# An Effective Multilevel Memetic Algorithm for Balanced Graph Partitioning

Una Benlic and Jin-Kao Hao
LERIA, Université d'Angers,
2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
Email: benlic,hao@info.univ-angers.fr

***Abstract*—The balanced graph partitioning consists in dividing the vertices of an undirected graph into a given number of subsets of approximately equal size, such that the number of edges crossing the subsets is minimized. In this work, we present a multilevel memetic algorithm for this NP-hard problem that relies on a powerful grouping recombination operator and a dedicated local search procedure. The proposed operator tends to preserve the backbone with respect to a set of parent individuals, i.e. the grouping of vertices which is same throughout each parent individual. Although our approach requires significantly longer computing time compared to some current state-of-art graph partitioning algorithms such as SCOTCH, METIS, CHACO, JOSTLE, etc., it competes very favorably with these approaches in terms of solution quality. Moreover, it easily reaches or improves on the best partitions ever reported in the literature.**

***Index Terms*—Graph partitioning, grouping recombination operator, local search, backbone.**

## I. Introduction

Graph partitioning is probably one of the most studied combinatorial optimization problems, which is widely applicable to many areas including VLSI design, data mining, image segmentation, etc. Since this problem is NP-complete [5], many heuristic methods have been devised to address this problem. These methods can be classified as move-based approaches, approaches based on meta-heuristics, and clustering algorithms. For a detailed description and comparison of these approaches, see the survey by Alpert and Kahng [1].

In the field of combinatorial optimization, hybrid evolutionary algorithms often produce very competitive results, for example for the graph coloring problem [11], [13], traveling salesman problem [4], and even the graph partitioning problem [14]. In general, these hybrid algorithms give better results if high quality solutions, produced by an efficient local search approach, are recombined in a meaningful manner.

In this work, we report on an original recombination operator specially devised for the partitioning problem. It is motived by the observation that, given a certain number of high quality solutions of a graph instance, there is always a very big number of vertices which are grouped together throughout these solutions. The proposed recombination, combined with a multilevel algorithm based on hill climbing with periodic perturbations, leads to a very effective algorithm. Indeed, experiments on a collection of representative benchmark instances from the Graph Partitioning Archive show that our multilevel memetic algorithm easily reaches or improves almost all the best known results reported so far. Moreover, even compared to the *imbalanced* partitions obtained by two of the most effective approaches, the *balanced* partitions produced by our approach remain competitive.

This paper is organized as follows. In Section 2, we provide the formal graph partitioning description and notations employed throughout this paper, and present the benchmark graphs used in this work. In Section 3, we briefly describe the multilevel paradigm and the general multilevel memetic algorithm. In Section 4, we present the proposed memetic refinement approach as well as the new recombination operator. Finally, in Section 5, we show computational results and comparisons.

## II. Graph partitioning

### A. Problem description and notation

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

Throughout this paper, we only work on graphs that have both vertices and edges of a unit cost weight. However, vertex and edge weights vary during the execution of a multilevel approach (see Section III). Let $|v|$ denote the weight of a vertex $v \in V$. Then, the weight $W(S_i)$ of a subset $S_i$ is equal to the sum of weights of the vertices in $S_i$, $W(Si) = \sum_{v \in S_i} |v|$.

The function $\pi$ induces a graph $G_\pi = G_\pi(S, E_c)$, where $S = \{S_1, S_2, ..., S_k\}$ and an edge $\{S_x, S_y\} \in E_c$ exists if there are two adjacent vertices $v_1, v_2 \in V$ mapped to different subsets, $v_1 \in S_x$ and $v_2 \in S_y$. The set $E_c$ corresponds to the set of cutting edges (i.e. edges that have endpoints in two different subsets).

There is often a trade-off between partition quality and imbalance, since allowing more imbalance may lead to partitions of better quality, i.e. partitions with a smaller number of cutting edges. An optimal subset weight is defined by $W_{opt} = \lceil |V|/k \rceil$, where $\lceil x \rceil$ is the ceiling function returning the first integer $\geq x$. In some applications, a small partition imbalance $\varepsilon$ is allowed, and can be defined as the maximal subset weight divided by the optimal weight, $\varepsilon = max_{i \in \{1..k\}} W(S_i)/W_{opt}$.

