Computers & Operations Research 38 (2011) 1066–1075

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

An effective multilevel tabu search approach for balanced graph partitioning

Una Benlic, Jin-Kao Hao *

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France

ARTICLE INFO

Available online 17 October 2010

Keywords:
Balanced partitioning
Multilevel approach
Iterated tabu search

ABSTRACT

Graph partitioning is one of the fundamental NP-complete problems which is widely applied in many domains, such as VLSI design, image segmentation, data mining, etc. Given a graph $G=(V,E)$, the balanced k-partitioning problem consists in partitioning the vertex set V into k disjoint subsets of about the same size, such that the number of cutting edges is minimized. In this paper, we present a multilevel algorithm for balanced partition, which integrates a powerful refinement procedure based on tabu search with periodic perturbations. Experimental evaluations on a wide collection of benchmark graphs show that the proposed approach not only competes very favorably with the two well-known partitioning packages METIS and CHACO, but also improves more than two thirds of the best balanced partitions ever reported in the literature.

1. Introduction

Given an undirected graph $G=(V, E)$, where V and E denote sets of vertices and edges respectively, the balanced k-partitioning problem consists in partitioning the vertex set V into k ($k \geq 2$) disjoint subsets of approximately equal size, such that the number of cutting edges (i.e. edges whose endpoints belong to different subsets) is minimized. The particular partitioning case when k is set to two is often called bisection. The class of partitioning problems is notable for its applicability to a wide range of important applications such as VLSI design [1,26], data mining [30], and image segmentation [24].

The general graph k-partitioning problem is NP-complete, and it remains NP-complete even for the case of bisection ($k=2$) [11]. Therefore, approximate methods constitute a natural and useful solution approach to address this problem. In recent years, many efforts have been made in devising a number of efficient and powerful heuristics.

One of the most popular graph bisection heuristics is the famous Kernighan–Lin (KL) algorithm [19] that iteratively refines an existing bisection. The idea is to find two subsets of vertices of about the same cardinality, one for each bisection part, and to improve the quality of the bisection by swapping two vertices of the two subsets of the bisection. Fiduccia and Mattheyses [10] presented an improvement of the KL algorithm by introducing the concept of cell gain and an efficient data structure to calculate cell gains. Additional KL-like heuristics are reported, for instance, in [17] for bisection and in [8,18,28] for k-partitioning. Other well-known algorithms are based on popular metaheuristics such as tabu search [7,23,3], simulated annealing [15], genetic and evolutionary algorithms [27,5,20,25,26].

To handle large and very large graphs, the so-called multilevel paradigm proves to be quite useful [2,8,18,28,21]. The basic idea of a multilevel approach is to successively create a sequence of progressively smaller graphs by grouping vertices into clusters (coarsening phase). This phase is repeated until the size of the smallest graph falls below a certain coarsening threshold. Afterward, a partition of the coarsest graph is generated (initial partitioning phase), and then successively projected back onto each of the intermediate graphs in reverse order (uncoarsening phase). The partition at each level is improved by a dedicated refinement algorithm before moving to the upper level. Even though almost all multilevel algorithms employ the same or similar heuristics in the coarsening phase, they differ greatly in the way the initial partition is generated, and how the successive partitions are refined throughout each uncoarsening phase.

As illustrated in [29], the multilevel paradigm is a useful approach to solving combinatorial optimization problems. Basically, the approach allows one to approximate the initial problem by solving successively smaller (and easier) problems. Moreover, the coarsening helps filter the solution space by placing restrictions on which solutions the refinement algorithm can visit.

This work is based on the multilevel paradigm. The main contribution reported in this paper is a perturbation-based iterated tabu search procedure which is specially devised for balanced k-partitioning and used for partition refinement of each coarsened graph. Integrated within the multilevel approach, our multilevel iterated tabu search (MITS) algorithm competes very favorably with the two well-known partitioning packages METIS [16] and

* Corresponding author.
E-mail addresses: benlic@info.univ-angers.fr (U. Benlic), hao@info.univ-angers.fr (J.-K. Hao).

0305-0548/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2010.10.007

CHACO [8]. Indeed, extensive experiments performed on the whole set of benchmark graphs from the University of Greenwich graph partitioning archive reveal that MITS is able to find better partitions (with fewer cutting edges) than METIS and CHACO for a large portion of these benchmark graphs. Furthermore, when the running time is prolonged up to 1 h, MITS even improves some two thirds of the current best balanced partitions ever reported in the literature while many of these best partitions were obtained by previous algorithms using a significantly longer running time (up to one week).

The rest of the paper is organized as follows. In the next section, we provide the formal graph partitioning description, as well as some notations used throughout this paper. In Section 3, we describe the multilevel paradigm, and detail the phases of the proposed multilevel algorithm. In Section 4, we present the perturbation-based tabu search algorithm, which is the key partition refinement mechanism in our multilevel approach. In Section 5, we present computation results and comparisons on the benchmark graphs. Finally, in Section 6 we investigate two important features of the proposed MITS algorithm.

2. Problem description and notations

Given an undirected graph $G=(V, E)$, V and E being the set of vertices and edges respectively, and a fixed number k, a k-partitioning of G can be defined as a mapping (partition function) $\pi : V \rightarrow \{1,2,\ldots,k\}$ that distributes the vertices of V among k disjoint subsets $S_1 \cup S_2 \cup \ldots \cup S_k = V$ of roughly equal size.

The partition function π induces a new graph $G_\pi = G_\pi(S,E_c)$, where $S= \{S_1, S_2,\ldots,S_k\}$ and an edge $\{S_x,S_y\} \in E_c$ exists if there are two adjacent vertices $u,v \in V$ such that $u \in S_x$ and $v \in S_y$. The set E_c corresponds to the set of cutting edges of G induced by the partition. Vertices u and v of a cutting edge $\{u,v\} \in E_c$ are called border vertices of the partition.

Throughout this paper, the initial input graph G is supposed to have a unit cost weight for both vertices and edges. However, as explained in Section 3.2, the multilevel approach generates intermediate (coarsened) graphs where vertices and edges may have different weights (see Section 3.2). As a general rule, these weights increase during the coarsening phase, and decrease during the uncoarsening phase. Let $|v|$ denote the weight of a vertex v. Then, the weight $W(S_i)$ of a subset S_i is equal to the sum of weights of the vertices in S_i, $W(S_i) = \sum_{v \in S_i} |v|$.

The optimization objective f of our balanced partition problem is to find a balanced partition with the minimum sum of weights of cutting edges E_c.

3. Multilevel tabu search graph partitioning

3.1. General procedure

Our multilevel tabu search approach follows the general multilevel paradigm [8,4,2].

Let $G_0=(V_0,E_0)$ be the initial graph, and let k denote the number of partition subsets. The multilevel paradigm can be summarized by the following steps.

(1) Coarsening phase: The initial graph G_0 is transformed into a sequence of smaller graphs G_1, $G_2,\ldots,G_m$ such that $|V_0| > |V_1| > |V_2| > \cdots > |V_m|$. This phase stops when $|V_m|$ reaches a fixed threshold (coarsening threshold).
(2) Partitioning phase: A k-partition P_m of the coarsest graph $G_m=(V_m,E_m)$ is generated.
(3) Uncoarsening phase: Partition P_m is progressively projected back to each intermediate G_i ($i=m-1, m-2,\ldots,0$). Before each projection, the partition is first refined (improved) by a refinement algorithm.

This process leads thus to a sequence of partitions $P_m,P_{m-1},P_{m-2},\ldots,P_0$. The last one, i.e. P_0 is returned as the final partition of the original graph G_0.

Our multilevel iterated tabu search (MITS) algorithm follows this general scheme and is described in Algorithm 1.

