Improvement of the Efficiency of Genetic Algorithms for Scalable Parallel Graph Partitioning in a Multi-level Framework

Cédric Chevalier and François Pellegrini

LaBRI and INRIA Futurs
Université Bordeaux I
351, cours de la Libération, 33405 TALENCE, France
{cchevali, pelegrin}@labri.fr

Abstract. Parallel graph partitioning is a difficult issue, because the best sequential graph partitioning methods known to date are based on iterative local optimization algorithms that do not parallelize nor scale well. On the other hand, evolutionary algorithms are highly parallel and scalable, but converge very slowly as problem size increases. This paper presents methods that can be used to reduce problem space in a dramatic way when using graph partitioning techniques in a multi-level framework, thus enabling the use of evolutionary algorithms as possible candidates, among others, for the realization of efficient scalable parallel graph partitioning tools. Results obtained on the recursive bipartitioning problem with a multi-threaded genetic algorithm are presented, which show that this approach outperforms existing state-of-the-art parallel partitioners.

1 Introduction

Graph partitioning is an ubiquitous technique which has applications in many fields of computer science and engineering, such as workload balancing in parallel computing, database storage, VLSI design or bio-informatics. It is mostly used to help solving domain-dependent optimization problems modeled in terms of weighted or unweighted graphs, where finding good solutions amounts to computing, eventually recursively in a divide-and-conquer framework, small vertex or edge cuts that balance evenly the weights of the graph parts.

For instance, the obtainment of small and balanced bipartitions is essential to the reordering of sparse matrices by nested dissection [5]. This method consists in computing a small vertex set that separates the adjacency graph of the sparse matrix in two parts, ordering the separator vertices with the highest indices available, then proceeding recursively on the two separated subgraphs until their size is smaller than some specified threshold. The smaller and more balanced the separators are, the smaller the fill-in incurred at the factorization stage, and thus the number of operations required to factor the matrix (referred to as the operation count, or OPC), is likely to be.

W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 243–252, 2006.

Currently, general-purpose sequential ordering software such as SCOTCH [12] or METIS [9] can handle graphs of about ten million vertices on an average workstation. However, as the power of parallel machines increases, so does the size of the problems to handle, and since the large graphs which model these problems cannot be processed on a single computer without incurring swapping, it is necessary to resort to parallel graph ordering tools, based on parallel graph bipartitioning algorithms. Several such tools have already been developed [9], but their outcome is mixed. In particular, they do not scale well, as partitioning quality tends to decrease, and thus fill-in to increase much, when the number of processors which run the program increases.

The purpose of the PT-SCOTCH software ("*Parallel Threaded* SCOTCH", an extension of the sequential SCOTCH software), developed at LaBRI within the SCALAPPLIX project of INRIA Futurs, is to provide efficient parallel tools to partition graphs with sizes up to a billion vertices, distributed over a thousand processors. Among our target applications is the parallel ordering of large graphs.

PT-SCOTCH is still under development, but several results have already been achieved. Section 2 presents a constrained banding technique which, based on the characteristics of the local optimization algorithms that are used to refine the partitions, reduces considerably the size of the problem space without loss of quality, already allowing one to develop semi-parallel programs that can compute efficient bipartitions of graphs having a billion nodes. Section 3 describes how this reduction enables us to use genetic algorithms, which are highly scalable but slow to converge, in a practical way. Some graph ordering results are presented, using a multi-threaded shared-memory genetic algorithm, which illustrate the quality of the orderings that can be produced. Then comes the conclusion.

2 Reducing Problem Space in a Multi-level Framework

Experience has shown that best partition quality is achieved when using a multi-level framework. This method, which derives from the multi-grid algorithms used in numerical physics, repeatedly reduces the size of the graph to partition by finding matchings that collapse vertices and edges, computes an initial partition for the coarsest graph obtained, and projects the result back to the original graph [2,6,8]. It is most often combined with greedy iterative algorithms, such as Kernighan-Lin [10] or Fiduccia-Mattheyses [4] (FM), to refine the projected partitions at every level, so that the granularity of the solution is the one of the original graph and not the one of the coarsest graph.

