A k-way Greedy Graph Partitioning with Initial Fixed Vertices for Parallel Applications

Maria Predari, Aurélien Esnard

A k-way Greedy Graph Partitioning with Initial Fixed Vertices for Parallel Applications

Maria Predari
Univ. of Bordeaux, LaBRI, UMR 5800,
HiePACS Project, INRIA,
F-33400 Talence, France.
maria.predari@inria.fr

Aurélien Esnard
Univ. of Bordeaux, LaBRI, UMR 5800,
HiePACS Project, INRIA,
F-33400 Talence, France.
aurelien.esnard@labri.fr

***Abstract*—Graph partitioning is used to solve the problem of distributing computations to a number of processors, in order to improve the performance of time consuming applications in parallel environments. A common approach to solve this problem is based on a multilevel framework, where the graph is firstly coarsened to a smaller instance and then it is partitioned in a number of parts using recursive bisection (RB) based methods. However, in applications where initial fixed vertices are used to model additional constraints of the problem, RB based methods often fail to produce partitions of good quality. In this paper, we propose a *new* direct k-way greedy graph growing algorithm, called KGGGP, that overcomes this issue and succeeds to produce partition with better quality than RB while respecting the constraint of fixed vertices. In the experimental section, we present results which compare KGGGP against state-of-the-art methods for graphs available from the popular *DIMACS'10* collection.**

***Index Terms*—high-performance computing; graph partitioning; multi-level framework; parallel simulations;**

I. Introduction

The increasing complexity of modern applications, executed on parallel architectures often dictates an efficient decomposition of the computational load in order to ensure high performance. In literature, many applications that arise in scientific computing [1], circuit design [2] or database modeling [3] use graph theory to describe and solve the problem of distributing computations to a number of available processors.

More precisely in a graph, a vertex represents a computation while an edge represents data dependencies between computations. Additionally, weights may be assigned to vertices and edges to further quantify their values. Hence, the problem of distributing computations becomes the partitioning problem of how to divide vertices of a graph in k parts of roughly equal size, such that the number of edges connecting vertices in different parts is minimized (edgecut minimization). Parts are then assigned to a number of available processing units in parallel computers. Nowadays, the most common approaches to solve the graph partitioning problem are based on the multilevel approach to compress the problem and on the recursive-bisection heuristic to solve it on a smaller instance.

In this paper, we focus on a variant of the classic graph partitioning problem, that is *the graph partitioning problem with fixed vertices*. It typically appears when the underlying application imposes additional constraints on the assignment of some computations to certain computers (processors) of the parallel environment. That is to say, some vertices of the graph, which we refer to as *fixed* vertices, are assigned a priori to some parts with the condition that when the partitioning is finished, they remain in place. Note also that in the remainder of the paper, we refer to vertices that are not fixed as *free*.

In the bibliography, problems that use the fixed vertex paradigm are drawn from various areas such as load balancing or VLSI CAD and are mainly formulated with variants of graph partitioning. For instance, a widely used algorithm for circuit design, called the top-down placement, is based on hypergraph[1] partitioning. Related study [2] demonstrates the importance of modeling the above problem with fixed vertices, proposing new partitioning heuristics which account for extra constraints. The authors suggest that the presence of fixed vertices models external dependencies or estimates on the positions of unplaced terminals reflecting more accurately the complex nature of the problem.

Moreover, a well-known example where the paradigm of fixed vertices occurs is the load balancing of adaptive scientific computations [5]. In such applications the discretization of the computational domain changes over time, leading to imbalanced load even for initially well-balanced simulations. The above feature gives rise to the repartitioning problem that is the problem of how to maintain dynamically changing load balance in a parallel application. The additional requirement of repartitioning is to minimize the migration volume for moving data among processors (parts). For this reason, the graph is enriched with one fixed vertex per part, along with (migration) edges connecting each fixed vertex with all the free vertices of its respective part. Thus, a good approach to solve the repartitioning problem is to perform a biased partitioning of the enriched graph, minimizing migration costs [6], [7].

A. Contributions

Our main contribution is a new algorithm, KGGGP, inside a multilevel framework that finds a direct k-way graph partitioning, extending a classic greedy approach for bipartitioning. Though KGGGP addresses the general problem of

[1]Hypergraph is a generalization of graph wherein edges may connect more than two vertices [4].

graph partitioning, its merit results appear mainly when fixed vertices are used during the partitioning procedure. Indeed, in the graph partitioning problem with initial fixed vertices, KGGGP exhibits the best partitioning quality (on edgecut computations) compared to state-of-the-art partitioning tools, that often use Recursive Bisection (RB) based techniques. For classic graph partitioning, RB remains one of the best choices.

The rest of the paper is organized as follows. In section II, we first give a motivating example, that illustrate why RB-based methods fail to handle fixed vertices. Then, in section III we review state-of-the-art partitioning techniques as well as existing work for partitioning with fixed vertices. In section IV, we present the KGGGP algorithm and in section V we confirm our observations presenting experiments performed on well known graph benchmarks. Finally, we conclude our results in section VI.

II. Issues of Recursive Bisection In partitioning with Fixed Vertices

The motivation behind our study comes from the observation that RB based algorithms perform rather poorly when the fixed vertex paradigm is involved, a remark which is also mentioned in [8]. Here, we attempt to further explain the above behavior.