Algorithm 1. Multilevel iterated tabu search partition approach

Require An undirected graph $G_0=(V_0, E_0)$; An integer k
Ensure A partition P_0 of G_0
  Begin
  $i := 0$
  while $|V_i| > coarsening\_threshold$ **do**
    $G_{i+1}=$**Coarsen**(G_i) {Section 3.2}
    $i := i + 1$
  end while
  $P_i=$**Initial_Partition**(G_i, k) {Section 3.3}
  $P_i=$**Tabu_Refinement** (P_i) {Section 4}
  while $i > 0$ **do**
    $i := i-1$
    $P_i=$**Project** (P_{i+1}, G_i) {Section 3.3.1}
    $P_i=$**Tabu_Refinement** (P_i)
  end while
  Return (P_0)
  End

3.2. Coarsening phase

Creating a coarser graph $G_{i+1} = (V_{i+1},E_{i+1})$ from $G=(V_i,E_i)$ consists in finding an independent subset of graph edges $\Gamma \subset E_i$, and then collapsing the two vertices of each edge of Γ. A set of edges is independent if no two edges in the set are incident to the same vertex, which implies that exactly two vertices are collapsed each time. If a vertex is not incident to any edge of subset Γ, it is simply copied over to G_{i+1}.

When vertices $v_1,v_2 \in V_i$ are collapsed to form a vertex $v_c \in V_{i+1}$, the weight of the resulting vertex v_c is set equal to the sum of weights of vertices v_1 and v_2. The edge that is incident to v_c becomes the union of all the edges incident to v_1 and v_2, minus the edge $\{v_1,v_2\} \in E_i$. Therefore, during the coarsening process, both the vertex and edge weight increases. At any level of the coarsening phase, the weight of a vertex (an edge) corresponds to the number of aggregated vertices (edges) of the initial graph G_0.

One key issue here is the selection of the independent subset of graph edges Γ to be collapsed at each step of the coarsening phase. This can be achieved by finding a maximum matching of the graph. Recall that a matching of a graph is a subset of edges such that no two edges share the same vertex [22]. There exist polynomial time algorithms for tackling this problem, with running time of at least $O(|V|^{2.5})$. Unfortunately, this is too slow to be applicable to the partitioning problem. That is why we compute maximal matching using an edge contraction heuristic called heavy-edge matching (HEM), which has $O(|E|)$ time complexity [17]. This method considers vertices in random order, matching each unmatched vertex v with its unmatched neighbor u, if any, such that the weight of edge $\{u,v\}$ is maximum among all the edges incident to v.

3.3. Initial partition and its refinement

Once we have obtained a sequence of smaller graphs in the coarsening phase, the next step is to generate an initial partition of the smallest graph $G_m=(V_m,E_m)$. The initial partitioning phase of

our algorithm consists of first assigning randomly the vertices of V_m to k subsets $S_i \in \{S_1, S_2, \ldots, S_k\}$, such that the number of clustered vertices in each subset is evenly balanced, i.e. the number of represented vertices n of G_0 in each subset $\{S_1, S_2, \ldots, S_k\}$ is $n \leq \lceil |V_m|/k \rceil$.

Afterward, we refine this initial partition by means of a dedicated perturbation-based tabu search algorithm which is described in Section 4. This refinement step is essential for our approach to improve progressively the quality of partitions. It should be noted that for certain graphs, it is impossible to obtain a perfectly balanced initial partition, since weights of vertices in the coarsest graph can be greatly inhomogeneous. The imbalance ε of a partition $\{S_1, S_2, \ldots, S_k\}$ is defined as $\varepsilon = \max_{i \in \{1..k\}} W(S_i)/W_{opt}$ where $W_{opt} = \lceil |V|/k \rceil$, $\lceil x \rceil$ being the ceiling function returning the first integer $\geq x$.

This imbalance is gradually reduced throughout each uncoarsening step, and (usually) completely eliminated by the end of the algorithm execution.

3.3.1. Uncoarsening phase

The uncoarsening phase carries out the inverse of the coarsening phase. The idea is to go from level to level, uncoarsening the clustered vertices in the opposite way they were grouped during the coarsening phase. The partition projection from a graph $G_i=(V_i, E_i)$ onto a partition of the parent graph $G_{i-1}=(V_{i-1}, E_{i-1})$ is a trivial process. If a vertex $v \in V_i$ is in subset S_m of the partition of G_i, then the matched pair of vertices $v_1, v_2 \in V_{i-1}$ which represents vertex $v \in V_i$ will be in subset S_m of the projected partition of G_{i-1}.

Before projecting a partition to the next level, we apply the perturbation-based tabu search refinement algorithm (Section 4) to improve the partition quality. As the uncoarsening/refining process proceeds, the partition quality of graph G_{i-1} is usually better than that of G_i because there is a greater degree of freedom for refinement. This is one of the most attractive characteristics of a multilevel algorithm.

4. Partition refinement by perturbation-based iterated tabu search

The refinement algorithm, used at each level for partition improvement, is based on a perturbation-based iterated tabu search procedure. Basically, the tabu search algorithm [13] ensures the intensification of the approach, while the perturbation mechanism brings controlled diversification into the search. Experiments show that the combination of these two mechanisms constitutes a very effective refinement method for obtaining high quality partitions. Algorithm 2 describes this refinement procedure, whose components are detailed in the following sections.

Algorithm 2. Perturbation-based tabu search for partition refinement

Require: Graph $G_i=(V_i, E_i)$ at level i; Initial partition $P_i=\{S_1, S_2, \ldots, S_k\}$.
Ensure: The best partition P_i^* of graph G_i

1: Initiate *iteration_counter*, *tabu_list*, *tabu_tenure*, *move_frequency_list*
2: $P_i^* \leftarrow P_i$ {P_i^* records the best partition found so far}
3: **for** each vertex v and each subset S_m of P_i such that $v \notin S_m$ **do**
4: Compute move gain $g(v, m)$
5: Insert v into the bucket structure corresponding to S_m {Section 4.1.3}
6: **end for**
7: **repeat**
8: {Apply neighborhoods N_1 and N_2 in token-ring way (Section 4.1.2)}
9: **if** turn to employ N_1 **then**
10: {Apply move operator *Single_Move*} {Sections 4.1.2 and 4.1.4}
11: Select target subset S_m and vertex v_m for migration
12: $P_i \coloneqq P_i \oplus \mathbf{Move}(v_m, S_m)$ {Move v_m to S_m to generate new partition}
13: Update tabu list and bucket sorting structure
14: **else if** turn to employ N_2 **then**
15: {Apply move operator *Double_Move*} {Sections 4.1.2 and 4.1.4}
16: Select target subset S_m and vertex v_m for migration
17: $P_i \coloneqq P_i \oplus \mathbf{Move}(v_m, S_m)$
18: Select target subset S_n and vertex v_n for migration
19: $P_i \coloneqq P_i \oplus \mathbf{Move}(v_n, S_n)$
20: Update tabu list and bucket sorting structure
21: **end if**
22: {Update the best solution P_i^* found so far}
23: **if** ($f(P_i) \leq f(P_i^*)$) and P_i is at least as well balanced as P_i^* **then**
24: $P_i^* \leftarrow P_i$
25: **end if**
26: **if** (P_i^* not improved after γ iterations) **then**
27: $P_i \coloneqq Perturb(P_i, p_{str})$ {Section 4.2}
28: **end if**
29: **until** stop condition not met

4.1. Tabu search

In this subsection, we focus on the tabu search engine of our partition refinement algorithm. It explores the search space by repeatedly replacing the current solution with a best non-recently visited neighboring solution, even if the later deteriorates the solution quality. Since tabu search relies on the belief that intelligent search should be based on learning, its underlying element is the usage of flexible memory that exploits and keeps track of the search history. In order to avoid possible cycling and go beyond local optimum, tabu search introduces the notion of tabu list to forbid the recently visited solutions.

