A GRASP WITH PATH-RELINKING FOR THE k-WAY GRAPH PARTITIONING PROBLEM

Bruno Menegola, Marcus Ritt
Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
{bmenegola,mrpritt}@inf.ufrgs.br

RESUMO

O problema de particionamento balanceado de grafos consiste em encontrar uma partição de tamanho k dos vértices de um grafo, minimizando o número de arestas que participam do corte e sujeito a uma restrição de balanceamento, onde cada parte terá um tamanho máximo predefinido. O problema possui diversas aplicações, dentre elas o particionamento de circuitos VLSI, a otimização da comunicação em processamento distribuído e a decomposição de redes de transporte. Neste artigo propomos uma abordagem para produzir boas soluções utilizando a metaheurística GRASP com path-relinking para algumas soluções elite e também usando generalizações de operadores heurísticos utilizados no caso de biparticionamento, como a heurística construtiva Differential Greedy e a busca local Fiduccia-Mattheyses. Apresentamos resultados experimentais que mostram que nosso particionador é competitivo com os do estado-da-arte em alguns casos.

PALAVRAS CHAVE. Problema de particionamento de grafos em k-partes, GRASP, Path-relinking, Heurísticas, Otimização Combinatória, Metaheurísticas.

MH - Metaheurísticas

ABSTRACT

The balanced graph partitioning problem consists in finding a partition of size k for the vertices of a graph, minimizing the number of edges belonging to the cut and subject to a balance restriction, in which each part shall have a predefined maximum size. The problem has several applications, among them the partitioning of VLSI circuits, the optimization of communication in distributed computing and road network decomposition. In this paper we propose an approach for generating good solutions using the metaheuristic GRASP along with a path-relinking to some elite solutions and also using generalizations of heuristic operators proposed for graph bipartitioning, such as the constructive heuristic Differential Greedy and the Fiduccia-Mattheyses local search. We present experimental results that show that our partitioner is competitive with state-of-the-art in some cases.

KEYWORDS. k-way graph partitioning problem, GRASP, Path-relinking, Heuristics, Combinatorial Optimization, Metaheuristics.

MH - Metaheuristics

1. Introduction

The balanced graph partitioning problem consists in finding a partition of size k for the vertices of a graph, subjected to a balance restriction. This restriction limits the maximum cardinality of the size of each part. For such partitioning, the most common objective is to minimize the sum of the weights of edges that connect vertices in distinct parts. Other objective functions have been defined, such as total or maximum communication volume (Schloegel et al., 2003), but the minimum cut is the most researched function. In this paper we focus on the case where the graph is undirected and vertices and edges have unit weights.

The balanced graph partitioning is an NP-hard combinatorial optimization problem (Bui and Jones, 1992) and has several applications. Examples include partitioning of VLSI circuits, road network decomposition, domain decomposition for parallel computing, image segmentation, data mining, finding ground-state magnetization of spin glasses, matrix decomposition, among others (Chardaire et al., 2007).

Being a hard problem, exact solutions are found in reasonable time just for small graphs. Using optimized models of integer programming (Boulle, 2004), for example, experiments show that a state-of-the-art solver can compute a solution for a graph of 500 vertices and 625 edges in about 25 minutes. However, the applications of the problem require to partition much larger graphs and so heuristic solutions are usually indicated. There are some specific heuristics for the problem, some constructive, such as the Min-Max Greedy algorithm (Battiti and Bertossi, 1999) or the Differential Greedy heuristic (Battiti and Bertossi, 1997); and some of iterative refinement, such as the Kernighan-Lin local search (Kernighan and Lin, 1970), the Fiduccia-Mattheyses local search (Fiduccia and Mattheyses, 1982) or the Lock-Gain local search(Kim and Moon, 2004). Although they provide good results, the refinement algorithms are usually used as local search procedures for other methods, like metaheuristics. Several metaheuristic approaches have already been proposed: Genetic Algorithms (Kim and Moon, 2004), Tabu Search (Rolland et al., 1996), Greedy Randomized Adaptive Search Procedure (GRASP) (Laguna et al., 1994), Simulated Annealing (Johnson et al., 1989), Memetic Algorithms (Galinier et al., 2011) and others.

In the following sections we are going to present a new approach for the partitioning problem. The partitioner is based on a GRASP that makes use of known heuristics, such as the Differential Greedy construction and a Tabu based Fiduccia-Mattheyses local search (Galinier et al., 2011), along with two newly proposed path-relinking operators. Results show that our approach is competitive with state-of-the-art heuristic in some cases.

The paper is organized as follows: Section 1.1 presents some notation and a formal definition of the problem. Section 2 presents methods that will be used in our approach, which is described in Section 3. Experimental results are given in Section 4 along with an analysis of them. We finally state some conclusions and future research directions in Section 5.