In particular, RB based methods work as follows: the original graph is first split in two parts (bisection) and the above procedure is recursively repeated until the desired number of parts is acquired. Note that at each step of the recursion, a bisection is computed by placing parts together based on an inherent numbering constraint. The constraint implies that at each t step, two part subsets are created that contain parts $[1, 2^{t-1}]$ and $[2^{t-1}+1, 2^t]$ respectively. However, when the part numbering of fixed vertices opposes to that of RB, the method can not successfully respect both the inherent numbering constraint and the additional constraint of fixed vertices leading to largely disjoint parts.

In Figure 1, we illustrate a simple but compelling example that exhibits the partitioning issues emerging when a RB algorithm is used within fixed vertices. More precisely, as one may see in 1a, we use a simple two-dimensional grid (of dimensions 1000×1000) with an initial part numbering of fixed vertices, such that vertices near the corners are assigned accordingly to 4 different parts. We consider the rest of the vertices as free (part -1). Following, we partition the whole graph in 4 parts. We compare two different methods, within the SCOTCH multilevel framework [9], tuned with the same parameters. Both of them directly find a k-way partition during the initial partitioning phase of the framework (see III-B) but the first one uses a RB based partitioning method (implemented by SCOTCH) while the second one uses the KGGGP method provided by the authors.

In figures 1b, we present the results before the uncoarsening (refinement) phase obtained by the RB partitioning tool, while in 1c, one may see the same results obtained by our own KGGGP algorithm. Here, one may clearly see that during the first recursion level of RB it is not possible to select a good bisection between parts $[1, 2]$ and $[3, 4]$ that also respect the constraint of initial fixed vertices. Note that even after the uncoarsening phase, where the refinement algorithm takes place, the final partition quality will remain considerably poor for the RB based method.

Note that the above example is just an illustration of the problematic behavior of RB methods with initial fixed vertices. Further experiments that confirm our observations follow in section V.

III. Related Work

In this section, we first present useful graph definitions and formal statements concerning the partitioning problem, and then we review existing related studies about the graph partitioning problem with fixed vertices.

A. Graph Definitions

Consider a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. Each vertex $u \in V$ has a weight $w(u)$ representing the computational load at each processor, while each edge $e \in E$ has a weight $w(e)$ representing the communication cost between different processors. $P = (V_1, V_2, \ldots, V_k)$ is a k-way partition of G if the following conditions hold: each part $V_i, 1 \leq i \leq k$ is a non empty subset of V, parts are pairwise disjoint ($V_i \cap V_l = \emptyset$ for all $1 \leq i, l \leq k$) and union of k parts is equal to V. When a vertex v is assigned to a part V_i, we note $part[v] = i$ its part number. A partition is considered balanced if each part V_i respects *the balance criterion*:

$$W_i \leq W_{avg}(1 + \epsilon) \text{ for } i = 1, \ldots, k.$$

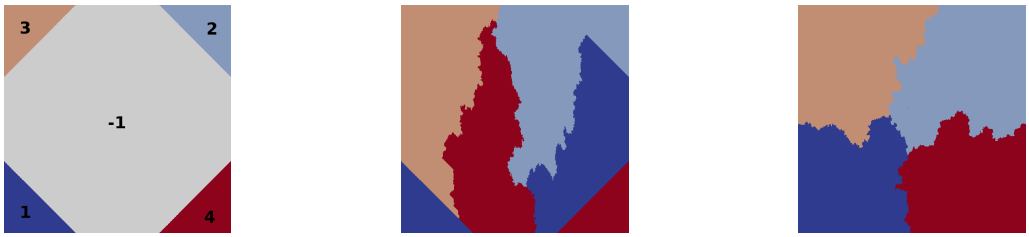
Weight W_i is defined as the total vertex weight in part V_i while $W_{avg} = \sum_{u_t \in V} w(u_t)/k$ represents the perfect load balance for all parts in G. Note that ϵ denotes the maximum imbalance tolerance allowed, where a typical value is $\epsilon = 5\%$ of the ideal weight. The *edgecut* of a partition is the weight sum of all edges whose incident vertices belong to different parts. The edgecut metric is known to approximate the total communication volume [10].

Hence, the classic objectives of a graph partitioning problem is to minimize the edgecut of the graph, while the balance constraint is satisfied for all parts.

B. Multilevel Framework

Despite the computational complexity of the graph partitioning problem (NP-complete [11]), many heuristic algorithms have been proposed in the past that find reasonably good partitions. Among them, greedy algorithms that add one by one vertices to parts or spectral methods [12] that use algebraic properties to perform the partitioning.

However, nowadays the most common approaches to solve the graph partitioning problem are based on a multilevel framework, where the initial graph is approximated by a sequence of smaller graphs [13]. The algorithm is divided in three phases: the coarsening, the initial partitioning and the uncoarsening phase.



(a) Initial fixed vertices. (b) RB (c) KGGGP

Fig. 1. Given initial fixed vertices, comparison of two different initial partitioning methods (RB and KGGGP) in a multilevel framework, before the refinement phase. Partition in 4 of a simple two-dimensional grid: the RB method fails to extend an initial partition when the part numbering is not appropriate, while the KGGGP method succeeds.