4.1.1. Objective and partition balance

The optimization objective f of our k-partitioning problem is to minimize the sum of weights of cutting edges of a balanced partition. Note that, in the given problem formulation, partition balance is imposed as a constraint. However, as we have already mentioned earlier, it is often impossible to establish perfect balance in coarsened graphs, since vertex weights can be extremely inhomogeneous. It is during the partition refinement of the levels which are closer to the original graph that the balance condition is (usually) completely satisfied.

More precisely, our refinement algorithm uses two move operators that not only help optimize the objective (number of cutting edges), but also take care of partition imbalance (see Section 4.1.2). Typically, these move operators transfer vertices to subsets with smaller weight. Applying these directional moves generally allows the search to progressively establish perfect partition balance towards the end of the backward projection process.

During the refinement process, the current solution P_i (k-partition) becomes the new best solution P^* only if P_i is at least as good as P^* in terms of the optimization objective and partition balance (see lines 23–25 of Algorithm 1).

4.1.2. Neighborhoods and their exploration

Our tabu search algorithm employs two neighborhood relations N_1 and N_2 that are based on two different *move operators*. These

operators transfer respectively one and two vertices between subsets of a partition.

To define these move operators, we first introduce the notion of *move gain* which indicates how much a partition is improved according to the optimization objective if a vertex is moved to another subset. Given a vertex v from subset S_m, the gain $g(v,n)$ can be computed for every other subset, S_n, $m \neq n$. As we show in Section 4.1.3, the vertex with the best (highest) gain can be easily determined using a special bucket data structure.

Let $P=\{S_1, S_2,\ldots,S_k\}$ be a k-partition, and let S_{max} be the subset with the maximum weight, $S_{max} = \max_{i \in \{1..k\}}\{W(S_i)\}$. The two move operators are given below.

Single_Move (N_1): Move one highest gain vertex v_m. Choose randomly a subset S_m, $m \neq \max$. Then, select a highest gain vertex v_m (see Section 4.1.4) whose current subset is S_c, such that $S_c \in \{S_i \in P | W(S_i) > W(S_m)\}$. Finally, move the selected vertex v_m to the target subset S_m.

Double_Move (N_2): Move two highest gain vertices v_m and v_n. Choose vertex v_m and its target subset S_m as for *Single_Move*. Choose randomly another target subset $S_n \in P$, $n \neq \max$ and $n \neq m$. Then, select a highest gain vertex v_n whose current subset is S_c, such that $S_c \in \{S_i \in P | S_i \neq S_n, S_i \neq S_m\}$. Move v_m to S_m, and v_n to S_n.

It is important to mention that a vertex v is considered to be moved to subset S_i if and only if v is adjacent to at least one vertex of S_i (v is a border vertex relative to S_i). Otherwise, moving v to S_i does not make any sense and would only deteriorate the partition quality. Our move operators focus thus on these critical vertices, reducing considerably the number of candidate moves to be examined at each iteration of the search process. Consequently, given the bucket data structure from Section 4.1.3, the worst-case complexity of choosing a vertex v to be moved to a subset S_i with any of the two move operators is equal to the number of border vertices relative to S_i.

Also note that these move operators progressively lead the search toward a balanced partition since basically they constraint (partially with *Double_Move*) vertex migration from heavy weight subsets to light weight subsets. Indeed, with *Single_Move* and the first choice of *Double_Move*, a vertex can never move to a subset with a higher weight. The second choice of *Double_Move* is allowed to bring some diversification into the search.

In the case of k-partitioning ($k > 2$), the two neighborhoods N_1 and N_2 are jointly explored by the tabu search algorithm in a token-ring way. In token-ring search [12], one neighborhood search is applied to the local optimum produced by the previous one, and this process continues until no improvement is possible. In the case of bisection ($k=2$), the neighborhood N_2 is not suitable and only N_1 is employed.

4.1.3. Bucket sorting

To ensure a fast evaluation of the neighborhoods, our implementation uses a bucket data structure which keeps vertices ordered by their gains. This data structure is used to avoid unnecessary search for the highest gain vertex and to minimize the time needed for updating the gains of vertices affected by each move.

The bucket sorting structure was first proposed by Fiduccia and Mattheyses [10] to improve the Kerninghan–Lin algorithm [19] for graph bisection. It consists in placing all vertices with the same gain g in a bucket that is ranked g. Then, finding a vertex with the maximum gain simply consists in finding the non-empty bucket with the highest rank, and selecting a vertex from the bucket (in MITS a vertex is selected from this bucket according to a policy detailed in Section 4.1.4). After each move, the bucket structure is updated by recomputing gains of the selected vertex and its neighbors, and transferring these vertices to appropriate buckets. The bucket data structure, as suggested in [10], maintains two arrays of buckets, one for each subset of a bisection, and is not suitable for k-partitioning.

We propose an adaptation of this idea for k-partitioning, designed to be used with the two neighborhood relations N_1 and N_2. In our implementation, we maintain k arrays of buckets, one for each subset of the k-partition, where each bucket of an array i, $1 \leq i \leq k$ is (as usually) represented by a double linked list. Each doubly linked list contains vertices with the gain corresponding to the rank of the link list in the given array of buckets. As proposed in [10], we also keep an additional array of vertices where each element (vertex) points to its corresponding vertex in the doubly linked lists. This enables a direct access to the vertices in doubly linked lists of each array. During the update of gains, this data structure also enables the transfer of vertices from one bucket to another in constant time.

In the search process, only border vertices are considered, and thus included in the bucket data structure. Therefore, given a vertex v with its current subset S_c, for every other subset S_m, $m \neq c$, we calculate the gain $g(v, m)$ in constant time, and place vertex v to the array of buckets corresponding to subset S_m only if there is at least one vertex in S_m adjacent to v, i.e. v is a border vertex relative to S_m. The gains of all the vertices adjacent to v are recalculated (in $O(1)$) and updated in the bucket structure in constant time (delete and insert operations in the bucket are both of $O(1)$ complexity). Therefore, the complexity of moving vertex v from S_c to a target subset S_m is equal to the number of vertices adjacent to v.

According to the complexity analysis, k has no influence on the performance of the proposed algorithm in terms of computing time. However, it does require a greater amount of memory as k increases.

An illustration of the proposed bucket sort data structure for 3-partitioning of a graph with seven vertices is provided in Fig. 1.

4.1.4. Selection strategy for vertex migration

As previously explained, when the non-empty bucket with the highest rank contains more than one vertex, a selection strategy is needed to determine the vertex v_m (and v_n) to be transferred to S_m (and v_n to S_n). Our selection strategy integrates several pieces of history information in order to make this choice as pertinent as possible.

The selection strategy is first conditioned by the tabu status (see Section 4.1.5). Let V_{cand} be the subset of candidate vertices (with the highest gain) for migration to subset S_m. A vertex $v \in V_{cand}$, whose current subset is S_c, is considered for migration to S_m if it is not tabu or moving v to S_m leads to a new partition better than the best partition P^* found so far (this later case is called *aspiration* in the tabu search terminology).

The vertex selection strategy employs two additional criteria which are based on *vertex move frequency* and *vertex weight*. The move frequency is a long term memory that records, for each vertex v, the number of times v has been moved to a different subset. Our usage of this frequency information penalizes moves with vertices having high frequency count, by giving priority to those that have been moved less often. This long term memory is reset to zero only before the beginning of each refinement phase (see line 2 in Algorithm 1). This strategy favors the diversification of the search.

If there is more than one vertex with the same move frequency in the set V_{cand}, we use the second criterion to distinguish them and prefer a vertex v which, when moved to subset S_m, minimizes the weight difference between the target subset S_m and the original subset S_c. This strategy helps to improve the partition balance.