1.1. Notation and formal definition

Consider an undirected graph $G = (V, E, c, \omega)$, with $n = |V|$ vertices and $m = |E|$ edges, where $c(u)$ and $\omega(\{u, v\})$ are non-negative weights of a vertex and an edge, respectively. Let B be a collection of k subsets $\{B_0, \ldots, B_{k-1}\}$ of V; and define the set $F = \{\{u, v\} \in E \mid u \in B_i, v \in B_j, 0 \leq i < j \leq k-1\}$ of cut edges. The problem of k-way graph partitioning with balance $1 \leq \epsilon < k$ can be described by the following model:

$$\begin{aligned}
\textbf{minimize} \quad & \sum_{\{u,v\}\in F} \omega(\{u,v\}) \\
\textbf{subject to} \quad & 0 < \sum_{u\in B_i} c(u) \leq \left\lfloor \frac{\epsilon}{k} \sum_{v\in V} c(v) \right\rfloor && \forall i \in [0,k-1] \\
& \bigcup_{0\leq i<k} B_i = V \\
& B_i \cap B_j = \emptyset && \forall i,j \in [0,k-1] \mid i \neq j
\end{aligned}$$

In this paper, all graphs will be unweighted, i.e., $c(u) = 1$ and $\omega(\{u,v\}) = 1$. The first restriction ensures that each part B_i does not exceed the size limit imposed by ϵ. The second and third restrictions ensure that B is a partition of V. Commonly $k = 2^l$, for some $l \in \mathbb{N}^+$ (usually $l \leq 6$), and so a bipartitioner could be defined and applied recursively, although the quality of the partitions tend to decrease with the increase of partition size (Pellegrini, 2007). If $\epsilon = 1$ the restriction for each part size should take into account the possibility of n being not divisible by k, and some parts may be bigger than others. A greater imbalance does not change the complexity class of the problem, provided that $1 \leq \epsilon < k$, i.e., the problem remains NP-hard (Bui and Jones, 1992).

2. Related approaches and state-of-the-art

The next sections are going to present known algorithms for the problem that we use as subprocedures of our approach, such as the constructive method Differential Greedy, the refinement method Fiduccia-Mattheyses and its Tabu variation. We also briefly present the metaheuristic GRASP and related attempts to build a partitioner.

2.1. Constructive algorithms

It is very easy to generate a feasible solution for the partitioning problem and there are several methods for such task. The trivial method is to generate a random solution: each vertex is assigned to a random part that does not violate its size restriction. This does not produce good solutions because it ignores the fact that real world graphs usually have an inherent structure that can be taken into account. A good constructive algorithm can generate cuts of expected size about a quarter of the expected size of a random partition for graphs with some structure (Karypis and Kumar, 1998; Battiti and Bertossi, 1997).

More robust constructive methods use spectral information of the graph (Karypis and Kumar, 1998) or a greedy function that grows the partition around seed vertices. One good heuristic is the Differential Greedy heuristic (Battiti and Bertossi, 1997) (Algorithm 1). New vertices are added to the parts based on the minimum difference between the number of edges that connect them to another part and the number of connections inside the part that it would be inserted (lines 5 and 6).

Lines 5 and 6 are critical but an efficient implementation of this algorithm uses a bucket structure to keep the gain values up-to-date (increases on the cut size induced by every unassigned vertex), as proposed by Fiduccia and Mattheyses (1982). This structure will allow a constant time access to the vertex that increases cut size the least, as well as eventual updates of neighbors of recently assigned vertices.

The presented algorithm builds perfectly balanced bipartitions of the input graph but it can be easily extended to k-way perfectly balanced partitioning. A critical issue arises for imbalanced partitioning because no assumptions can be made on which part may grow larger first. If one lets the parts grow to unequal sizes (within the desired imbalance), experience shows that worse solutions can be generated on average when compared with perfect balanced ones. Our partitioner does not try to create imbalanced initial partitions and just use the generated k-way perfectly balanced ones.

Algorithm 1: Differential Greedy constructive heuristic (Battiti and Bertossi, 1997)

Input: Graph $G = (V, E)$
Output: A bipartition of V

$u, v \leftarrow$ two random vertices $\in V \mid u \neq v$;
$A \leftarrow \{u\}$; $B \leftarrow \{v\}$; $V' \leftarrow V \setminus \{u, v\}$;
while $|V'| > 0$ **do**
    let $E(v, S) = |\{\{v, u\} \in E | u \in S\}|$;
    $m \leftarrow \min_{v \in V'}(E(v, B) - E(v, A))$;
    $C \leftarrow \{v \in V' \mid E(v, B) - E(v, A) = m\}$;
    $b \leftarrow$ random vertex $\in C$;
    $A \leftarrow A \cup \{b\}$;
    $V' \leftarrow V' \setminus \{b\}$;
    swap sets A and B;
return $\{A, B\}$;

2.2. Refinement heuristics

Suppose for a moment that we are interested in improving a perfectly balanced bipartition $\{A, B\}$ of a graph. One possible approach to refine the solution is swapping subsets $X \subset A$ and $Y \subset B$ (with $|X| = |Y|$) to opposite sides so that a reduction in cut size is achieved. However, finding these subsets is intractable just as the partitioning problem itself (Schloegel et al., 2003). With this in mind, some specific heuristics for the problem were proposed. One of great importance was the Kernighan-Lin (KL) local search (Kernighan and Lin, 1970) that later was practically replaced by that of Fiduccia and Mattheyses (1982). Although the latter uses the same heuristic, it is a major improvement over the former because it reduced time complexity from $O(n^2)$ to $O(m)$, by using a bucket structure, and also permits to generate imbalanced partitions. Fiduccia-Mattheyses (FM) local search is the refinement algorithm used in this paper, and therefore we will explain it next in more details.