The main idea of the multilevel framework is to reduce the size of the graph, find a partition for the coarsest graph and project it back to the original one. Here, we give a brief description of the multilevel framework, also called *V-cycle*.

During the coarsening phase, a sequence of smaller graphs $G_1, G_2, \ldots, G_t$ is constructed from the original graph $G = G_0 = (V_0, E_0)$, such that $|V_i| < |V_{i-1}|$ for $i = 1, \ldots, t$. The goal is to contract edges, merging the adjacent vertices into a new supervertex and update the weights in the coarser graph. The above phase terminates when the reduction of successively coarser graphs becomes small enough (e.g., $|V_i| \leq 10 \times k$). The time complexity of this step is $O(|E|)$ for the mainly used *heavy edge matching* heuristic [14].

The algorithm that is used to partition the coarsest graph during the multilevel framework can be any partitioning algorithm, including bisection methods, spectral methods, greedy methods or geometric ones. The time complexity of this phase is often considered negligible compared to the other phases, since the size of the coarsest graph is smaller ($O(k)$).

Finally, the uncoarsening of a partition is quite simple since we just need to assign a vertex of the finer graph to the same part as its counterpart vertex in the coarser graph. However since finer graphs in the sequence have more degree of freedom implying further edge cut minimization, we usually apply local refinement algorithms (bisection refining). A class of local refinements that have good results and thus being used widely in partitioning tools is based on the algorithm of Kernighan-Lin (KL) [15] and Fidducia-Mattheyses (FM) [16], known to have a time complexity in $O(|E|)$ for the best methods.

Initially, the multilevel framework was only used to compute a 2-way partitioning (bisection) faster, by compressing the initial problem to solve a smaller one. Subsequently, the same scheme was applied recursively on each subgraph in order to compute a k-way partitioning. This *multilevel recursive-bisection* framework (MLRB) was first introduced in [13] and requires $\log_2(k)$ steps to compute the final k-way partition and a total of $k-1$ V-cycles (assuming k is a power of 2). The time complexity of MLRB is $O(\log(k)|E|)$.

More recently, another class of multilevel algorithms, called *multilevel k-way* (MLKW), propose to construct a k-way partitioning of a graph directly, that is, within the initial partitioning phase of the multilevel framework. Here, there is a single V-cycle: after the coarsening phase, the coarsest graph is directly partitioned into k parts and then it is projected back to the original graph. Note that refining a k-way partitioning is considerably more complicated than local 2-way refinement, so the uncoarsening phase of k-way direct partitioning algorithms is usually more time consuming [14]. More precisely, the time complexity for MLKW is dominated by the uncoarsening step, that is $O(k|E|)$ in general and $O(|E|)$ for KMETIS thanks to an highly-optimized k-way FM refinement heuristic [14].

C. Graph Partitioning Algorithms with Fixed Vertices

In Table I, we present some useful information about widely used graph and hypergraph partitioning tools, such as SCOTCH, METIS or PATOH. Our first remark is that despite the research interest on the graph partitioning problem with initial fixed vertices, more than half of the given tools do not handle at all the above problem. Note that this is especially true for graph partitioning tools. More importantly, as it is shown in the motivating example in Sec. II, we notice that even the tools which provide such algorithms do not successfully minimize the edgecut, resulting most of the times in low quality partitions. As we mentioned before, we locate the problem in the extended use of RB based algorithms in many partitioning tools, which work fine in the classic graph partitioning problem but fails to properly handle the fixed vertex paradigm.

Nevertheless, there are some interesting studies about partitioning algorithms that successfully handle initial fixed vertices, and we briefly review them here.

First, we review the work introduced in KPATOH [8], where a multilevel direct k-way hypergraph partitioning with fixed vertices is detailed. The authors identify the inferior performance of RB when fixed vertices are involved, mentioning the problem of relabeling parts during partitioning and they propose a new multilevel direct k-way algorithm that correct the above deficiency. They start by modifying the coarsening phase in order to respect initial fixed vertices, such that fixed vertices assigned to different parts can not be matched together. During the initial partitioning phase, the main idea is to remove completely the fixed vertices, partition the coarsest graph with RB and reassign them to the

TABLE I
GRAPH AND HYPERGRAPH PARTITIONING TOOLS.

Tools	Type	Fixed	Parallel	Scheme	Initial Part.	Available
METIS [17]	graph	no	no	MLRB	–	source
KMETIS [14]	graph	no	no	MLKW	RB	source
ParMetis [17]	graph	no	yes	MLKW	RB	source
Scotch [9]	graph	yes	no	MLKW	RB	source
PT-Scotch [9]	graph	no	yes	MLRB	–	source
RM-Metis [6]	graph	only k	no	MLKW	greedy	no
KaFFPa [18]	graph	no	no	MLKW	RB	source
Chaco [13]	graph	no	no	MLRB	spectral	source
HMetis [17]	hypergraph	yes	no	MLRB	–	binary
KHMetis [17]	hypergraph	no	no	MLKW	RB	binary
PATOH [19]	hypergraph	yes	no	MLRB	–	binary
KPATOH [8]	hypergraph	yes	no	MLKW	RB*	no
ZOLTAN (PHG) [20]	hypergraph	yes	yes	MLRB	–	source

resulting partition. They formulate the problem of reassigning fixed vertices to parts as a maximum weight bipartite graph matching problem, where the edge cut remains minimized. We refer to this method in table I as RB*. Finally, they use k-way refinements during the uncoarsening phase with the condition that fixed vertices are locked to their parts. Their experiments with fixed vertices show an edgecut improvement compared to the multilevel RB-based method used in PATOH. However, as it is mentioned in table I their implementation, KPATOH is not publicly available, therefore in the experiments we can only compare their result indirectly with PATOH.