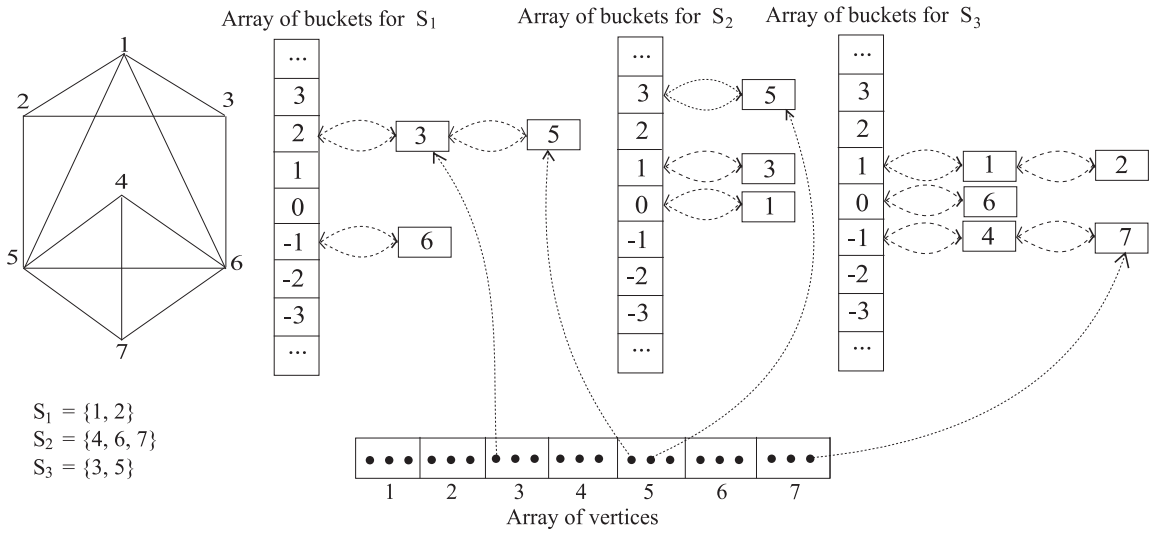

Fig. 1. An example of the bucket structure for 3-partitioning.

4.1.5. Tabu list and tabu tenure management

Each time a vertex v is moved from a subset S_c to another subset S_m, it is forbidden to move v back to its original subset S_c for the next tt iterations (tabu tenure). The tabu tenure tt of the move (v, S_c) is tuned dynamically according to the number of border vertices relative to S_c,

$$tt(v,S_c) = |V(S_c)| * \alpha + random(0,2),$$

where $|V(S_c)|$ is the number of border vertices relative to S_c, α a coefficient from {0.05,0.1,0.2,0.3}, and $random(0,2)$ a random integer number in the range [0,2].

4.2. Perturbation mechanism

The tabu search algorithm described above employs an aggressive search strategy to explore the search space. Indeed, for a transition from the current partition to a new partition, only border vertices relative to the target subset S_m are considered. Such a consideration helps the search to focus on critical vertices and improve rapidly the solution quality.

To complement this search strategy, a perturbation mechanism is applied to bring diversification into the search, if the best partition P^* is not improved during γ iterations (see lines 26–28 of Algorithm 2). The perturbation consists in moving p_{str} *selected* vertices (perturbation strength) to a given subset.

Precisely, let $\{S_1, S_2, \ldots, S_k\}$ be the current partition and let S_{max} be the subset with the maximum vertex weight, $S_{max} = \max_{i \in \{1,\ldots,k\}} \{W(S_i)\}$. Then the perturbation phase is defined as follows.

(1) Randomly select a subset $S_m \in \{S_1, S_2, \ldots, S_k\} - \{S_{max}\}$.
(2) Randomly choose a vertex v whose current subset is S_c, such that $S_c \in \{S | W(S) > W(S_m)\}$.
(3) Move the selected vertex v to subset S_m.
(4) Repeat steps (1)–(3) p_{str} times.

The described mechanism is similar to the neighborhood relation N_1. However, there is a significant difference. Not only does it neglect the tabu status of a vertex, but it also allows a vertex v to be moved to subset S_m even if it is not a border vertex relative to S_m. This leads the search to new areas and helps to escape from deep local optima (see also Section 6).

5. Experimental results

5.1. Benchmark instances

To evaluate the efficiency of the proposed approach, we carry out extensive experiments on a set of well-known benchmark graphs that are frequently used to assess graph partitioning algorithms. These graphs correspond to real-life problems arising in different applications. They can be downloaded from the University of Greenwich Graph Partitioning Archive[1] in the same format as used by JOSTLE, CHACO and METIS. Table 1 provides the whole set of the graphs together with their sizes, vertex degrees and types.

5.2. Experimental protocol

Our partition algorithm is programmed in C++, and compiled with GNU gcc on a Xeon E5440 with 2.83 GHz and 8GB.

To assess the performance of our MITS algorithm, we report computational results of two experiments: a comparison with two state-of-art graph partitioning packages (METIS [16] and CHACO [8]), and a comparison with the best partitions ever reported in the literature. The parameter settings of MITS applied in both experiments are given in Table 2, and are determined by a preliminary experiment using a small number of graph instances.

5.3. Comparison with the state-of-art solvers

In this section, we compare the partitions obtained by our approach in short computing time (limited to 2.5 min), with those obtained with the two packages METIS and CHACO. In both cases, we use the latest versions (METIS-4.0, CHACO-2.2). For METIS, we use the multilevel pMetis algorithm, and for CHACO we choose the multilevel KL algorithm with recursive bisection and a coarsening threshold of 100. Since pMetis and CHACO do not allow repeating runs in a randomized way, we run our MITS algorithm only once. In addition, we fix the parameter α (see Section 4.1.5) to 0.1. The cutoff limit depends on the size of graph, and is from 1 s (for graphs with up to 3000 vertices) to 2.5 min (for the largest graph *fe_ocean*).

Tables 3 and 4 present, for each graph and each value of k, the number of cutting edges of the partition obtained with the

[1] http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/

Table 1
The list of benchmark graphs together with their characteristics.

Graph	Size		Degree			Type
	$\|V\|$	$\|E\|$	Max	Min	Avg	
add20	2395	7462	123	1	6.23	20-bit adder
data	2851	15 093	17	3	10.59	3D FEM
3elt	4720	13 722	9	3	5.81	2D nodal graph
uk	4824	6837	3	1	2.83	2D dual graph
add32	4960	9462	31	1	3.82	32-bit adder
bcsstk33	8738	291 583	140	19	66.74	3D stiffness matrix
whitaker3	9800	28 989	8	3	5.92	2D nodal graph
crack	10 240	30 380	9	3	5.93	2D nodal graph
wing_nodal	10 937	75 488	28	5	13.80	3D nodal graph
fe_4elt2	11 143	32 818	12	3	5.89	2D FEM
vibrobox	12 328	165 250	120	8	26.8	Sparse matrix
bcsstk29	13 992	302 748	70	4	43.27	3D stiffness matrix
4elt	15 606	45 878	10	3	5.88	2D nodal graph
fe_sphere	16 386	49 152	6	4	5.99	3D FEM
cti	16 840	48 232	6	3	5.73	3D semi-structured graph
memplus	17 758	54 196	573	1	6.10	Memory circuit
cs4	22 499	43 858	4	2	3.90	3D nodal graph
bcsstk30	28 924	1 007 284	218	3	69.65	3D stiffness matrix
bcsstk31	35 588	572 914	188	1	32.197	3D stiffness matrix
fe_pwt	36 519	144 794	15	0	7.93	3D FEL
bcsstk32	44 609	985 046	215	1	44.1636	3D stiffness matrix
fe_body	45 097	163 734	28	0	7.26	3D FEM
t60k	60 005	89 440	3	2	2.98	2D dual graph
wing	62 032	121 544	4	2	2.57	3D dual graph
brack2	62 631	366 559	32	3	11.71	3D nodal graph
finan512	74 752	261 120	54	2	6.99	Stochastic programming matrix
fe_tooth	78 136	452 591	39	3	11.58	3D FEM
fe_rotor	99 617	662 431	125	5	13.30	3D FEM
598a	110 971	741 934	26	5	13.37	3D FEM
fe_ocean	143 437	409 593	6	1	5.71	3D dual graph

Table 2
Settings of important parameters.