In this work, we aim to find partitions of perfect balance ($\varepsilon = 1.00$), while minimizing the sum of the edge weights in

$E_c$.

### B. Benchmarks

To evaluate our approach, we use all the graph instances from a set of benchmark graphs employed in [14]. These instances can be downloaded from the Graph Partitioning Archive http://staffweb.cms.gre.ac.uk/∼c.walshaw/partition/, which is maintained by the University of Greenwich. Table I gives the main characteristics of the graphs. Computational results and comparisons on these graphs are reported in Section V.

## III. The general multilevel algorithm

The basic idea of a multilevel approach, as proposed in [2], [7], is to match pairs of vertices to form clusters that define a new smaller graph (coarsening phase). This matching procedure is recursively applied until the graph size falls below a certain threshold. Afterwards, an initial partition of the coarsest graph is generated (initial partitioning phase), and then successively projected towards the original graph, followed by partition refinement (uncoarsening phase). Our multilevel approach that employs memetic partition refinement is presented in Algorithm 1.

Creating a coarser graph $G_{i+1}(V_{i+1}, E_{i+1})$ from $G_i(V_i, E_i)$ (lines 1–5 of Algo. 1) consists in finding an independent subset of graph edges, and then collapsing vertices that are incident on each edge. A set of edges is independent if no two edges in the set share the same vertex, which implies that exactly two vertices are collapsed during the matching. If a vertex is not incident on any edge of the matching, it is simply copied over to $G_{i+1}$. When vertices $v_1, v_2 \in V_i$ are collapsed to form a (clustered) 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 $v_2$, while 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 $\{v_1, v_2\} \in E_i$. So, the weight of a (clustered) vertex $v_c$ (edge $e_c$ respectively) in a coarse graph represents the number of vertices (edges) of the initial graph $G_0$ it aggregates in $v_c$ ($e_c$ respectively). Most of the multilevel approaches for graph partitioning use the same heuristics in the coarsening phase. For details on different coarsening schemes see [10].

To create each individual of the initial population in the second phase (line 6 of Algo. 1), we first assign randomly the vertices of the coarsest graph $G(V_m, E_m)$ to subsets $S_i \in \{S_1, S_2, ..S_k\}$, such that the number of clustered vertices in each subset is evenly balanced, i.e. the number $n$ of vertices in each subset $S_i$ is $n \leq \lceil |V_m|/k \rceil$. Afterwards, we apply a short run of local search to improve all individuals of this initial population (see Section IV-D), followed by the memetic refinement which is described in Section IV (lines 7–8 of Algo. 1).

The partition projection from a graph $G_i(V_i, E_i)$ onto a partition of the parent graph $G_{i-1}(V_{i-1}, E_{i-1})$ (line 11 of Algo. 1) is a trivial process. If a vertex $v \in V_i$ is in subset $S_m$, then the matched pair of vertices $v_1, v_2 \in V_{i-1}$ which represents vertex $v \in V_i$ will also be in subset $S_m$. Before projecting a partition on to the next level, we first apply a short run of local search to improve all the individuals of population $POP$, which is immediately followed by the memetic refinement (lines 12–13 of Algo. 1).

**Algorithm 1** The general scheme of the proposed multilevel algorithm

**Require:** An undirected graph $G_0(V_0, E_0)$ and the number of subsets $k$
**Ensure:** A partition of $G_0$
1: $i := 0$
2: **while** $|V_i| > coarsening\_threshold$ **do**
3: $G_{i+1}$ = **Coarsen**$(G_i)$
4: $i := i + 1$
5: **end while**
6: $POP_i$ = Initial_Partition$(G_i)$
7: $POP_i$ = **Short_Hill_Climbing**$(POP_i)$ {Section IV-D}
8: $POP_i$ = **Memetic_Refinement**$(POP_i)$ {Section IV}
9: **while** $i > 0$ **do**
10: $i := i - 1$
11: $POP_i$ = Project$(POP_{i+1}, G_i)$
12: $POP_i$ = **Short_Hill_Climbing**$(POP_i)$
13: $POP_i$ = **Memetic_Refinement**$(POP_i)$
14: **end while**

## IV. The memetic refinement

The proposed memetic algorithm incorporates a powerful recombination operator, which takes into consideration the backbone with respect to a subset of individuals from the population. After each recombination, an efficient local search with periodic perturbations is applied to the newly formed individual. Finally, we apply a replacement strategy that considers both the partition quality and the distance between individuals in the population. In this section, we describe the main components of the proposed memetic algorithm.