Another related algorithm that addresses the repartitioning problem is proposed in RM-METIS, where the adaptive object space decomposition is modeled as a graph instance [6]. Here, the solution proposed is a multilevel direct k-way graph repartitioning approach, that handles fixed vertices, and that is not based on RB for the initial partitioning phase. As the previously mentioned approach, they reformulate the coarsening phase to respect fixed vertices and during the uncoarsening phase they use k-way refinements. However, during the initial partitioning phase, they employ a greedy graph growing approach using $k-1$ growing parts and a shrinking one. Note that RM-METIS has limitations to the number of initial fixed vertices that may be used during the partitioning, namely one per each part, whereas KGGGP has no such restrictions. Unfortunately, like with KPATOH, the implementation of RM-METIS is not publicly available as far as we know.

IV. THE KGGGP ALGORITHM

In this section, we describe a direct k-way graph partitioning algorithm, called KGGGP (k-way greedy graph growing partitioning), which can be easily integrated in a multilevel framework and that successfully handles any number of initial fixed vertices.

To begin with, we briefly describe here the standard greedy approach for bipartitioning [21–23], that has served as key idea for many partitioning algorithms and particularly for KGGGP. The standard greedy algorithm starts by placing two random "seed" vertices into the two parts. Subsequently, the vertices are added alternately to the parts, selecting each time the vertex whose displacement results in minimizing a selected criterion.

A. Algorithmic description of KGGGP

The KGGGP algorithm is an extension of the standard greedy bipartitioning algorithm for a k-way graph partitioning, where a partitioning of k parts (instead of just two) is directly computed. In a certain way, KGGGP can be seen as a $k+1$ FM (Fidducia-Mattheyses) refinement where the additional part, called *shrinking part* and denoted as -1, initially contains all (free) vertices and has to be empty at the end. Here, we give the detailed description of KGGGP in Figure IV-A.

As we mentioned above, greedy algorithms often use seeds to initiate the partitioning procedure, usually based on BFS (breadth first search) [24], however in KGGGP, the use of seeds is optional. As an alternative, the selected minimization criterion determines the first vertex displacement for each part.

The KGGGP algorithm selects the best *global* displacement (v, p), among all free vertices and all possible parts. Note that we consider all vertices in part -1 as candidates to move to any of the k parts and we choose the best displacement, based on an edgecut minimization criterion (gain), that must also respect the balance constraint. Despite the existence of multiple minimization criteria, in KGGGP we use the classic gain formula (# of internal edges $-$ # of external edges) also used in k-way FM refinement method [14]. Additionally to the above criteria, the algorithm enforces the selection of displacements where v is connected with vertices in p (connectivity constraint).

In order to quickly locate the best displacement, we use a similar data structure as in FM, adapted here for k parts. This structure, called *gain bucket data structure*, maintains a sorted list of displacement gains. To implement the bucket we use an array, whose i^{th} entry contains a doubly-linked list of all displacements with gain currently equal to i. Additionally, an array containing references of all displacements is used to perform quick gain updates, in the same way as in FM. More precisely, in KGGGP, a two dimensional array is employed which can be accessed by vertex and part numbering (v, p) allowing in constant time the updates of neighbor vertices after a displacement.

The KGGGP algorithm uses three instances of the gain bucket structure to store and select displacements: H_{REG} initially contains all possible displacements while H_{NCC} and H_{NBC} store displacements that do not respect the connectivity and balance constraint respectively. Note that H_{NBC} and H_{NCC} are at first empty. In the main loop of the algorithm, we search initially for displacements in H_{REG} and each time we encounter a displacement that do not respect the connectivity or the balance constraint we move it to the appropriate bucket (H_{NCC} and H_{NBC} respectively). It is expected that at a given time, bucket H_{REG} will become empty while buckets H_{NCC} and H_{NBC} contain the rest of possible displacements. In this case, until the algorithm terminates, we repeat the above procedure searching displacements first in H_{NCC} and if necessary move them in bucket H_{NBC}. Finally, when H_{NCC} becomes empty, we select accordingly displacements only from H_{NBC}. Once a displacement (v, p) is chosen (line 31), v

is moved to the corresponding part p (line 33) and then (v, p) is removed from the respective bucket (line 35). Additionally, we remove other possible displacements of the same vertex from any bucket (line 36) and we update the selection criterion of its neighbors as in the FM algorithm (line 39). Finally, since a vertex is moved to a part, we need to redetermine if displacements of neighboring vertices respect once more the connectivity constraint. In that case, those displacements are moved from H_{NCC} back to H_{REG} (line 44).

In figure IV-A, we illustrate the evolution of the part growing when a simple 100×100 grid is partitioned into 4 parts, using the KGGGP algorithm without a multilevel framework. As one may see, the algorithm aims to respect both the balance and the connectivity constraints leading to a rather balanced and connected partition even with no refinements 4d.