A version of the FM algorithm for k-way partitioning works as follows (Osipov and Sanders, 2010). A candidate vertex is selected to change its part when it is expected to reduce the size of the cut. The candidates are ordered according to the gain that they induce when moved to a specific part. The gain function $g_i(v)$ quantifies the gain of moving a vertex $v \in B_j$ to a part B_i (with $i \neq j$):

$$g_i(v) = \left|\left\{\{v, u\} \in E \mid u \in B_i\right\}\right| - \left|\left\{\{v, u\} \in E \mid u \in B_j\right\}\right|.$$

In order to keep gains ordered, each part B_i has a heap that holds the vertices that can be moved to it. Among the possible moves to all parts, an iteration of the local search uses the vertex of largest gain whose move still satisfies all restrictions. It is not allowed to move any vertex more than once. The local search finishes when no vertex is eligible for moving (because of some restriction or since no more vertices are available) or when a stopping criterion is reached. This algorithm allows, at some point, movements that lead to worse cuts, but the best cut found is always restored at the end. An iterative improvement version of FM restarts the local search until no more improvements are made.

The stopping criterion is based on a random walk model (Osipov and Sanders, 2010) and interrupts the search when there is a low probability of returning to a state of improvement. If the move gains have expectation μ and variance σ^2, as observed on the last p steps (moves) until the last improvement, and, with this, it is unlikely that the local search will get to a new best solution if $p\mu^2 > \alpha\sigma^2 + \beta$, for some adjust parameters α and β,

FM local search does not allow to move any vertex more than once (we call a vertex that has been moved once locked). Galinier et al. (2011) proposes a different approach, based on Tabu Search. The variation basically locks the vertices for some iterations by adding them to a Tabu list

but allow them to be moved several times before some stopping criterion is reached. The search stops when a maximum number of iterations is reached or there are no more vertices allowed for moving. The number of iterations that a vertex should be locked is specified by a tenure function as follows: the function $\tau : \mathbb{N}^+ \to \mathbb{N}^+$ gives the tenure, based on a iteration i, and equals to $\tau(i)$; the function depends on a parameter $\texttt{maxT}$ that limits the maximum number of iterations a vertex will be tabu; it is a step function and, hence, it can be defined by, $\forall i \in \{x_j, \ldots, x_{j+1} - 1\} : \tau(i) = a_j$; it is also periodic and it defines 15 intervals (greater intervals are based on these, e.g., the 16th interval is equal the first one); the values are defined by $a_j = \texttt{maxT} \times b_j$, $x_1 = 1$, $x_{j+1} = x_j + 4 \times \texttt{maxT} \times b_j$ and $(b_j)_{j \in \{1,\ldots,15\}} = \frac{1}{8}(1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1)$. For example, if $\texttt{maxT} = 40$, we have $x = (1, 20, 60, 80, 160, \ldots)$ and $a = (5, 10, 5, 20, 5, \ldots)$, and a vertex that enters the tabu list at iteration $i = 25$ will be locked for $\tau(25) = 10$ iterations. Galinier et al. (2011) have demonstrated the usefulness of this tenure function, and we will use it in our approach.

2.3. Graph partitioners

Several metaheuristics have been proposed for this problem. One of the first methods, proposed by Johnson et al. (1989) was a simulated annealing algorithm that used Kernighan-Lin heuristic for local optimization.

The Greedy Randomized Adaptive Search Procedure (GRASP) methodology was developed in late 1980s by Feo and Resende (1989) and is a multi-start heuristic that in each iteration constructs an initial solution and refines it through a local search. The initial solution must have some random element such that the space of possible solutions can be explored properly. The constructive algorithm must be greedy, i.e., it chooses the best item for insertion in the solution, and adaptive, because this choice is made based on the previously inserted items. This methodology was later applied to perfectly balanced bipartitioning problem by Laguna et al. (1994). It also used the Kernighan-Lin algorithm for local search, but introduced the idea of constructive algorithms based on greedy functions.

The usage of a good constructive algorithm combined with metaheuristics were further analyzed in Battiti and Bertossi (1999) with a definition of a new constructive heuristic, the Min-Max Greedy, which is the base of Differential Greedy construction explained in Section 2.1.

Chardaire et al. (2007) present a new metaheuristic called PROBE for the graph partitioning problem, which has some similarities with GRASP. As part of the procedure, the authors introduced a method to use the information of two good partitions for an escape attempt of local minima. This method inspired our path-relinking (PR) approach. PR is an enhancement for GRASP that can lead to significant improvements in solution time and quality (Resende and Ribeiro, 2003). It explores trajectories connecting solutions to find better ones. Often one of them is chosen from a set of elite solutions found during the search. The idea for connecting two solutions is to find a path through their neighborhood by making small modifications on the first solution until an equivalent solution to the second one is reached. Only those modifications which make the first solution more similar to the second are allowed. If a solution better than both is found on this path, it substitutes the current solution. The methods of PR for the graph partitioning problem will be defined in the next section.