Parameters	Description	Values
ct	Coarsening threshold	200
α	Tabu tenure management factor	[0.05, 0.1, 0.2, 0.3]
p_{str}	Perturbation strength	$0.02 * \|V\|$
γ	Non-improvement TS iterations before perturbation	$0.01 * \|V\|$

packages and our proposed algorithm. METIS, CHACO and our approach are labeled as pMetis, CHACO and MITS respectively. The last row in each table with heading *Total* gives the number of times each algorithm produced the best partition over the 30 benchmark graphs. From Tables 3 and 4, we observe that for each value of k MITS performs far better than either of the two packages in terms of partition quality.

From these tables, we also note that for $k \geq 32$, some partitions found with pMetis and our approach are not perfectly balanced. For these cases, we indicate the partition imbalance in parentheses. As it will be seen from the second experiment, MITS in some cases requires more computation time to establish partition balance. Unlike pMetis and our algorithm, CHACO generates perfectly balanced partitions for every value of k since it uses recursive bisection.

5.4. Comparison with the best known partitions

The second comparison is with the best balanced partitions reported at the graph partitioning archive. The majority of these best known results were generated with a very powerful algorithm presented by Soper et al. [25], which combines an evolutionary search approach with the JOSTLE multilevel procedure used as a black box. Since each run consists of 50,000 calls to JOSTLE, this approach requires significant running time of up to one week for large graphs. Another great portion of these best partitions were produced with the iterative multilevel algorithm by Walshaw [29] which also makes a number of calls to a multilevel procedure. The remaining best results are obtained with several other approaches [9,21,6]. It should be mentioned that the majority of these results were produced within very long computing time (up to one week for larger graphs).

Our second experiment assesses the performance of MITS algorithm in terms of partition quality relative to these state-of-the-art algorithms. For this experiment, we use cutoff time limits ranging from 1 min for the smallest graph up to 1 h for the largest one. Above this time limit, the algorithm does not produce significantly better results. Given its stochastic nature, the MITS algorithm is run 30 times for each graph and each value of k (see Table 2).

Tables 5 and 6 give, for each graph and each value of k, the best results ever reported, the best results produced by our approach,[2] the average results as well as the standard deviations after 30 executions of the proposed algorithm. The row with heading *Total* gives the number of times our approach succeeded to reach or improve a best known up-to-date partition.

The results show that, in the case of bisection, our approach succeeded to reach the same solution quality of more than two thirds of the best reported balanced bisections. It also improved the best bisection in two cases. Interestingly, as k increases ($4 \leq k \leq 64$), our approach improved even 60%, 86%, 90%, 90% and 76% of the

[2] Results available at: http://www.info.univ-angers.fr/pub/hao/MITS.html

Table 3
Comparison of our approach with pMetis and CHACO for k equal to 2, 4 and 8.

Graph	$k=2$			$k=4$			$k=8$		
	pMetis	CHACO	MITS	pMetis	CHACO	METIS	pMetis	CHACO	MITS
add20	729	742	**708**	1292	1329	**1224**	1907	1867	**1750**
data	218	199	**189**	480	433	**427**	842	783	**679**
3elt	108	103	**90**	231	234	**214**	388	389	**352**
uk	**23**	36	**23**	67	69	**47**	**101**	119	113
add32	21	**11**	**11**	42	56	**40**	81	115	**74**
bcsstk33	10 205	10 172	**10 171**	23 131	23 723	**22 492**	40 070	39 070	**34 568**
whitaker3	135	131	**127**	406	425	**385**	719	765	**672**
crack	187	225	**184**	382	445	**371**	773	777	**720**
wing_nodal	1820	1823	**1797**	4000	4022	**3715**	6070	6147	**5481**
fe_4elt2	**130**	144	**130**	**359**	402	423	654	718	**621**
vibrobox	12 427	11 367	**11 184**	21 471	21 774	**19 811**	28 177	33 362	**24 840**
bcsstk29	**2843**	3140	2852	8826	9202	**8572**	**16555**	18158	17014
4elt	154	158	**139**	406	433	**390**	635	688	**615**
fe_sphere	440	424	**386**	872	852	**774**	1330	1302	**1243**
cti	**334**	372	366	1113	1117	**1039**	2110	2102	**1838**
memplus	6337	7549	**5696**	10 559	11 535	**9982**	13 110	14 265	**12 642**
cs4	414	517	**377**	1154	1166	**987**	1746	1844	**1529**
bcsstk30	**6458**	6563	9812	17 685	**17 106**	22 436	**36 357**	37 406	36 373
bcsstk31	3638	3391	**2820**	8770	9199	**8751**	16 012	15 551	**15 262**
fe_pwt	366	362	**360**	**738**	911	1249	1620	1670	**1531**
bcsstk32	**5672**	6137	6936	12 205	15 704	**9864**	**23 601**	25 719	24 435
fe_body	311	1036	**271**	957	1415	**728**	1348	2277	**1293**
t60k	100	91	**86**	255	235	**226**	561	524	**522**
wing	950	901	**861**	2086	1982	**1770**	3205	3174	**2686**
brack2	738	976	**731**	**3250**	3462	3291	7844	8026	**7644**
finan512	**162**	**162**	**162**	**324**	325	405	810	**648**	729
fe_tooth	4297	4642	**3827**	8577	8430	**7460**	13 653	13 484	**12 083**
fe_rotor	2190	2151	**2122**	8564	8215	**7765**	15 712	15 244	**13 558**
598a	2504	2465	**2402**	8533	8975	**8159**	17 276	17 530	**16 270**
fe_ocean	505	499	**468**	**2039**	2110	2850	4516	5309	**4272**
Total	7	2	26	5	1	24	4	1	25

Table 4
Comparison of our approach with pMetis and CHACO for k equal to 16, 32 and 64.

Graph	$k=16$			$k=32$			$k=64$		
	pMetis	CHACO	MITS	pMetis	CHACO	MITS	pMetis	CHACO	MITS
add20	2504	2297	**2120**	NAN	2684	**2524** (1.03)	3433 (1.07)	3349	**3219** (1.03)
data	1370	1360	**1167**	2060 (1.01)	2143	**1933**(1.01)	3116 (1.03)	3145	**2924** (1.07)
3elt	665	660	**585**	1093	1106	**1053**	1710	1722	**1606** (1.03)
uk	189	211	**163**	316 (1.01)	343	**296** (1.01)	**495** (1.02)	540	496 (1.03)
add32	**128**	174	143	288 (1.01)	303	**266** (1.01)	626 (1.02)	730	**571** (1.01)
bcsstk33	59 791	61 890	**55 538**	86 008	**84 613**	90 438	116 203 (1.01)	**115 530**	131 639 (1.05)
whitaker3	1237	1218	**1120**	1891	1895	**1758**	2796 (1.01)	2811	**2628**
crack	1255	1253	**1157**	1890	1962	**1741**	2847 (1.01)	2904	**2628** (1.01)
wing_nodal	9290	9273	**8465**	13 237	13 258	**12 238**	17 899 (1.01)	17 783	**16 258** (1.01)
fe_4elt2	1152	1135	**1039**	1787	1796	**1688**	2765 (1.01)	2781	**2590**
vibrobox	37 441	43 064	**34 392**	**46 112**	51 006	47 048(1.01)	**53 764** (1.01)	58 392	54 503
bcsstk29	28 151	28 629	**26 055**	41 190	42 935	**38 346**	62 891 (1.01)	63 576	**59 548** (1.01)
4elt	1056	1083	**1005**	1769	1766	**1631**	2953	2921	**2676**(1.01)
fe_sphere	2030	2037	**1855**	2913	2920	**2701**	4191	4151	**3776**
cti	3181	3083	**3033**	4605	4532	**4479**	6461	6334	**6181**
memplus	14 942	16 433	**14 097**	17 303	17 936	NAN	19 140 (1.01)	**18 978**	NAN
cs4	2538	2552	**2293**	3579	3588	**3137**	4791	4817	**4286**
bcsstk30	**77 293**	81 069	79 265	131 405	128 694	**117 414**	191 691	**191 445**	175 845 (1.02)
bcsstk31	27 180	28 557	**25 787**	42 645	45 354	**40 029**	66 526	68 375	**61 150**
fe_pwt	2933	3200	**2857**	**6029**	6036	6596	9310	9231	**8487**
bcsstk32	43 371	47 829	**39 902**	70 020	73 377	**64 138**	106 733	108 855	**96 197**
fe_body	2181	2947	**2076**	3424	4194	**3290**	5843	6326	**5097**
t60k	998	977	**937**	1613	1594	**1539**	2484	2506	**2345**
wing	4666	4671	**4188**	6700	6843	**6067**	9405	9308	**8123**
brack2	12 655	13 404	**12 240**	19 786	20 172	**18 411**	28 872	29 223	**27 130**
finan512	1377	**1296**	1458	**2592**	**2592**	**2592**	**10 842**	11 962	11 077
fe_tooth	19 346	20 887	**18 336**	29 215	29 849	**26 110**	40 162	40 306	**35 988**
fe_rotor	23 863	23 936	**21 687**	36 225	36 367	**32 746**	53 623	52 497	**48 206**
598a	28 922	29 674	**26 565**	44 760	45 780	**40 980**	64 307	65 094	**57 303**
fe_ocean	9613	9690	**8397**	14 613	15 059	**13 358**	23 317	22 692	**21 212**
Total	2	1	27	3	2	26	3	3	24