B. Fixed Vertex Management

As most greedy algorithms, KGGGP may easily handle initial fixed vertices. More precisely, fixed vertices in KGGGP are directly placed in their respective parts before the selection of displacements starts and are simply not considered as candidates in the initial gain bucket H_{REG}, remaining always in place.

C. KGGGP in a Multilevel Framework

The KGGGP algorithm can be easily integrated in a multilevel framework with some simple adjustment regarding the initial fixed vertices. Firstly, during the coarsening phase, an extra constraint is added, so that fixed vertices which belong to different parts can not be matched together, while they may be directly matched with free vertices. Following, we partition the coarsest graph with KGGGP as it is described above and we continue with the uncoarsening phase, where several k-way FM refinements (KFM) are performed for further edgecut minimization. Note that during this phase, we maintain all fixed vertices locked, forcing them to remain in place.

D. Time and Space Complexity

As the time complexity is concerned, the main steps of KGGGP algorithm consist of initializing displacements in $O(k|V|)$, selecting displacements in $O(|V|)$ and updating the bucket structures in $O(k|E|)$, for it is required to visit the neighborhood of each selected vertex for all parts. Therefore, the total complexity of KGGGP is $O(k|E|)$, considering that $|V|$ is dominated by $|E|$. As a reminder, the time complexity of RB is $O(log(k)|E|)$. Finally, the memory complexity is mainly due to saving all possible displacements in the gain bucket data structure, which is $O(k|V|)$.

E. Optimization: Local Greedy Approach

In order to reduce the total time complexity of KGGGP, we implement a second version of the method, where we enforce *local* selection of displacements instead of the *global* selection described above. The key idea here is to search for upcoming displacements only in the neighborhood of vertices that belong already to a part. This approach is similar to the one used in KMETIS to optimize the k-way FM refinement heuristic [14]. As a result, we do not need to initialize the H_{REG} bucket structure by computing *a priori* the gain value for all possible displacements. Instead, after a displacement (v, k) is chosen, we dynamically insert in H_{REG} new displacements (v', k) for all neighboring vertices v' of v, that remain free. If H_{REG} becomes empty while the partition is not complete, the method switches back to the global approach for all the remaining free vertices, given a time complexity of $O(k|E|)$ in the worst case and $O(|E|)$ in the best case.

TABLE II
LIST OF GRAPHS USED FOR EXPERIMENTS FROM THE POPULAR DIMACS'10 COLLECTION.

group	graph	# vtx	# edges	avg d°	min d°	max d°
walshaw	fe_rotor	99 617	662 431	13.30	5	125
walshaw	144	144 649	1 074 393	14.86	4	26
walshaw	wave	156 317	1 059 331	13.55	3	44
walshaw	m14b	214 765	1 679 018	15.64	4	40
matrix	audikw1	943 695	38 354 076	81.28	20	344
matrix	ecology1	1 000 000	1 998 000	4.00	2	4
matrix	thermal2	1 227 087	3 676 134	5.99	2	10
matrix	af_shell10	1 508 065	25 582 130	33.93	14	34
numerical	NACA0015	1 039 183	3 114 818	5.99	3	10
numerical	333SP	3 712 815	11 108 633	5.98	2	28
numerical	NLR	4 163 763	12 487 976	6.00	3	20
numerical	adaptive	6 815 744	13 624 320	4	2	4

V. EXPERIMENTS

In this section, we present the experimental results of KGGGP algorithm[2] and we compare them with some widely used partitioning tools, already described on Table I.

To do so, we implement two versions of the KGGGP algorithm inside the SCOTCH multilevel framework, one that follows the global greedy approach (KGGGP_G) and one with local greedy approach (KGGGP_L) as it is described in section IV-E. The two implementations of KGGGP use exactly the same partitioning parameters[3] as the other partitioning tools with which we compare our results. For some special parameters only available on certain tools, we keep the default values. The imbalance factor is set universally to 5%, as it is considered a rather acceptable imbalance tolerance.

Furthermore, we perform experiments on the partitioning quality and the time performance of KGGGP_G and KGGGP_L and compare them to SCOTCH 6.0.4 and KMETIS 5.1.0 when fixed vertices are not involved in the partitioning, and equally, to SCOTCH 6.0.4, PATOH 3.0 and ZOLTAN 3.81 (SIMPI) when fixed vertices are involved. Note that for the latter case (with fixed vertices), as far as we know, we compare KGGGP with all the available partitioning tools that solve the problem. Since we develop KGGGP_G and KGGGP_L inside the SCOTCH framework and since SCOTCH provides partitioning solution for fixed vertices problem, we decide to compare all our results relatively to SCOTCH. To

[2]The code of KGGGP is publicly available in *MetaPart* library http://metapart.gforge.inria.fr.

[3]Parameters: 4 iterations of initial partitioning, HEM for coarsening, maximum coarsest graph size equal to $30 \times k$, FM refinement with 10 passes and a maximum number of negative moves allowed set to 100 for each refinement pass.

Fig. 2. The KGGGP algorithm.