3. A GRASP for the k-way graph partitioning problem

In this section we propose a GRASP for the k-way graph partitioning problem. The GRASP requires two basic operators: constructive and refinement algorithms. We adapted the Differential Greedy algorithm (Section 2.1) for the k-way partitioning since it has the necessary elements to be an initial solution generator for this metaheuristic and provides good results compared to other generators as demonstrated in Battiti and Bertossi (1997). It is also the base of one of the path-relinking operators that it will be explained in Section 3.2. The refinement algorithm is the k-way

and tabu based adaptation of the Fiduccia-Mattheyses (FM) heuristic explained in Section 2.2. It appears in two forms: an iterative and a non-iterative procedure. The former uses the latter to refine a solution until no improvement is made.

In order to improve GRASP using path-relinking, it is required that we maintain some good solutions, the so-called elite, which serve as target solutions in path-relinking. In the next section we present how our elite pool handles new solutions and how they are classified as elite ones.

3.1. Elite solutions

In our approach, a solution is considered an elite when it is the best known solution or when it is different enough from the other solutions in the pool. The difference between solutions is quantified by a similarity measure.

The similarity corresponds to the number of vertices that do not need to change parts to achieve an equivalent partition. Given two partitions A and B and some bijection mapping the parts of A to the parts of B $\sigma : \{0, \ldots, k-1\} \rightarrow \{0, \ldots, k-1\}$, let $S(\sigma) = \{v \in V \mid v \in A_i, v \in B_j, \sigma(i) = j\}$, be the set of vertices that makes the two partitions similar based on the correspondence of parts imposed by σ. Two partitions are said to be equivalent when $|S(\sigma)| = n$. The critical issue in this case is defining σ when two partitions differ. We are in fact interested in the mapping σ_* that maximizes the similarity, i.e., $|S(\sigma_*)| = \max_\sigma(|S(\sigma)|)$. The difference between two partitions is then defined as $\Delta(A, B) = |S(\sigma_*)|/n$, while the similarity is $1 - \Delta$.

In order to define the mapping σ_*, a greedy algorithm that does not guarantee optimality is introduced in Galinier et al. (2011). We define σ_* optimally through a maximal matching in a bipartite graph defined as follows: each part A_i can be related to each part B_j; the weight $\omega(\{i, j\})$ of this edge corresponds to the number of similar vertices, i.e., $\omega(\{i, j\}) = |\{v \in V \mid v \in A_i, v \in B_j\}|$. The maximal matching is obtained via the Kuhn-Munkres (Hungarian) algorithm (Munkres, 1957) and the solution corresponds to the mapping σ_*. The complexity of this algorithm is $O(k^3)$.

As mentioned before, when a solution is not the best, only partitions different enough from the others are allowed to enter elite pool. When a new non best partition is found, we compute its difference to all elite solutions. If the minimum difference is greater than a parameter `EliteMinDiff` and its cut size is less or equal to that of the worst elite solution, we allow the current solution to enter elite pool. If the elite pool exceeds its maximum size, defined by a parameter `EliteMaxSize`, some solution is required to be dropped. The strategy is remove the solution with the minimum difference to the recently inserted one (or the second one, because we never drop the best solution). Since we have a constraint that allows a solution to be an elite only if it has a better cut than the worst one, it is not interesting to keep similar partitions.

Another rule defines which partitions should be used for the path-relinking. The selected solutions that will be part of it are chosen randomly between one of the best partitions in elite pool. The list size of the solutions that can be chosen is specified by another parameter: `EliteRandLimit`. We introduce this limit such that only the best solutions are relinked but at the same time the pool is big enough so we can do a regeneration.

The operation of pool regeneration is done every `EliteRegeneration` iterations. During regeneration (Algorithm 2), several path-relinking operations are made with some of the best solutions to all the others in the same elite pool trying to reach new best results. This operation also introduces variability into the pool; otherwise the pool can converge to a set of solutions that do not help to improve new partitions and do not allow new solutions to enter on it.

3.2. Path-relinking

With the basic GRASP, based on a Differential Greedy heuristic construction, the local search of Fiduccia-Mattheyses, and the introduction of an elite pool, we have the required frame-

Algorithm 2: Elite regeneration

Input: Elite pool $E = \{E_0, E_1, \dots\}$ sorted by non-decreasing cut size; parameters `EliteMinDiff`
Output: A new elite pool E'

1 $E' \leftarrow \{E_0\}$;
2 **foreach** $P \in \{E_i \in E \mid i <$ `EliteRandLimit`$\}$ **do**
3 **foreach** $Q \in E$ **do**
4 $R \leftarrow$`path_relink_fm(`P, Q`);`
5 $R \leftarrow$`iterated_local_search(`R`);`
6 **if** R is the best solution **or** $\Delta(R, S) \geq$ `EliteMinDiff`$, \forall S \in E'$ **then** $E' \leftarrow E' \cup \{R\}$;
7 **if** $|E'| >$ `EliteMaxSize` **then** drop the most similar solution with R from E';
8 **return** E';

work for applying the path-relinking. We propose two path-relinking operators with specific purposes. The first one, `path_relink_dg` , is designed to be used in the constructive phase of GRASP. The second one, `path_relink_fm` , is used to discover new local minima based on two previously found solutions. For both of them to work, we must first lock vertices on equivalent parts.

The comparison of partitions uses the mapping σ_* previously discussed on Section 3.1. As explained above, this function maps the parts of two partitions such that the total number of vertices common to the mapped parts is maximized. This mapping is used to lock these common vertices so they are forbidden to change parts during PR. The remaining vertices are moved according to each operator.