Table 5
Comparison with the best partitions found in literature for k equal to 2, 4 and 8.

Graph	$k=2$				$k=4$				$k=8$			
	Best	b_MITS	Avg	Std	Best	b_MITS	Avg	Std	Best	b_MITS	Avg	Std
add20	**596**	636	709.5	27.94	1203	**1176**	1201.17	12.56	1714	**1697**	1713.20	19.43
data	**189**	**189**	190.97	3.2	**383**	**383**	396.63	10.68	679	**672**	680.16	8.49
3elt	**90**	**90**	90.0	0.0	**201**	**201**	205.73	2.97	348	**346**	349.33	4.41
uk	**20**	**20**	21.77	1.93	43	**42**	44.83	2.25	89	**86**	89.83	2.79
add32	11	**10**	11.16	1.00	34	**33**	34.3	1.95	75	**66**	69.3	1.84
bcsstk33	**10171**	**10171**	10227.5	293.87	**21719**	21779	22418.8	381.19	34579	**34464**	34567.1	66.57
whitaker3	**127**	**127**	127.0	0.0	**382**	**382**	383.4	2.73	661	**657**	661.9	2.52
crack	**184**	**184**	184.16	0.89	368	**366**	366.60	0.49	687	**680**	693.03	6.75
wing_nodal	**1707**	**1707**	1754.10	48.18	3581	**3577**	3625.53	32.83	5443	**5439**	5495.93	39.25
fe_4elt2	**130**	**130**	130.00	0.0	**349**	**349**	349.26	0.51	**610**	**610**	616.43	5.22
vibrobox	**10343**	**10343**	10372.5	38.83	19245	**19143**	19658.5	311.02	24715	**24609**	24838.6	126.75
bcsstk29	**2843**	**2843**	2851.57	2.65	**8159**	8475	8554.4	43.09	**14322**	15245	16177.4	482.5
4elt	**139**	**139**	141.83	10.6	**326**	327	340.57	13.62	548	**547**	562.76	16.78
fe_sphere	**386**	**386**	386.0	0.0	**770**	**770**	340.26	11.83	1193	**1167**	1185.93	18.52
cti	**334**	**334**	334.0	0.0	963	**955**	992.33	32.56	1812	**1797**	1844.07	32.11
memplus	**5513**	5672	5672.0	0.0	**9643**	9677	9823.5	148.29	11872	**11858**	12309.1	176.7
cs4	**371**	374	376.03	1.7	964	**939**	969.9	12.43	1496	**1451**	1473.93	11.75
bcsstk30	**6394**	**6394**	7997.0	2084.69	**16652**	16686	18249.5	1655.39	34921	**34898**	36428.0	3480.49
bcsstk31	**2762**	**2762**	3038.17	629.67	7469	**7395**	7845.3	412.0	13812	**13371**	13722.6	293.53
fe_pwt	**340**	**340**	358.83	5.04	709	**707**	720.5	6.53	1465	**1450**	1490.0	37.17
bcsstk32	**4667**	**4667**	5726.9	840.26	9492	**9401**	10499.8	1033.4	22757	**21102**	22709.1	589.93
fe_body	**262**	**262**	264.57	8.31	703	**621**	674.1	39.5	1234	**1048**	1093.1	27.08
t60k	**79**	82	94.83	20.61	**213**	219	225.16	6.34	**476**	478	496.03	12.11
wing	**791**	807	835.53	19.86	1666	**1638**	1692.13	32.51	2589	**2517**	2558.93	23.11
brack2	**731**	**731**	731.0	0.0	3090	**3084**	3142.63	57.22	7269	**7144**	7383.0	142.12
finan512	**162**	**162**	162.0	0.0	**324**	**324**	429.3	89.1	**648**	**648**	723.6	58.90
fe_tooth	3850	**3823**	3978.23	152.25	7142	**6919**	7108.4	136.02	11935	**11480**	11674.7	132.6
fe_rotor	**2098**	**2098**	2106.5	11.17	7480	**7294**	7742.33	225.55	13292	**12864**	13203.1	228.44
598a	**2398**	2399	2403.07	3.7	8154	**8032**	8234.7	402.18	16884	**15927**	16223.1	197.89
fe_ocean	**464**	**464**	467.76	0.88	1902	**1886**	1925.2	113.25	4299	**4216**	4303.5	83.99
Total	28	24			12	24			4	28		

Table 6
Comparison with the best partitions found in literature for k equal to 16, 32 and 64.