Input: graph $G = (V, E)$
Input/Output: partition array $part[]$ (of size $|V|$), initialized with fixed and free vertices
% initialization step of gain bucket structures (H_{REG}, H_{NCC} and H_{NBC})
$H_{REG} \leftarrow$ initialize with all displacements (v, p) of any free vertices v to any parts p
$H_{NBC} \leftarrow \emptyset$
$H_{NCC} \leftarrow \emptyset$
% main loop
while there are free vertices **do**
  % select the best displacement
  repeat
    if H_{REG} is not empty **then**
      $(v, p) \leftarrow$ consider a displacement with maximum gain from H_{REG}
      if part p is empty **then**
        choose (v, p)
      else if balance constraint is not respected for displacement (v, p) **then**
        move (v, p) from H_{REG} to H_{NBC}
      else if connectivity constraint is not respected for displacement (v, p) **then**
        move (v, p) from H_{REG} to H_{NCC}
      else
        choose (v, p)
      end if
    else if H_{NCC} is not empty **then**
      $(v, p) \leftarrow$ consider a displacement with maximum gain from H_{NCC}
      if balance constraint is not respected for displacement (v, p) **then**
        move (v, p) from H_{NCC} to H_{NBC}
      else
        choose (v, p)
      end if
    else if H_{NBC} is not empty **then**
      $(v, p) \leftarrow$ consider a displacement with maximum gain from H_{NBC}
      choose (v, p)
    end if
  until a displacement (v, p) is chosen
  % perform the chosen displacement (v, p)
  $part[v] \leftarrow p$
  % update buckets
  remove (v, p) from gain bucket structures
  remove (v, p') where $p' \neq p$ from gain bucket structures
  for all vertex v' adjacent to v **do**
    for all parts p' **do**
      update the gain of displacement (v', p') in gain bucket structures
      if $(v', p') \in H_{NCC}$ and $p' = p$ **then**
        move (v', p') from H_{NCC} to H_{REG}
      end if
    end for
  end for
end while

Fig. 3. Steps of KGGGP while partitioning a simple grid in 4 parts.

(a) Step 100.

(b) Step 3000.

(c) Step 6000.

(d) Final step.

perform the experiments, we use graphs from real life numerical applications of *DIMACS'10* collection [25], a publicly available collection for experimentation on graph partitioning and graph clustering problems. One finds in Table II some useful information, such as the total number of vertices or the average degree of the graphs used in our experiments.

For the sake of brevity and readability of our experimental results, each bar of the charts in Figures 4 and 5 represents the *average* value of each result for all graphs in our data set. Additionally, note that each experiment is performed 10 times for every graph. In the charts, we present results on edgecut and execution time, while the number of desired parts increases from 10 to 100.

In the first experiment (Fig. 4), we compare the KGGGP algorithms to SCOTCH and KMETIS on the classic graph partitioning problem, that is without using initial fixed vertices. The purpose of this experiment is to exhibit that even though KGGGP is not the best algorithm for the classic problem, the results we obtain are quite good. More precisely, KMETIS produces partitions with minimized edgecut followed by SCOTCH with a 5% increase of edgecut which overall, confirms (as expected) that RB based methods perform better than the greedy ones for this problem. However the edgecut increase for both KGGGP_G and KGGGP_L is also less than 5% more than that of SCOTCH which proves that KGGGP can be a fine partitioning choice even for classic graph partitioning problems. Additionally, on the results of the execution time, one may notice that the performance of KGGGP_G and KGGGP_L compared to SCOTCH and KMETIS becomes slower as the number of parts increases which is not surprising if we recall its time complexity compared to RB based algorithms. However, note that KGGGP_L obviously successes to reduce the execution time compared to KGGGP_G and is not much slower than SCOTCH or KMETIS.

Following, we present results with fixed vertices in Figure 5

from two experimental cases, each representing a different way to distribute the initial fixed vertices to the graph, before the partitioning. We denote these schemes *bubble* and *repart*. In the *bubble* scheme, we simply compute k initial seeds as explained before (based on BFS [24]), and we use each one as the center of a bubble in order to add more fixed vertices. Particularly each bubble, which may be seen as an initial part of fixed vertices, grows using the levelset algorithm until it reaches 20% of the desired part size. For the *repart* scheme, we follow the repartitioning scheme proposed by ZOLTAN [7], where an initial partition is used, and its total vertex weight is randomly modified in order to obtain 50% of load imbalance. Based on the above imbalanced partition, we build an enriched graph adding one single fixed vertex per part along with the migration edges that connect it with its respective part. Note that PATOH, and ZOLTAN are hypergraph partitioning tools, and do not handle simple graph structure. For these purpose we convert our test graphs to hypergraphs, but as expected we do not take into account this overhead in the total execution time of the algorithms. Consequently, KGGGP_G and KGGGP_L have the best partitioning quality among the other tools and obtain up to 20% increase on edgecut minimization for *bubble* scheme and around 10% for *repart* scheme compared to SCOTCH. Furthermore, ZOLTAN produces partitions with slightly better quality than SCOTCH for the *bubble* scheme, but exhibit a 10% increase of edgecut compared to SCOTCH for the *repart* scheme. As one may see, PATOH does not give good results for any of the experiments with fixed vertices. Regarding the performance results, one may notice that KGGGP_L is most of the times the fastest tool, except when the number of parts becomes large enough. Note also that KGGGP_G is always slower than KGGGP_L while PATOH and ZOLTAN are more than two times slower than the SCOTCH reference. The obvious reason for the above results is the more complicated structure of hypergraphs which impose the use of more time consuming partitioning algorithms.