Because of the local nature of both the FM and the uncoarsening algorithms, it is most likely that the refined partition computed at any level will not differ much from the partition that was projected back to this level, as this latter is itself the projection of a partition that was a local optimum in the coarser levels. Therefore, to refine a partition, FM-like algorithms may not need to know more of the graph topology than a small "band" around the boundary of the projected partition. The locality of the optimization process is already exploited in many implementations of FM-like algorithms which, in order to save time and memory,

Table 1. Some of the test graphs that we use

Graph	Size ($\times 10^3$)		Average degree
	V	E	
598a	111	742	13.37
aatken	43	88	4.14
auto	449	3315	14.77
bcsstk29	14	303	43.27
bcsstk30	29	1007	69.65
bcsstk32	45	985	44.16
body	45	164	7.26
bracket	63	367	11.71
coupole8000	1768	41657	47.12
m14b	215	1679	15.64
ocean	143	410	5.71
pwt	37	145	7.93
rotor	100	662	13.30
s3dkq4m2	90	2365	52.30
tooth	78	453	11.58

Graph	Size ($\times 10^3$)		Average degree
	V	E	
audikw1	944	38354	81.28
b5tuer	163	3874	47.64
bmw32	227	5531	48.65
bmwcra1	149	5248	70.55
crankseg2	64	7043	220.64
inline1	504	18156	72.09
mt1	98	4828	98.96
oilpan	74	1762	47.77
ship001	35	2305	132.00
shipsec5	180	4967	55.23
thread	30	2220	149.32
x104	108	5030	92.81
altr4	26	163	12.50
chanel1m	81	527	13.07
conesphere1m	1055	8023	15.21

compute and update vertex swapping gains only for vertices that have to be considered, that is, the ones that are in the immediate vicinity of vertices that currently belong to the separator. However, these vertices cannot be known in advance. Our idea is that, since the FM algorithm is local, we can constrain it to operate on a small, predefined band of graph vertices without changing significantly its outcome.

To validate this assumption, we have instrumented our SCOTCH sequential partitioning software in order to measure how much refined partitions differ from projected partitions. Since our current target application requires vertex separators, we have focused on them for these experiments, but the same kind of measures could be obtained from edge separation routines as well. The test graphs we have used in all of our experiments are well-known cases of various sizes, listed in Table 1.

For every separator computed in a nested dissection process (which stops when subgraphs are of sizes of about a hundred vertices), we accumulate the numbers of refined separator vertices that end up at a given distance from the projected separators. These results are presented in Table 2.

As expected, the overwhelming majority of refined separator vertices is not located at a distance greater than three from the vertices of the projected separators. Therefore, it can be assumed that the quality of partitions should not be impacted if refined partitions are computed on band graphs only. In order to validate this second assumption, we have developed in SCOTCH a partitioning method which extracts a band subgraph of given width from a given graph and its given initial separator, applies a FM separator refinement method to the initial separator of the band subgraph, and projects back the refined band separator to the full graph. We have then replaced all of our calls to the FM refinement algorithm by calls to this band FM refinement algorithm.

Table 2. Distance histogram (in % of the number of separator vertices) of the location of refined separator vertices with respect to projected separators. These statistics have been collected over all separators when performing nested dissection on the given graphs.

Graph	Distance					Graph	Distance				
	0	1	2	3	≥ 4		0	1	2	3	≥ 4
598a	76.23	23.45	0.32	0.00	0.00	audikw1	91.44	8.55	0.01	0.00	0.00
aatken	77.00	20.45	2.27	0.24	0.04	b5tuer	74.18	22.96	1.85	0.42	0.59
auto	77.89	21.89	0.22	0.00	0.00	bmw32	80.98	18.31	0.50	0.08	0.14
bcsstk29	82.04	17.66	0.30	0.00	0.00	bmwcra1	91.29	8.58	0.13	0.00	0.00
bcsstk30	87.17	12.53	0.29	0.01	0.00	crankseg2	95.80	4.17	0.01	0.02	0.00
bcsstk32	81.91	17.80	0.23	0.03	0.02	inline1	87.57	12.35	0.08	0.00	0.00
body	67.49	30.20	2.08	0.20	0.04	mt1	84.79	14.00	0.93	0.25	0.04
bracket	72.47	26.19	1.08	0.16	0.10	oilpan	77.60	20.54	1.20	0.17	0.49
coupole8000	90.23	9.74	0.03	0.00	0.00	ship001	91.43	8.51	0.05	0.00	0.00
m14b	78.65	21.17	0.18	0.00	0.00	shipsec5	82.29	17.28	0.41	0.03	0.00
ocean	60.43	32.86	4.58	1.29	0.84	thread	91.40	8.53	0.06	0.00	0.00
pwt	54.35	37.31	6.10	1.56	0.69	x104	86.64	12.81	0.51	0.03	0.00
rotor	77.09	21.99	0.75	0.11	0.06	altr4	74.19	24.89	0.80	0.12	0.00
s3dkq4m2	78.72	20.34	0.89	0.04	0.00	chanel1m	74.65	24.09	1.16	0.10	0.00
tooth	69.90	26.82	2.42	0.63	0.24	conesphere1m	82.16	17.67	0.17	0.00	0.00