Graph	$k=16$				$k=32$				$k=64$			
	Best	b_MITS	Avg	Std	Best	b_MITS	Avg	Std	Best	b_MITS	Avg	Std
add20	2149	**2063**	2090.07	13.95	2687	**2406** (1.01)	2420.5	10.12	3236	**3108**	3081.33	23.98
data	1162	**1146**	1164.13	14.23	1865	**1838** (1.01)	1863.1	14.93	**2798**	2862 (1.04)	2878.1	9.74
3elt	581	**576**	581.0	3.84	**969**	970	975.76	3.2	1564	**1559** (1.01)	1569.13	4.69
uk	159	**153**	159.76	3.33	280	**265** (1.01)	275.17	4.84	**438**	463	459.23	5.65
add32	121	**117**	128.00	5.55	230	**212** (1.01)	224.4	11.99	**493**	522 (1.01)	540.46	10.35
bcsstk33	55 136	**54 841**	55 500.1	413.68	78 132	**78 054**	83 549.0	5924.61	108 505	**107 980** (1.01)	110 091.0	7876.97
whitaker3	1108	**1096**	1105.2	5.17	1718	**1702**	1714.13	6.79	2569	**2558**	2572.03	7.77
crack	1108	**1096**	1120.97	12.26	1728	**1695**	1716.4	12.69	**2566**	2569 (1.01)	2586.57	8.05
wing_nodal	8422	**8353**	8405.87	13.48	12 080	**11 844**	11 942.3	45.35	16 134	**15 904** (1.01)	16 003.7	48.99
fe_4elt2	1018	**1010**	1016.1	5.24	1657	**1638**	1650.73	7.44	2537	**2519**	2541.27	8.48
vibrobox	32 654	**32 197**	33 028.7	376.65	0 42 187	**40 053**	40 874.7	916.51	49 521	**47 651** (1.01)	51 684.2	2344.16
bcsstk29	**22 869**	23 262	25 541.5	276.16	36 104	**35 422**	36 445.7	547.21	**57 054**	57 074 (1.01)	58 833.4	2099.54
4elt	956	**942**	958	11.15	1592	**1564**	1590.0	16.43	2636	**2595**	2618.33	11.50
fe_sphere	1750	**1737**	1742.3	2.78	2567	**2543**	2569.7	14.14	3663	**3634**	3672.0	16.45
cti	2909	**2867**	2927.5	27.48	4288	**4163**	4237.5	33.85	5955	**5885**	5976.0	41.56
memplus	13 516	**13 062**	13 170.9	93.33	14 634	**14 220**	14 533.0	167.61	17 446	**16 665**	16 433.0	280.88
cs4	2206	**2105**	2135.73	15.68	3110	**2952**	2992.77	16.97	4223	**4066**	4106.17	17.75
bcsstk30	72 007	**70 681**	73 133.9	1411.9	119 164	**114 493**	116 811.0	977.22	173 945	**172 929**	174 456.0	944.45
bcsstk31	24 551	**23 930**	24 479.7	261.95	38 484	**37 652**	39 004.7	597.29	60 724	**58 625**	59 305.5	388.62
fe_pwt	2855	**2838**	2848.9	12.82	5758	**5683**	5813.9	110.35	8495	**8335**	8379.17	21.93
bcsstk32	38 711	**36 650**	37 650.3	700.91	63 856	**61 211**	62 542.6	672.13	95 199	**92 717**	94 543.3	778.54
fe_body	2057	**1774**	1849.5	41.37	3371	**2935**	2994.8	33.72	5460	**4879**	4963.7	51.06
t60k	**866**	880	899.0	7.93	**1440**	1441	1462.67	14.69	**2233**	2259	2279.83	10.45
wing	4198	**3890**	3956.77	25.69	6009	**5649**	5725.83	35.43	8132	**7712**	7782.07	36.58
brack2	12 323	**11 649**	11 941.0	125.81	18 229	**17 490**	17 780.0	141.62	27 178	**26 088**	26 490.6	159.6
finan512	**1296**	**1296**	1344.6	53.73	**2592**	**2592**	2592.0	0.0	**10 560**	**10 560**	10 772.8	84.04
fe_tooth	18 382	**17 437**	17 690.6	134.7	26 346	**25 045**	25 385.1	156.9	35 980	**34 593**	34 935.9	128.05
fe_rotor	21 241	**20 483**	20 888.8	178.96	32 783	**31 383**	32 004.9	261.9	49 381	**46 110**	46 608.4	291.39
598a	26 427	**25 817**	26 115.2	129.38	41 538	**38 693**	38 991.2	138.45	59 708	**56 378**	56 798.5	254.16
fe_ocean	8622	**7771**	7987.6	112.46	14 277	**12 811**	13 091.0	109.16	22 301	**20 068**	10 772.8	84.04
Total	3	28			3	28			7	24		

current best k-partitions where k is equal to 4, 8, 16, 32 and 64 respectively.

As in the previous comparison, we note from Table 6 that in 12 cases where k is equal to 32 and 64, our approach did not attain the perfect partition balance ($\varepsilon = 1.00$). However, this imbalance is generally low, with an average of 1.013 over these 12 imbalanced partitions. We also note that the number of imbalanced partitions is now slightly minimized compared to the first experiment when the running time was reduced. This implies that MITS sometimes requires more iterations to establish good balance. If the partitions are imbalanced, we compare them with the best known partitions of the same balance, which are also reported at the graph partitioning archive.

All these results show that the proposed approach is the overall winner when it comes to generating balanced k-partitions for these benchmark graphs.

6. Analysis

In this section, we turn our attention to the analysis of two important features of the MITS algorithm and try to answer the following questions: Is the perturbation mechanism (see Section 4.2) relevant for the MITS? Is the newly proposed *Double move*, which is used to define the neighborhood relation N_2 (see Section 4.1.2), a value-added one?

To answer these questions, we report in Table 7 the average results over 10 executions with MITS, when its two components (perturbation mechanism and neighborhood N_2) are included and excluded. We conduct these experiments with k set to 4, 8, 16 and 32, on a set of 10 representative instances from the graph partitioning archive. For each k, column "$+(PM\ \&\ N_2)$" provides the average results when both the perturbation mechanism and the neighborhood N_2 are used within MITS. columns "$-PM$" and "$-N_2$" report respectively the average results when the perturbation mechanism or the neighborhood N_2 is disabled. Finally, the last column "$-(PM\ \&\ N_2)$" gives the average results when both features are excluded from the MITS algorithm. The last row with heading *Total* shows the number of times each version of MITS produced the best average result over the 10 graphs.

From Table 7, we see that the performance of our MITS algorithm decreases when the perturbation mechanism or neighborhood N_2, or both are disabled. This confirms our previous observation that some diversification is needed to complement the aggressive tabu search strategy, which only considers border (critical) vertices. Indeed, although examining only border vertices for move consideration decreases the number of candidate moves at each iteration, this does not allow the search to escape from deep local optima. Therefore, this experiment confirms this intuitive explanation.

We can also see from Table 7 that in most cases, MITS produces significantly better partitions when the two neighborhoods N_1 and N_2 are jointly explored (see Section 4.1.2). This can be explained by the fact that neighborhood N_2 does not strictly impose the balance constraint as neighborhood N_1 does. With N_2, there are more vertices which are considered to be moved to the target subset, introducing thus a greater degree of freedom during the search process. This complements the strict selection criterion of neighborhood N_1.

From Table 7 we can see that in several cases the simplest version of MITS, i.e. when both features are excluded, gives the best average results. However, as can be expected, the initial MITS still outperforms the simplest version in about 80% of cases. This implies that both of these elements are important for the efficiency of the proposed MITS algorithm.

7. Conclusion and remarks

In this paper, we presented MITS, a multilevel iterated tabu search approach for balanced graph k-partitioning. The proposed algorithm follows the basic idea of the multilevel approach and

Table 7
The average results over 10 executions of MITS when the perturbation mechanism and the neighborhood relation N_2 are included and excluded.

Graph	$k=4$				$k=8$			
	$+(PM\ \&\ N_2)$	$-PM$	$-N_2$	$-(PM\ \&\ N_2)$	$+(PM\ \&\ N_2)$	$-PM$	$-N_2$	$-(PM\ \&\ N_2)$
3elt	**206.1**	253.0	223.2	255.3	**349.8**	359.3	372.2	377.1
uk	**44.9**	64.0	47.7	61.9	**92.2**	95.5	98.3	102.5
bcsstk33	22 180.8	22 310.9	**22 166.9**	22 579.9	34 581.5	**34 563**	34 598.3	34 614.1
crack	**367.3**	452.4	388.2	503.4	**695.1**	735.1	731.4	754.0
fe_4elt2	**350.2**	379.8	350.3	402.7	**617.3**	634.1	645.0	645.5
vibrobox	**19 601.7**	23 114.2	21 757.3	23 649.7	**24 804.6**	27 726.1	27 904.6	27 405.0
4elt	340.4	387.8	345.2	**320.3**	557.6	558.2	565.3	**531.5**
cs4	**967.2**	1093.6	980.6	1110.3	**1478.6**	1579.3	1485.2	1579.4
t60k	232.2	258	224.6	**320.3**	**490.4**	533.0	498.6	531.5
brack2	**3149.4**	4349.8	3150.3	4499.8	**7404.2**	7665.1	8489.7	7986.6
Total	7	0	1	2	8	1	0	1
Graph	$k=16$				$k=32$			
	$+(PM\ \&\ N_2)$	$-PM$	$-N_2$	$-(PM\ \&\ N_2)$	$+(PM\ \&\ N_2)$	$-PM$	$-N_2$	$-(PM\ \&\ N_2)$
3elt	**582**	586.4	630.8	625.6	**976.9**	992.3	984.8	980.6
uk	**160.8**	171.0	163.7	171.8	**275.9**	284.2	292.0	288.2
bcsstk33	55 572.8	62 445	55 541.3	**55 548.8**	88 996.9	85 913.5	**78 891.5**	79 327.9
crack	**1116.7**	1148.5	1184.9	1182.4	**1713.4**	1748.7	1820.1	1822
fe_4elt2	**1015.4**	1024.6	1020.4	1027.9	**1652.0**	1703.3	1680.2	1696
vibrobox	**33 250.4**	36 704.9	33 471.8	33 621.2	41 074.6	51 313	**40 654.5**	42 962.5
4elt	962.5	960.6	962	**932.5**	1581.5	1638.6	1605.3	**1506.4**
cs4	**2146.6**	2248.4	2163	2253.6	**3003.4**	3294.5	3152.4	3159.2
t60k	901.8	**895.0**	908.3	932.5	**1469.0**	1484.0	1499.7	1506.4
brack2	11 995.7	12 536.1	**11 962.1**	12 197.0	**17 814.0**	18 691.5	17 907.3	18 051.5
Total	6	1	1	2	6	0	2	1