The operator `path_relink_dg` moves unlocked vertices based on the Differential Greedy (DG) algorithm explained before. The DG requires one seed vertex on each part (chosen randomly according to the basic algorithm). This PR operator starts the DG algorithm with the locked vertices as the part's seeds and continues growing the solution as defined by DG. If a part has no locked vertices, we choose an unlocked vertex randomly to be its seed. Since DG is unable to generate imbalanced solutions, we always grow the smaller parts first until all have the same size.

The other operator, `path_relink_fm` , starts with the first partition and locks the vertices common to parts of the second. The unlocked vertices are only allowed to be moved to their equivalent part. For example, if a vertex $u \in A_1$ and $u \in B_3$, but $\sigma_*(2) = 3$, the only allowed move for u is from part A_1 to A_2 because A_2 is equivalent to B_3, as defined by σ_*. With this constraint, after all unlocked vertices are moved to their destinations, the first partition will have been transformed to the second. The order of movements are guided by Fiduccia-Mattheyses heuristic explained before and all allowed movements are performed. After a vertex is moved it gets locked. If a solution better than the two initial partitions is found, we update the best solution for this GRASP iteration.

3.3. Full partitioner

We finally present our complete algorithm for the graph partitioning problem. Algorithm 3 is the proposed GRASP with path-relinking and a pool of elite solutions. The first phase of a GRASP iteration corresponds to lines 4 to 9. It is an iterative constructive and refinement of a solution for the current iteration. Instead of generating only a single solution with DG, this loop builds one every iteration (line 5) and merges the information of it and the best of the phase (line 7) to improve results of the DG construction. The local search on line 6 is intended to refine the solution from DG. PR turned out to be more useful this way, probably because DG results are still too random to be compared with the best solution of the phase. Line 8 is the iterative refinement detailed before. Lines 10 to 15 deal with the PR phase. The iterative loop continues only when P' improved, i.e., a new local minimum is found between P and Q. Lines 16 to 18 manage the elite pool as detailed in Section 3.1.

Algorithm 3: GRASP for k-way graph partitioning problem

Input: Graph $G = (V, E)$; imbalance limit ϵ; partition size k; parameters from Section 3.1
Output: A k-way partition of V

$E \leftarrow \emptyset$;
while no stop criterion is satisfied **do**
  $P \leftarrow$ initial solution by `diff_greedy`;
  repeat
    $Q \leftarrow$ initial solution by `diff_greedy`;
    $Q \leftarrow$ `local_search(`Q`)`;
    $P \leftarrow$ `path_relink_dg(`P, Q`)`;
    $P \leftarrow$ `iterative_improvement_local_search(`P`)`;
  until P not improved;
  if elite set E has some solution **then**
    repeat
      $Q \leftarrow$ randomly choose a solution from the first `EliteRandLimit` in E;
      $P' \leftarrow$ `path_relink_fm(`P, Q`)`;
      if P' is better than P and Q **then** $P \leftarrow$ `iterative_improvement_local_search(`P'`)`;
    until P did not improve;
  if R is the best solution **or** $\Delta(R, S) \geq$ `EliteMinDiff`$, \forall S \in E'$ **then** $E' \leftarrow E' \cup \{R\}$;
  if $|E'| >$ `EliteMaxSize` **then** drop the most similar solution with R from E';
  if current iteration multiple of `EliteRegeneration` **then** $E \leftarrow$ `regenerate(`E`)`;
return best solution in E;

Table 1. Graphs from the graph partitioning archive (Walshaw, 2000).

Instance	n	m	Instance	n	m	Instance	n	m	Instance	n	m
add20	2395	7462	fe_4elt2	11143	32818	bcsstk31	35588	572914	fe_rotor	99617	662431
data	2851	15093	vibrobox	12328	165250	fe_pwt	36519	144794	598a	110971	741934
3elt	4720	13722	bcsstk29	13992	302748	bcsstk32	44609	985046	fe_ocean	143437	409593
uk	4824	6837	4elt	15606	45878	fe_body	45087	163734	144	144649	1074393
add32	4960	9462	fe_sphere	16386	49152	t60k	60005	89440	wave	156317	1059331
bcsstk33	8738	291583	cti	16840	48232	wing	62032	121544	m14b	214765	1679018
whitaker3	9800	28989	memplus	17758	54196	brack2	62631	366559	auto	448695	3314611
crack	10240	30380	cs4	22499	43858	finan512	74752	261120			
wing_nodal	10937	75488	bcsstk30	28924	1007284	fe_tooth	78136	452591			

4. Experimental results

We conducted a series of experiments in order to evaluate the quality of the solutions generated by our GRASP. We used graphs from a well-known graph partitioning archive (Walshaw, 2000). Along with the instances, it provides the best known values for each of them. Characteristics of the graphs are shown in Table 1. For result analysis, we divided the instances in three classes: small instances, from `add20` to `cs4`; medium instances, from `bcsstk30` to `finan512`; and large instances, from `fe_tooth` to `auto`.