VI. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we revisit the belief that RB based methods, which are extensively used for classic graph partitioning problem, fall short when initial fixed vertices are involved in the partitioning. Considering the above argument, we present here a new graph partitioning method integrated in the SCOTCH multilevel framework that solves the above problem.

More particularly, the KGGGP algorithm is a direct k-way greedy graph growing partitioning algorithm that successfully handles initial fixed vertices and produces partitions of better quality. Indeed, after performing experiments on real-life graph applications, where we evaluated all available partitioning tools that handle fixed vertices, we concluded that KGGGP manages to minimize the edgecut and achieve, almost always, good execution time. The above results lead us to believe that RB based algorithms are not suitable for partitioning with initial fixed vertices, despite their good performance. Thus, we believe that k-way direct graph growing algorithms, like KGGGP should be used instead.

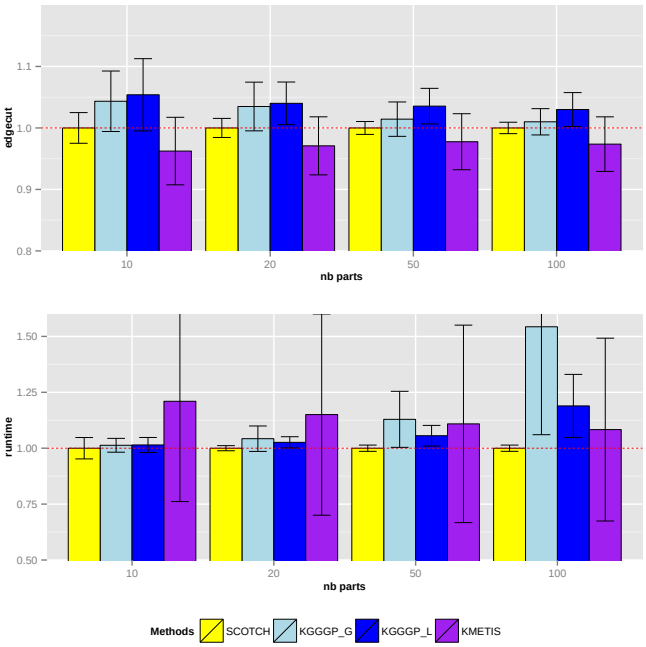


Fig. 4. All experimental results on classic partitioning (without fixed vertices) on quality and performance of KGGGP and other partitioning tools.

Finally, one of our future goals is to apply the KGGGP algorithm for the load-balancing of complex coupled simulation, using a recent technique proposed by the authors in [26], that is based on biased partitioning with fixed vertices. Since KGGGP is the most suitable partitioning algorithm for such problems, we expect better load-balancing results for coupled simulations.

REFERENCES

[1] J. D. Teresco, K. D. Devine, and J. E. Flaherty, "Partitioning and dynamic load balancing for the numerical solution of partial differential equations," in *Numerical Solution of Partial Differential Equations on Parallel Computers*, 2006, vol. 51, pp. 55–88.

[2] A. E. Caldwell, A. B. Kahng, A. A. Kennings, and I. L. Markov, "Hypergraph partitioning for VLSI CAD: Methodology for heuristic development, experimentation and reporting," in *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, ser. DAC '99, 1999, pp. 349–354.

[3] E. Demir, C. Aykanat, and B. Barla Cambazoglu, "Clustering spatial networks for aggregate query processing: A hypergraph approach," *Inf. Syst.*, vol. 33, no. 1, pp. 1–17, Mar. 2008.

[4] C. Berge, *Graphs and Hypergraphs*. Elsevier Science Ltd., 1985.

[5] B. Hendrickson and K. Devine, "Dynamic load balancing in computational mechanics," in *Computer Methods in Applied Mechanics and Engineering*, vol. 184, 2000, pp. 485–500.

[6] C. Aykanat, B. B. Cambazoglu, F. Findik, and T. Kurc, "Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids," *J. Parallel Distrib. Comput.*, vol. 67, pp. 77–99, January 2007.

[7] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdağ, R. T. Heaphy, and L. A. Riesen, "A repartitioning hypergraph model for dynamic load balancing," *J. Parallel Distrib. Comput.*, vol. 69, no. 8, pp. 711–724, 2009.

[8] C. Aykanat, B. B. Cambazoglu, and B. Uçar, "Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices," *J. Parallel Distrib. Comput.*, vol. 68, pp. 609–625, May 2008.

[9] F. Pellegrini, "SCOTCH," http://www.labri.fr/perso/pelegrin/scotch/.