integrates a new and powerful tabu search refinement procedure. The dedicated tabu search algorithm, which is the key component of the approach, includes a number of interesting features like the joint use of two neighborhoods, an adaptation of bucket sorting for incremental evaluation of move gains, guided selection strategies for vertex migration based on long term memory (move frequency) and a dynamic tabu tenure technique. To reinforce the diversification of the search process, the tabu search algorithm is combined with a perturbation mechanism to escape from deep local optima.

We assessed extensively the efficiency of the proposed algorithm with both short and long run times, on a collection of benchmark graphs provided by the graph partitioning archive, with the cardinal number k set to 2, 4, 8, 16, 32 and 64. The results generated in short computation time (from 1 s up to 2.5 min) are far better than those produced by the two well-known partitioning packages METIS or CHACO. When the running time is prolonged from 1 min up to 1 h, our approach succeeds even to improve more than two thirds of the best balanced partitions ever reported in the literature (which were often obtained with much longer computing time up to one week).

Finally, it is recognized that there may be a significant trade-off between partition quality and balance. Indeed, allowing a certain degree of imbalance usually leads to partitions of better quality. In this work, we aimed to produce perfectly balanced partitions, and did not specifically address the possibility to allow imbalance up to a certain limit. Yet, we tested our MITS algorithm with the balance constraint relaxed up to certain degree. We observed that the quality of these imbalanced partitions, even if they are not reported here, remains highly competitive with respect to the best known partitions of the same imbalance. In fact, we managed to improve once again a number of the current best partitions, although the percentage of these improved partitions was lower than in the case of balanced partitions.

Acknowledgments

We are grateful to the referees for their comments and questions which helped us to improve the paper. The work is partially supported by the Pays de la Loire Region (France) within the projects RaDaPop (2009–2013) and LigeRO (2010–2013).

References

[1] Alpert CJ, Kahng AB. Recent directions in netlist partitioning. Integration: The VLSI Journal 1995;19(12):1–81.
[2] Barnard ST, Simon HD. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. Concurrency: Practice & Experience 1994;6(2):101–17.
[3] Battiti R, Bertossi A. Greedy and prohibition-based heuristics for graph partitioning. IEEE Transactions on Computers 1999;48:361–85.
[4] Bui TN, Jones C. A heuristic for reducing fill-in in sparse matrix factorization. In: Sincovec RF, editor. Parallel processing for scientific computing. Philadelphia: SIAM; 1993. p. 445–52.
[5] Bui TN, Moon BR. Genetic algorithm and graph partitioning. IEEE Transactions on Computers 1996;45:841–55.
[6] Chardaire P, Barake M, McKeown GP. A PROBE based heuristic for graph partitioning. IEEE Transactions on Computers 2007;56(12):1707–20.
[7] Dell'Amico M, Maffioli F. A tabu search approach to the 0–1 equicut problem. In: In metaHeuristics 1995: the state of the art. Kluwer Academic Publishers; 1996. p. 361–77.
[8] Hendrickson B, Leland R. A multilevel algorithm for partitioning graphs. In: Karin S, editor. Proceedings of the 1995 ACM/IEEE conference on supercomputing (CDROM). New York: ACM; 1995. article No. 28.
[9] Holtgrewe M, Sanders P, Schulz C. Engineering a scalable high quality graph partitioner. Technical Report; 2009.
[10] Fiduccia C, Mattheyses R. A linear-time heuristics for improving network partitions. In: Proceedings of the 19th design automation conference; 1982. p. 171–85.
[11] Garey M, Johnson D. Computers & intractability: a guide to the theory of NP-completeness. W. H. Freeman and Company; 1979.
[12] Di Gaspero L, Schaerf A. Neighborhood portfolio approach for local search applied to timetabling problems. Journal of Mathematical Modeling and Algorithms 2006;5(1):65–89.
[13] Glover F, Laguna M. Tabu Search. Boston: Kluwer Academic Publishers; 1997.
[15] Johnson DS, Aragon CR, Mcgeoch LA, Schevon C. Optimization by simulated annealing: an experimental evaluation. Part-I: graph partitioning. Operations Research 1989;37:865–92.
[16] Karypis G, Kumar V. MeTiS 4.0: unstructured graphs partitioning and sparse matrix ordering system. Technical Report, Department of Computer Science, University of Minnesota; 1998.
[17] Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 1998;20(1):359–92.
[18] Karypis G, Kumar V. Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing 1998;48(1):96–129.
[19] Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal 1970;49:291–307.
[20] Merz P, Freisleben B. Fitness landscapes, memetic algorithms and greedy operators for graph bi-partitioning. Evolutionary Computation 2000;8(1):61–91.
[21] Meyerhenke H, Monien B, Sauerwald T. A new diffusion-based multilevel algorithm for computing graph partitions of very high quality. In: Proceedings of the 22nd international parallel and distributed processing symposium (IPDPS'08); 2008. p. 1–13.
[22] Papadimitriou C, Steiglitz K. Combinatorial optimization: algorithms and complexity. Prentice-Hall; 1982.
[23] Rolland E, Pirkul H, Glover F. Tabu search for graph partitioning. Annals of Operations Research 1996;63:209–32.
[24] Shi J, Malik J, Normalized cuts and image segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition; 1997. p. 731–7.
[25] Soper AJ, Walshaw C, Cross M. A combined evolutionary search and multilevel optimisation approach to graph-partitioning. Journal of Global Optimization 2004;29(2):225–41.
[26] Slowik A, Bialko M. Partitioning of VLSI circuits on subcircuits with minimal number of connections using evolutionary algorithm. In: Rutkowski L, editor. ICAISC 2006, Lecture notes in artificial intelligence, vol. 4029. Springer-Verlag; 2006. p. 470–8.
[27] Talbi E-G, Bessiere P. A parallel genetic algorithm for the graph partitioning problem. In: Proceedings of the 5th international conference on supercomputing; 1991. p. 312–20.
[28] Walshaw C, Cross M. Mesh partitioning: a multilevel balancing and refinement algorithm. SIAM Journal on Scientific Computing 2000;22:63–80.
[29] Walshaw C. Multilevel refinement for combinatorial optimisation problems. Annals of Operations Research 2004;131:325–72.
[30] Zha H, He X, Ding C, Simon H, Gu M. Bipartite graph partitioning and data clustering. In: Proceedings of the ACM 10th international conference on information and knowledge (CIKM 2001); 2001. p. 25–31.