The quality criterion that we have chosen is the operation count (OPC) required to factor the reordered matrix using a Cholesky method; it is an indirect measurement of the overall quality of all bipartitions, in the practical context of nested dissection ordering. The results that we obtain for all of our test matrices, using band graphs with a width of three, show only marginal differences in OPC compared to the original FM refinement algorithm, and no difference on average. An explanation to this is that, even if the separator cannot move more than three vertices away at any level, it has the ability to move again at the next levels to reach its local optimum, therefore compensating on several levels for the moves it could not do on a single level.

An interesting feature of band FM refinement is that is seems to be more stable than the classical FM algorithm. In the production version of Scotch, two runs of multi-level bipartitioning were performed for each subgraph, and then the best separator of the two was kept. When using band FM refinement, equivalent results are obtained with only one run, as presented in Table 3. Most of the time, the quality of band FM lies between the one exhibited by one and two runs of the classical FM method. In terms of time, we can evidence a moderate over-cost with respect to a single run of classical FM, because of the computation of the band graph. It seems that, by "amortizing" the move of the frontier, the band FM algorithm prevents it from exploring local minima that differ too much from the "pseudo-global" solution computed at the coarsest level and in which it could be trapped afterwards. Further experiments are required to investigate this.

Table 3. Comparison between band FM and classical FM. Tests have been run on a 375MHz IBM SP3.

Graph	Band FM (1 run)		FM (2 runs)		FM (1 run)	
	OPC	Time (s)	OPC	Time (s)	OPC	Time (s)
aatken	1.72e+11	6.17	**1.70e+11**	10.79	1.73e+11	5.38
auto	5.14e+11	47.09	**4.98e+11**	75.00	5.27e+11	39.40
bcsstk32	1.40e+9	1.16	**1.28e+9**	1.65	1.40e+9	1.02
coupole8000	7.57e+10	210.15	**7.48e+10**	346.81	7.57e+10	183.72
m14b	6.27e+10	21.4	6.31e+10	33.42	**6.03e+10**	17.56
tooth	**6.50e+9**	5.66	6.51e+9	9.01	6.71e+9	4.64
audikw1	5.58e+12	59.32	**5.48e+12**	86.78	5.64e+12	50.33
bmw32	3.15e+10	4.52	**2.75e+10**	6.51	3.07e+10	4.08
oilpan	2.92e+9	0.73	**2.74e+9**	0.95	2.99e+9	0.69
thread	4.17e+10	1.62	**4.14e+10**	2.30	4.17e+10	1.44
x104	1.84e+10	1.97	**1.64e+10**	2.60	1.80e+10	1.83
altr4	3.68e+8	1.55	**3.65e+8**	2.52	3.84e+8	1.32
conesphere1m	**1.83e+12**	122.03	1.85e+12	192.27	1.88e+12	100.19

By using this limitation of problem space, we can already devise a way to compute high-quality partitions of distributed 3D mesh graphs of up to a billion vertices: since the expected size of the separator of a n-vertex 3D mesh graph is in $O(n^{2/3})$ [14], the order of magnitude of the first separator of a 3D graph of about a billion vertices should be of about a million vertices, which can be handled by a sequential computer. Therefore, basing on existing parallel coarsening algorithms such as the one of [13], one can coarsen a distributed graph so as to get a coarsened graph that fits in the memory of a sequential computer, compute an initial bipartition of this coarse graph using existing sequential partitioners, and project back this partition as follows. During each uncoarsening step, once the separator has been projected back to the finer distributed graph, a centralized copy of the distributed band graph surrounding the projected separator is gathered on every processor. All of the processors can then run independently a classical sequential FM algorithm on their centralized band graph, leading to a better exploration of the reduced problem space, after which the best refined separator found is projected back to the finer distributed graph. This uncoarsening process is repeated up to obtain a distributed bipartition of the original graph. Recursive bipartitioning can then take place on the two parts created, with separators of smaller sizes.

