	**5774.8**
m14b	3.298	72498.3	3.298	72498.3	3.355	72498.3	**2.163**	114897.6	2.452	109672.5	3.817	**61310.7**
ocean	1.330	29339.7	1.334	29339.7	1.361	29339.7	**0.837**	40283.4	0.958	36795.2	1.274	**26266.7**
stufe10	0.148	3712.4	0.148	3712.4	0.151	3712.4	**0.084**	5084.7	0.090	4507.3	0.086	**2780.9**
t60k	0.369	8577.2	0.370	8577.2	0.382	8577.2	**0.225**	11276.4	0.253	9820.6	0.235	**6325.5**
vibrobox	0.402	6800.3	0.392	6800.3	0.417	6800.3	0.158	10640.1	**0.148**	10419.6	0.437	**5576.5**
wave	2.263	48204.8	2.267	48204.8	2.392	48204.8	**1.372**	74861.3	1.558	71069.1	2.473	**40790.6**

Table 4. Time (s) and memory (Kb) consumption of the computations. The best average result is printed bold while results not within 25% of this value are shaded gray.

Concerning the influence of the permutations, the most interesting information is presented in Table 5. It shows the standard deviation of the computed solutions' quality (edge cut and balance) for each library, respectively. First, it is evident that permuting the graphs' vertices has a large impact on the results computed by all libraries. Furthermore, one can see that the variation of the edge cut produced by Jostle is usually highest, followed by Metis, whereas Party more often calculates partitionings a bit more close together. There are still a few exceptions to this rule, like in the case of the 'm14b' and 'ocean' graphs. This might be an indication that the current implementation has some problems with these graph and that further investigation is required to solve this matter. Comparing the balance, it is more difficult to make any conclusions, since all libraries compute comparable results, the same imbalance setting implied. Furthermore, a high variation here on the one hand could be interpreted as an indicator how extensively the search space is explored. On the other hand, this could also point to problems in leaving local minima.

5 Conclusion

In this paper we have shown that the solutions computed by the graph partitioning libraries Jostle, Metis and Party are highly affected by a random permutation of the graphs' vertices. Since experiments and comparisons usually are thought

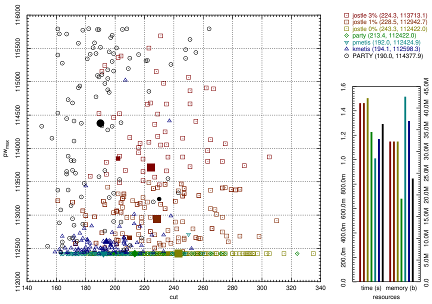

Fig. 6. 2-partitioning of 'dime20'

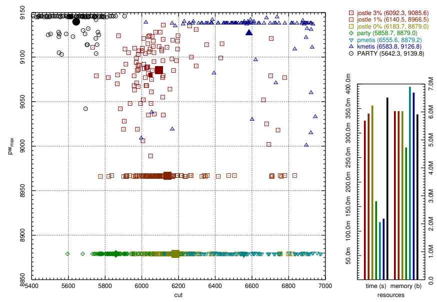

Fig. 7. 2-partitioning of 'memplus'

Graph	Jostle 3%		Jostle 1%		Jostle 0%		pmetis		kmetis		PARTY	
	edge cut	pw_{max}	edge cut	pw_{max}	edge cut	pw_{max}	edge cut	pw_{max}	edge cut	pw_{max}	edge cut	pw_{max}
144	541.6	575.1	540.7	184.6	536.2	0.0	211.2	0.0	236.1	261.5	151.6	331.1
4elt	26.4	72.7	27.3	25.5	27.2	0.0	23.4	0.0	19.6	60.3	15.4	65.5
airfoil1	8.3	19.6	8.1	5.6	8.7	0.0	7.0	0.0	8.3	19.3	5.0	13.4
biplane9	26.2	97.7	27.1	32.9	26.6	0.0	10.8	0.0	11.7	76.1	5.0	93.7
crack	14.4	44.6	15.1	15.2	15.5	0.0	14.2	0.0	10.9	45.6	6.7	50.2
dime20	29.5	784.1	30.8	306.6	33.3	0.0	20.3	28.8	18.2	341.3	24.6	1098.8
grid100	7.7	52.7	10.6	13.4	11.0	0.0	9.9	0.0	7.9	43.3	0.3	17.3
memplus	206.7	31.9	208.3	0.5	208.5	0.0	223.2	0.4	236.4	32.0	114.9	14.4
m14b	567.0	812.7	567.4	257.7	569.9	0.0	131.4	0.0	349.5	148.3	124.0	702.9
ocean	296.2	518.0	279.4	173.3	262.7	0.0	21.5	26.3	49.8	365.4	64.6	686.2
stufe10	22.0	97.4	21.7	33.5	22.0	0.0	16.9	6.1	18.5	82.5	5.6	116.1
t60k	14.1	222.1	12.6	86.3	12.4	0.0	9.8	40.2	49.9	224.7	4.9	74.3
vibrobox	426.2	47.2	406.9	12.3	380.7	0.0	319.1	0.4	367.1	55.0	207.9	40.4
wave	393.7	636.9	394.8	197.5	398.1	0.0	344.6	0.0	352.7	280.8	182.9	581.6

Table 5. Standard deviation of edge cut and balance occurring during 100 runs.

as a kind of prediction of how good a heuristic performs on certain types of graphs, we think that it is essential to consider this fact in this kind of experiments. Therefore, we have proposed a simple evaluation scheme, that is able to affect all, not only nondeterministic implementations.

Although our scheme does not overcome the evaluation problems caused by 'small' test sets, it has shown to be helpful in developing the improvements made to Party, saving time and pointing us to some difficulties producing high variations in the solutions' quality. Additionally, it provides information about how reliable a graph partitioning heuristic is and therefore indicates in what quality range solutions can be expected.

Concerning the improvement of Party's Helpful-Set algorithm, we were able to reduce the computed average edge cut by about 2% – 19% (no imbalance allowance) and 2% – 22% (3% imbalance allowance) for the graphs in our test set. Compared to state-of-the-art partitioning libraries, superior 2-partitionings can often be found in a comparable amount of time.

References

1. R. Diekmann, B. Monien, and R. Preis. Using helpful sets to improve graph bisections. In D.F. Hsu, A.L. Rosenberg, and D. Sotteau, editors, *Interconnection Networks and Mapping and Scheduling Parallel Computations*, volume 21 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 57–73. AMS, 1995.
2. U. Elsner. *Static and Dynamic Graph Partitioning*. PhD thesis, Technische Universität Chemniz, 2001.
3. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *ACM IEEE Nineteenth Design Automation Conference Proceedings*, pages 175–181, Los Alamitos, Ca., USA, Jun 1982. IEEE Computer Society Press.
4. B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In *Proceedings of Supercomputing'95*, San Diego, CA, Dec 1995. ACM/IEEE.
5. J. Hromkovic and B. Monien. The bisection problem for graphs of degree 4 (configuring transputer systems). In A. Tarlecki, editor, *Proceedings of Mathematical Foundations of Computer Science. (MFCS '91)*, volume 520 of *LNCS*, pages 211–220, Berlin, Germany, Sep 1991. Springer.
6. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, Jan 1999.
7. B. W. Kernighan and S. Lin. An efficient heuristic for partitioning graphs. *Bell Systems Technical Journal*, 49:291–308, Feb 1970.

8. C. Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1):63–80, 2000.