Our experiments were run on a PC with Intel Core i7 930 processor running at 2.8 GHz with 12 GiB DDR3 of RAM. We imposed a time limit of two hours for each instance. These times are acceptable since state-of-the-art approaches impose limits from one to five hours (Osipov and Sanders, 2010; Sanders and Schulz, 2010; Benlic and Hao, 2011). In order to find good parameter settings, we conducted some preliminary tests. The resulting parameters and respective values are: $\alpha = 100$, $\beta = \log_2 n$, `maxT` $= k\sqrt{m}$, local search movements limit $= 10n$, `EliteMinDiff` $= 5\%$, `EliteMaxSize` $= 5$, `EliteRandLimit` $= 5$, `EliteRegeneration` $= 32$. Tests were repeated three times for each combination of $k \in \{4, 8\}$ and $\epsilon \in \{1.03, 1.05\}$. The best and average cut size found for each instance in all repetitions is shown in Tables 2 and 3, for the cases $\epsilon = 1.03$ and $\epsilon = 1.05$ respectively.

An overall view of the results shows us that the GRASP is more effective for the cases where $k = 4$ in both tested imbalances. Compared to best known values, our algorithm achieves an average relative deviation of 3.9%. The deviation is 1.6% for the small instances, 9.7% for the medium instances and 2.1% for the large instances. We were able to improve the results of eleven

cases and achieved the best known value in 40 of them.

The large relative deviations are mostly due to some graphs that the procedure has difficulties to deal with, such as `uk`, `memplus`, `cs4`, `fe_body`, `t60k`, `wing` and `auto`. The problem with the last one is due to the fact that only few iterations were able to be performed in the given time limit. This also supports our first finding, based on analysis of the partitionings, that the path-relinking is responsible for most of the improvements on cut size. The large instances that had a low number of performed iterations were not able to take full advantage of the newly proposed operators. For the instances `fe_body`, `t60k` and `wing`, our constructive algorithm turned out to be a lot less effective than expected when compared to the remaining instances in the dataset. With these results we observe that good initial partitions play an important role on the whole procedure and we shall search for better methods.

The improvements show that our approach is promising because it is competitive even with the recent extra attention that the graph partitioning gained as one of the target problems of the 10th DIMACS Implementation Challenge[1]. Almost all best known values at this time have been obtained by the algorithm KaFFPaE proposed by Sanders and Schulz (2011) during this challenge. KaFFPaE is a distributed, evolutionary and multilevel partitioner and the best values were obtained, for each test, within a time limit of two hours on 16 processor units. In particular, the fact that we were able to improve some of the smallest instances indicates, in our opinion, that our approach can make a qualitative contribution compared to existing methods, since these are the easiest instances and these values have not been found in tests of more than 35 proposed heuristics.

Another finding with our approach is related to the performance of the tabu based local search versus the simple version of Fiduccia-Mattheyses. Inside the GRASP, the tabu version reduces by 10% the cut sizes on average, even requiring some adjustments.

5. Conclusions

In this paper we proposed a new approach to the k-way graph partitioning problem based on GRASP, path-relinking, extensions of heuristics proposed for the graph bipartitioning problem such as the constructive algorithm Differential Greedy and the refinement algorithm Fiduccia-Mattheyses along with some state-of-the-art enhancements for the latter one. The proposed GRASP was very effective in some instances, achieving new best values. One interesting finding is related to the path-relinking, which has proven to be very effective. Nevertheless, the procedure generates cuts that are in average still 3.9% above the best known values, which is mainly due to a few hard instances. We plan to investigate the causes and attempt to improve partition quality by better calibrating the involved parameters and finding a better and simpler tenure function for the tabu based Fiduccia-Mattheyses.

References

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

Battiti, R. and Bertossi, A. A. (1999). Greedy, prohibition, and reactive heuristics for graph partitioning. *IEEE Transactions on Computers*, 48:361–385.

Benlic, U. and Hao, J. (2011). A multilevel memetic approach for improving graph k-partitions. *IEEE Transactions on Evolutionary Computation*. to appear.

Boulle, M. (2004). Compact mathematical formulation for graph partitioning. *Optimization and Engineering*, 5:315–333.

[1] http://www.cc.gatech.edu/dimacs10

Bui, T. N. and Jones, C. (1992). Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42:153–159.

Chardaire, P., Barake, M., and McKeown, G. P. (2007). A probe-based heuristic for graph partitioning. *IEEE Transactions on Computers*, 56:1707–1720.

Feo, T. A. and Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71.

Fiduccia, C. M. and Mattheyses, R. M. (1982). A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181, Piscataway, NJ, USA. IEEE Press.

Galinier, P., Boujbel, Z., and Fernandes, M. C. (2011). An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research*, 191:1–22.

Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. (1989). Optimization by simulated annealing: an experimental evaluation. part i, graph partitioning. *Oper. Res.*, 37:865–892.

Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392.

Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307.

Kim, Y. and Moon, B. (2004). Lock-gain based graph partitioning. *Journal of Heuristics*, 10:37–57.

Laguna, M., Feo, T. A., and Elrod, H. C. (1994). A greedy randomized adaptive search procedure for the two-partition problem. *Operations Research*, 42(4):677–687.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38.

Osipov, V. and Sanders, P. (2010). n-level graph partitioning. In *Proceedings of the 18th annual European conference on Algorithms: Part I*, ESA'10, pages 278–289, Berlin, Heidelberg. Springer-Verlag.