The above scheme, which may be useful to handle large graphs at the expense of quite little work on top of existing software, is clearly not fully satisfactory, since the refinement of the partitions is sequential in nature, and thus not scalable. In fact, local optimization algorithms are not well suited, because of their iterative nature, while global heuristics, although more scalable, are usually not considered as good candidates because of the size of the problem spaces to explore. However, taking advantage of the reduction of problem space that we have evidenced, they could be, as described in the following.

3 Using Genetic Algorithms in the Reduced Problem Space

Currently, there exist only few software that do graph ordering in parallel, and their quality is not equivalent to the one of sequential algorithms. For instance, PARMETIS [9] implements a parallel version of a FM algorithm to refine its bipartitions but, in order to relax the strong sequentiality constraint of the algorithm when moving vertices that have neighbors on other processors, only such moves that improve the quality of the solution are accepted, therefore limiting the hill-climbing feature of the FM algorithm and reducing further the quality of the solutions as the number of processors (and thus, of potential distant neighbors) increase.

To avoid this intrinsic sequentiality problem, we have decided to turn to a completely different class of algorithms. Genetic algorithms (GA) are highly scalable meta-heuristics which allow to solve multi-criteria optimization problems using an evolutionary method. It is an iterative method that consists in simulating the evolution of a population of individuals which represent solutions to the problem, selecting best-fitting individuals as candidates for breeding the next generation. GA are known to converge very slowly and cannot therefore be applied to large graphs [1,3], but might be of use in the reduced problem spaces of band graphs. In the graph separation problem, every vertex can belong to three different domains: the separator, or any of the two separated parts. Therefore, every individual in the population is implemented as a linear array, similar in principle to a chromosome, which associates a number between 0 and 2 to any graph vertex index.

The reproduction operator is a classical multi-points cross-over operator, which is applied at a randomly-selected position of two mated individuals, and swaps one part of their arrays to produce two descendants. The mutation operator consists in swapping the part of randomly chosen vertices on some individuals. Since these naive operators cannot enforce that the crossed-over and mutated individuals be valid solutions, they are post-processed with a consistency-checking phase which adds vertices to the separator whenever necessary, and removes unneeded separator vertices.

Individuals are evaluated by means of a fitness function, which linearly combines dimensionless numbers such as the ratio of graph vertices that belong to the separator, the imbalance between the two parts, and the ratio of graph edges that link separator vertices. The first generation is made up of individuals that are mutations of the projected partition, plus some entirely random individuals which provide genetic diversity. To select and mate individuals, we have implemented several classical algorithms [7,11]. Although all methods behave quite similarly, best results were achieved with a mix of the elitism and roulette methods: the 5% best individuals are kept unconditionally, and each of the remaining ones is kept with a probability proportional to its fitness. Then, individuals are mated by pairs of descending fitness, and bred so as to keep constant population.

In order to increase concurrency in the GA algorithm, all of the individuals that are located on the same processor are considered as an isolated population

Table 4. OPC of the reordered *bcsstk29* matrix when multi-level band GA is used for all levels of nested dissection. Classical multi-level FM yields an OPC of $3.43e+8$ in 0.74s.

Deme size	**# Demes**	**Generations**	**OPC**	**Time** (s)
40	1	25	5.322334e+08	4.05
80	1	25	5.370016e+08	7.95
80	1	100	4.355475e+08	25.72
40	2	25	4.653384e+08	6.61
40	2	100	4.569806e+08	20.17
80	8	100	3.751443e+08	50.90

(also called "deme") living on an island [15]. Only occasionally can a few "champions" move from one island to another, to propagate their successful chromosomes into other populations which can have been trapped in local optima. In our current sequential implementation, every deme is handled by a different thread. Migration is performed when the variety of the population in some deme decreases, *i.e.* when individuals are too similar to their local champion.

To evaluate the convergence speed of our GA algorithm, we have computed nested dissection orderings of several test graphs with our multi-level band GA method. All of our tests were run on the M3PEC machine of the Université Bordeaux I, an eleven-node IBM machine with eight 1.5 MHz dual-core processors and 32 GB of memory per node. Since our current implementation is thread-based only, timings of tests involving more than sixteen threads (written between parentheses) are estimated: these tests are still run on a single SMP node, with as many threads per core as necessary, and the running time is divided by the appropriate ratio. PARMETIS, however, uses MPI, and runs fully in parallel.