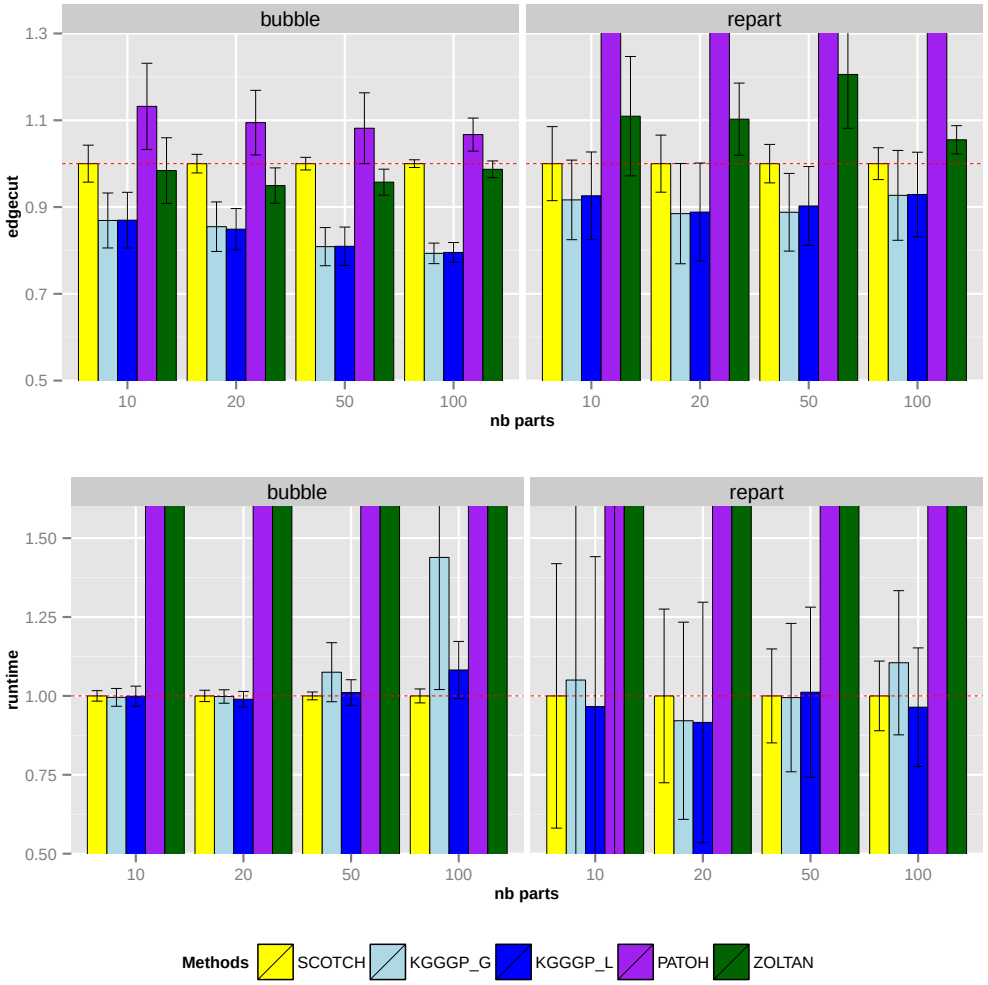


Fig. 5. All experimental results on partitioning with fixed vertices on quality and performance of KGGGP and other partitioning tools.

[10] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Comput.*, vol. 26, no. 12, pp. 1519–1534, 2000.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[12] B. Hendrickson and R. Leland, "An improved spectral graph partitioning algorithm for mapping parallel computations," *SIAM J. Sci. Comput.*, vol. 16, no. 2, 1995.

[13] R. Leland and B. Hendrickson, "A multilevel algorithm for partitioning graphs," in *1995 ACM/IEEE conference on Supercomputing*, 1995.

[14] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, pp. 96–129, 1998.

[15] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, pp. 291–307, February 1970.

[16] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," *19th Design Automation Conference*, pp. 175–181, 1982.

[17] G. Karypis, "METIS, HMETIS, PARMETIS," http://glaros.dtc.umn.edu/gkhome/metis.

[18] P. Sanders and C. Schulz, "Think Locally, Act Globally: Highly Balanced Graph Partitioning," in *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, ser. LNCS, vol. 7933. Springer, 2013, pp. 164–175.

[19] mit V. atalyrek and C. Aykanat, "PaToH: A Multilevel Hypergraph Partitioning Tool," http://bmi.osu.edu/~umit/software.html#patoh, 1999.

[20] "Zoltan: Parallel partitioning, load balancing and data-management services," http://www.cs.sandia.gov/Zoltan/Zoltan.html.

[21] J. Ciarlet, P. and F. Lamour, "On the validity of a front-oriented approach to partitioning large sparse graphs with a connectivity constraint," *Numerical Algorithms*, vol. 12, no. 1, pp. 193–214, 1996.

[22] R. Battiti and A. Bertossi, "Differential greedy for the 0-1 equicut problem," in *in Proceedings of the DIMACS Workshop on Network Design: Connectivity and Facilities Location*. American Mathematical Society, 1997, pp. 3–21.

[23] S. Jain, C. Swamy, and K. Balaji, "Greedy algorithms for k-way graph partitioning," in *the 6th international conference on advanced computing*, 1998.

[24] R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw, "Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM," *Parallel Computing*, vol. 26, no. 12, pp. 1555–1581, 2000.

[25] D. Bader, H. Meyerhenke, P. Sanders, C. Schulz, A. Kappes, and D. Wagner, "Benchmarking for graph clustering and partitioning," in *Encyclopedia of Social Network Analysis and Mining*, R. Alhajj and J. Rokne, Eds. Springer New York, 2014, pp. 73–82. [Online]. Available: http://www.cc.gatech.edu/dimacs10

[26] M. Predari and A. Esnard, "Coupling-aware graph partitioning algorithms: Preliminary study," in *21st International Conference on High Performance Computing, HiPC 2014, Goa, India, December 17-20, 2014*, 1–10, 2014.