Pellegrini, F. (2007). A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries. In Kermarrec, A., Bougé, L., and Priol, T., editors, *Euro-Par 2007 Parallel Processing*, volume 4641 of *Lecture Notes in Computer Science*, pages 195–204. Springer Berlin / Heidelberg.

Resende, M. G. C. and Ribeiro, C. C. (2003). GRASP and path-relinking: Recent advances and applications. In Ibaraki, T. and Yoshitomi, Y., editors, *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*.

Rolland, E., Pirkul, H., and Glover, F. (1996). Tabu search for graph partitioning. *Annals of Operations Research*, 63:209–232.

Sanders, P. and Schulz, C. (2010). Engineering multilevel graph partitioning algorithms. *CoRR*, abs/1012.0006.

Sanders, P. and Schulz, C. (2011). Distributed evolutionary graph partitioning. *CoRR*, abs/1110.0477.

Schloegel, K., Karypis, G., and Kumar, V. (2003). Graph partitioning for high-performance scientific simulations. In Dongarra, J., Foster, I., Fox, G., Gropp, W., Kennedy, K., Torczon, L., and White, A., editors, *Sourcebook of parallel computing*, pages 491–541. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Walshaw, C. (2000). The graph partitioning archive. http://staffweb.cms.gre.ac.uk/~c.walshaw/partition.

Table 2. Best and average results found by our GRASP among all repetitions compared with the best known values (BKV) when $\epsilon = 1.03$. Bold values mark (new) best known values. Relative deviation is presented as a percentage of our cuts relative to BKV. We also provide some statistics such as the number of results that were improved, matched BKV or had worst cuts, the average and standard deviation of relative deviation and the average of standard deviation counting only worst results.

	k = 4					k = 8				
				Relative Deviation					Relative Deviation	
Instance	BKV	Best	Avg	of best	of avg	BKV	Best	Avg	of best	of avg
add20	1158	**1135**	1142.7	-2.0	-1.3	**1689**	1693	1701.7	0.2	0.7
data	**369**	**369**	369.0	0.0	0.0	**638**	**638**	638.3	0.0	0.1
3elt	**198**	**198**	198.0	0.0	0.0	**334**	335	335.7	0.3	0.5
uk	**39**	42	42.3	7.7	8.5	**78**	85	90.0	9.0	15.4
add32	**33**	**33**	33.0	0.0	0.0	**66**	**66**	66.3	0.0	0.5
bcsstk33	20854	**20762**	20763.3	-0.4	-0.4	34078	**34065**	34067.3	0.0	0.0
whitaker3	**378**	**378**	378.0	0.0	0.0	**650**	653	653.3	0.5	0.5
crack	**360**	**360**	360.0	0.0	0.0	**671**	673	673.7	0.3	0.4
wing_nodal	**3538**	**3538**	3539.3	0.0	0.0	**5361**	5366	5367.0	0.1	0.1
fe_4elt2	**342**	**342**	342.7	0.0	0.2	**595**	597	598.7	0.3	0.6
vibrobox	**18736**	18742	18742.3	0.0	0.0	**24170**	24227	24231.3	0.2	0.3
bcsstk29	**7971**	7993	8026.7	0.3	0.7	**13717**	13791	13794.7	0.5	0.6
4elt	**319**	**319**	319.0	0.0	0.0	**522**	523	523.0	0.2	0.2
fe_sphere	**764**	**764**	764.0	0.0	0.0	**1152**	**1152**	1152.0	0.0	0.0
cti	**916**	**916**	916.0	0.0	0.0	**1714**	**1714**	1716.0	0.0	0.1
memplus	**9362**	10473	10577.3	11.9	13.0	**11624**	12928	13113.3	11.2	12.8
cs4	**917**	988	1005.3	7.7	9.6	**1424**	1546	1558.7	8.6	9.5
bcsstk30	16399	**16371**	16392.3	-0.2	0.0	**34137**	34179	34232.0	0.1	0.3
bcsstk31	**7150**	7171	7191.3	0.3	0.6	**12985**	13201	13234.3	1.7	1.9
fe_pwt	**700**	**700**	700.0	0.0	0.0	**1410**	1415	1415.0	0.4	0.4
bcsstk32	**8725**	8880	9449.0	1.8	8.3	**19956**	20827	21202.0	4.4	6.2
fe_body	**598**	661	674.7	10.5	12.8	**1016**	1147	1226.7	12.9	20.7
t60k	**203**	252	268.3	24.1	32.2	**449**	856	930.7	90.6	107.3
wing	**1593**	1803	1822.7	13.2	14.4	**2451**	3088	3462.7	26.0	41.3
brack2	**2834**	**2834**	2834.0	0.0	0.0	**6800**	6835	6860.3	0.5	0.9
finan512	**324**	**324**	324.0	0.0	0.0	**648**	**648**	648.0	0.0	0.0
fe_tooth	**6764**	6777	6784.3	0.2	0.3	**11274**	11464	11481.3	1.7	1.8
fe_rotor	7118	**7103**	7138.0	-0.2	0.3	**12445**	12886	12928.7	3.5	3.9
598a	**7816**	7858	7869.0	0.5	0.7	**15613**	15875	15900.3	1.7	1.8
fe_ocean	**1693**	**1693**	1693.0	0.0	0.0	**3920**	3949	3962.3	0.7	1.1
144	**15078**	15151	15196.7	0.5	0.8	**25092**	26515	26687.3	5.7	6.4
wave	**16665**	16728	16736.0	0.4	0.4	**28495**	28966	29007.0	1.7	1.8
m14b	**12948**	13281	13332.3	2.6	3.0	**25390**	26564	26613.0	4.6	4.8
auto	**25789**	26621	26904.3	3.2	4.3	**44785**	49207	49909.3	9.9	11.4
Improved:				4	3				1	1
Matched:				14	12				5	2
Worst:				16	19				28	31
Average (%):				2.42	3.19				5.81	7.48
Std (%):				5.38	6.74				15.94	19.46
Average of worsts (%):				5.31	5.80				7.05	8.20