Table 4 provides some results for graph *bcsstk29*. These results show that GA converges quite well, and that quality can be improved by increasing computation time and/or population size. As expected, running times are high, but GA are highly scalable, so that computation time can be reduced by adding processors, and partitioning quality can be increased by giving more time.

The second class of experiments that we have run aimed at evaluating the scalability of our method in terms of quality and running time. In order to compare our ordering software to PARMETIS in similar conditions, we ran our method on numbers of processors p that are powers of two (while our method does not require it), and performed band GA on the first $\log_2(p)$ levels only, using band FM afterwards; we will refer to this method as "limited GA" (LGA) in all of the following. When running GA, the population is evenly spread on all of the threads, with at least 100 individuals on the whole and at least 25 individuals per deme; therefore, above 4 threads, the population doubles along with the number of threads.

Our results, which are summarized in Table 5, are extremely encouraging. First of all, partitioning quality is not degraded too much when the number of processors increases: on our worst case, *bmw32*, we loose about 60% in OPC quality between 1 and 64 processors, and the quality is almost constant for *coupole8000*. Above

Table 5. Comparison between PARMETIS (PM) and our multi-level limited band genetic algorithm (LGA) for several graphs. C_{LGA} and C_{PM} are the OPC for LGA and PM, respectively. Dashes indicate abortion due to memory shortage. LGA timings between parentheses are extrapolated times for cases requiring more than 16 threads, as we had to run several threads per core on a single SMP node. Timings for PARMETIS are provided for graph *altr4* to give an idea of its speed, but t_{PM} and t_{LGA} cannot be compared, because PM is a fully parallel program, while our LGA testbed is the purely sequential nested dissection routine of SCOTCH, which has been parametrized so as to run the multi-threaded LGA algorithm only during the uncoarsening phases of the first $\log_2(p)$ stages of the nested dissection process.

Test case	**Number of processors or threads**						
	1	2	4	8	16	32	64
bcsstk32							
C_{LGA}	1.60e+9	1.55e+9	1.67e+9	**1.82e+9**	**1.83e+9**	**1.53e+9**	**2.07e+9**
C_{PM}	**1.29e+9**	**1.55e+9**	**1.62e+9**	3.09e+9	4.11e+9	5.85e+9	4.01e+9
t_{LGA}	0.42	0.88	0.84	0.97	2.07	(2.86)	(4.06)
audikw1							
C_{LGA}	**5.68e+12**	**5.91e+12**	**5.70e+12**	**5.82e+12**	**5.99e+12**	**6.44e+12**	**6.02e+12**
C_{PM}	–	–	–	7.78e+12	8.88e+12	8.91e+12	1.07e+13
t_{LGA}	19.78	22.77	29.55	32.89	60.24	(74.64)	(91.78)
bmw32							
C_{LGA}	3.04e+10	3.44e+10	**3.75e+10**	**4.13e+10**	**4.64e+10**	**4.57e+10**	**5.01e+10**
C_{PM}	**2.84e+10**	**3.22e+10**	4.09e+10	5.11e+10	5.61e+10	5.74e+10	6.31e+10
t_{LGA}	1.69	1.79	2.48	2.36	3.67	(5.11)	(7.80)
altr4							
C_{LGA}	**3.46e+8**	**3.71e+8**	**4.23e+8**	**4.06e+8**	**4.31e+8**	**4.92e+8**	**4.71e+8**
C_{PM}	4.25e+8	4.20e+8	4.49e+8	4.46e+8	4.64e+8	5.03e+8	5.16e+8
t_{LGA}	0.65	1.78	2.25	1.95	3.36	(5.43)	(7.20)
t_{PM}	0.58	0.31	0.20	0.13	0.11	0.27	0.31
conesphere1m							
C_{LGA}	**1.90e+12**	**1.92e+12**	**1.99e+12**	**2.37e+12**	**2.34e+12**	**2.53e+12**	**2.63e+12**
C_{PM}	2.04e+12	2.20e+12	2.46e+12	2.78e+12	2.96e+12	2.99e+12	3.29e+12
t_{LGA}	44.03	69.66	86.47	90.44	120.87	(134.85)	(158.07)
coupole8000							
C_{LGA}	**7.64e+10**	**7.64e+10**	**7.62e+10**	**7.65e+10**	**7.66e+10**	**7.68e+10**	**7.66e+10**
C_{PM}	–	–	–	8.17e+10	8.26e+10	8.58e+10	8.71e+10
t_{LGA}	125.69	75.40	55.19	49.16	52.59	(61.93)	(77.26)
thread							
C_{LGA}	4.10e+10	3.99e+10	**4.41e+10**	**4.64e+10**	**4.43e+10**	**4.59e+10**	**5.19e+10**
C_{PM}	**3.65e+10**	**3.98e+10**	6.60e+10	1.03e+11	1.24e+11	1.53e+11	1.28e+11
t_{LGA}	0.56	2.33	3.10	2.93	4.22	(5.02)	(5.92)