Table 3. Best and average results found by our GRASP among all repetitions compared with the best known values (BKV) when $\epsilon = 1.05$.

	k = 4					k = 8				
				Relative Deviation					Relative Deviation	
Instance	BKV	Best	Avg	of best	of avg	BKV	Best	Avg	of best	of avg
add20	1149	**1128**	1128.0	-1.8	-1.8	**1675**	1682	1684.3	0.4	0.6
data	**363**	**363**	363.0	0.0	0.0	**628**	**628**	628.0	0.0	0.0
3elt	**197**	**197**	197.0	0.0	0.0	**329**	**329**	329.7	0.0	0.2
uk	**39**	40	41.7	2.6	6.8	**75**	84	88.3	12.0	17.8
add32	**33**	**33**	33.0	0.0	0.0	**63**	**63**	63.3	0.0	0.5
bcsstk33	**20167**	**20167**	20167.0	0.0	0.0	33919	**33916**	33916.0	0.0	0.0
whitaker3	377	**376**	376.3	-0.3	-0.2	**644**	648	649.0	0.6	0.8
crack	**360**	**360**	360.0	0.0	0.0	**666**	**666**	667.0	0.0	0.2
wing_nodal	**3521**	3522	3522.7	0.0	0.0	**5341**	5345	5346.3	0.1	0.1
fe_4elt2	**335**	337	337.0	0.6	0.6	**578**	583	583.7	0.9	1.0
vibrobox	**18690**	**18690**	18693.3	0.0	0.0	**23924**	23944	23957.3	0.1	0.1
bcsstk29	**7925**	7986	7992.0	0.8	0.8	**13540**	13556	13615.0	0.1	0.6
4elt	**315**	**315**	315.3	0.0	0.1	**515**	516	516.0	0.2	0.2
fe_sphere	**762**	**762**	762.0	0.0	0.0	**1152**	**1152**	1152.0	0.0	0.0
cti	**889**	**889**	889.0	0.0	0.0	**1684**	**1684**	1684.0	0.0	0.0
memplus	**9292**	10387	10468.0	11.8	12.7	**11543**	12796	12882.3	10.9	11.6
cs4	**909**	977	995.7	7.5	9.5	**1420**	1521	1543.7	7.1	8.7
bcsstk30	16186	**16169**	16180.0	-0.1	0.0	**34071**	34121	34149.0	0.1	0.2
bcsstk31	7086	**7079**	7096.7	-0.1	0.2	**12853**	13129	13161.3	2.1	2.4
fe_pwt	**700**	**700**	700.0	0.0	0.0	**1405**	**1405**	1405.3	0.0	0.0
bcsstk32	**8441**	8468	9071.7	0.3	7.5	**19411**	20290	20386.7	4.5	5.0
fe_body	**588**	675	707.0	14.8	20.2	**1013**	1129	1200.7	11.5	18.5
t60k	**195**	231	238.0	18.5	22.1	**443**	786	854.3	77.4	92.9
wing	**1590**	1716	1767.0	7.9	11.1	**2440**	3058	3182.0	25.3	30.4
brack2	**2731**	**2731**	2731.7	0.0	0.0	**6592**	6609	6615.3	0.3	0.4
finan512	**324**	**324**	324.0	0.0	0.0	**648**	**648**	648.0	0.0	0.0
fe_tooth	**6688**	6704	6705.7	0.2	0.3	**11154**	11297	11338.7	1.3	1.7
fe_rotor	6899	**6804**	6824.0	-1.4	-1.1	**12309**	12775	12808.3	3.8	4.1
598a	**7728**	7762	7764.7	0.4	0.5	**15414**	15676	15712.0	1.7	1.9
fe_ocean	**1686**	**1686**	1686.3	0.0	0.0	**3893**	3961	3963.7	1.7	1.8
144	**14982**	15102	15122.7	0.8	0.9	**24767**	25899	26066.7	4.6	5.2
wave	**16533**	16633	16636.7	0.6	0.6	**28489**	28733	28835.3	0.9	1.2
m14b	**12945**	13062	13187.7	0.9	1.9	**25143**	26236	26527.7	4.3	5.5
auto	**25271**	25867	26031.3	2.4	3.0	**44206**	47488	49172.7	7.4	11.2
Improved:				5	4				1	1
Matched:				13	9				8	4
Worst:				16	21				25	29
Average (%):				1.95	2.82				5.27	6.61
Std (%):				4.63	5.83				13.79	16.68
Average of worsts (%):				4.38	4.71				7.17	7.75