8 processors, our results clearly outperform the ones of PARMETIS, by a factor greater than two for *thread*. As expected, the higher the degree of the graph is, the bigger the difference is, because PARMETIS can only do gradient local optimizations on nodes which have neighbors on other processors.

Partitioning times are very good, too. Although the running time of a single sequential band GA refinement algorithm is between 30 and 80 times higher than the one of its sequential band FM counterpart, the overall running time of our LGA ordering program does not increase too much when the number of processors increase. While a doubling of the number of processors implies the turning of a whole level of band FM refinements into band GA refinements, the running time of LGA increases reasonably along with the number of threads, because when the number of processors increases it is levels of smaller subgraphs that are passed to the GA, which only results in a limited increase in the overall running time compared to the time taken by the first GA levels. Much hope is therefore placed in the development of a fully parallel, distributed-memory LGA algorithm.

4 Conclusion and Future Work

In this paper, we have presented a constrained banding approach which dramatically decreases problem size during the refinement phase of multi-level partitioning schemes. This method, which can be used with any refinement algorithm, allows us to take advantage of heuristics which are usually too expensive to be considered, such as genetic algorithms. We have implemented a shared memory multi-threaded GA, and tried it on numerous test cases. Although our GA is slower than distributed FM-like algorithms, it is scalable and provides better results, and its quality can be parametrized more easily (in terms of population size and of number of generations) to account for eventual time or quality constraints.

We are currently developing a distributed memory version of our GA algorithm, based on MPI, which will allow us to run tests on a larger number of processors, and to investigate the limits of using GA as a band refinement method for very large graphs. Since the testbed that we will use for this new version will be the parallel ordering routine of PT-Scotch, we will be able to compare its running time with the one of other parallel ordering software. Moreover, in order to have a reference for the quality of orderings, we are also currently completing the coding in PT-Scotch of the centralized band FM refinement algorithm described at the end of Section 2, which will allow us to compute, in a semi-parallel fashion, high quality orderings of very large graphs.

References

1. S. Areibi and Zeng Y. Effective memetic algorithms for VLSI design automation = genetic algorithms + local search + multi-level clustering. *Evolutionary Computation*, 12(3):327–353, 2004.
2. S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, 1994.
3. T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. Comput.*, 45(7):841–855, 1996.

4. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. 19th Design Automat. Conf.*, pages 175–181. IEEE, 1982.
5. A. George and J. W.-H. Liu. *Computer solution of large sparse positive definite systems*. Prentice Hall, 1981.
6. B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of Supercomputing*, 1995.
7. J. Horn, N. Nafpliotis, and D. E. Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. In *IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, 1994.
8. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Scientific Computing*, 20(1):359–392, 1998.
9. MeTiS: Family of multilevel partitioning algorithms. `http://glaros.dtc.umn.edu/gkhome/views/metis`.
10. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, February 1970.
11. P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts, towards memetic algorithms. Technical Report 826, California Intitute of Technology, Pasadena, CA 91125, U.S.A., 1989.
12. Scotch: Static mapping, graph partitioning, and sparse matrix block ordering package. `http://www.labri.fr/~pelegrin/scotch/`.
13. K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel algorithms for multiconstraint graph partitioning. In *Proceedings of EuroPar*, pages 296–310, 2000.
14. H. D. Simon and S.-H. Teng. How good is recursive bipartition. *SIAM J. Sc. Comput.*, 18(5):1436–1445, 1995.
15. D. Whitley, S. Rana, and R. B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–47, 1999.