### A. Encoding and fitness function

Given a graph $G = (V, E)$ and a number of subsets $k$ (numbered from 1 to $k$), an individual $I$ corresponds to a partition of $V$ into $k$ disjoint groups or subsets $I = \{S_1, S_2, ..., S_k\}$, such that $S_i$ consists of all the vertices that are assigned to the $i^{th}$ subset.

The optimization objective of our $k$-partitioning problem is to minimize the sum of edge weights in $E_c$, while maintaining the best possible balance between partition subsets. Therefore, we use two fitness functions during the search.

The first function $f_1(I)$ is the initial optimization objective, i.e. the minimization of the weighted edges of $E_c$. Let $|V_0|$ be the number of vertices in the original graph $G_0(V_0, E_0)$. The second fitness function $f_2(I)$ evaluates the partition imbalance as follows:

$$f_2(I) = \begin{cases} \sum_{i=1}^{k} abs(W(S_i) - |V_0|/k), & |V_0| \ \ even; \\ \sum_{S_i \in \{S | W(S) \neq \lceil |V_0|/k \rceil\}} abs(W(S_i) - |V_0|/k), & otherwise. \end{cases} \quad (1)$$

TABLE I
THE LIST OF BENCHMARK GRAPHS TOGETHER WITH THEIR CHARACTERISTICS

| Graph | Size $\|V\|$ | $\|E\|$ | Degree Max | Min | Avg | Type |
|---|---|---|---|---|---|---|
| uk | 4824 | 6837 | 3 | 1 | 2.83 | 2D dual graph |
| add32 | 4960 | 9462 | 31 | 1 | 3.82 | 32-bit adder |
| crack | 10240 | 30380 | 9 | 3 | 5.93 | 2D nodal graph |
| wing-nodal | 10937 | 75488 | 28 | 5 | 13.80 | 3D nodal graph |
| vibrobox | 12328 | 165250 | 120 | 8 | 26.81 | Sparse matrix |
| 4elt | 15606 | 45878 | 10 | 3 | 5.88 | 2D nodal graph |
| cti | 16840 | 48232 | 6 | 3 | 5.73 | 3D semi-structured graph |
| bcsstk32 | 44609 | 985046 | 215 | 1 | 44.16 | 3D stiffness matrix |
| cs4 | 22499 | 43858 | 4 | 2 | 3.90 | 3D nodal graph |
| t60k | 60005 | 89440 | 3 | 2 | 2.98 | 2D dual graph |
| wing | 62032 | 121544 | 4 | 2 | 2.57 | 3D dual graph |
| brack2 | 62631 | 366559 | 32 | 3 | 11.71 | 3D nodal graph |

Then, individual $I^A$ is considered better than individual $I^B$ if $f_1(I^A) < f_1(I^B)$ and $f_2(I^A) \leq f_2(I^B)$.

### B. Distance measure

To determine the distance between two individuals $I^A = \{S_1^A, S_2^A, .., S_k^A\}$ and $I^B = \{S_1^B, S_2^B, .., S_k^B\}$, we use the well-known set-theoretic distance [6] (call it $d$), which is the minimum number of one-move steps needed to transform $I^A$ to $I^B$, i.e. $d(I^A, I^B) = |V| - sim(I^A, I^B)$, where $sim(I^A, I^B)$ is the similarity function.

Given the partition encoding from Section IV-A, the similarity function $sim(I^A, I^B)$ is defined as $max_{\sigma \in \Gamma} \sum_{i=1}^{k} M_{i,\sigma(i)}$, where $\Gamma$ is the set of all the possible permutations of $\{1, 2, .., k\}$ and $M$ a matrix with elements $M_{i,j} = |S_i^A \cap S_j^B|$. This function $sim(I^A, I^B)$, which reflects structural similarity, corresponds to the number of elements that do not need to be moved to transform $I^A$ to $I^B$.

### C. The recombination operator

*1) Notion of backbone and motivation for the proposed operator:* For optimization and approximation problems, the term backbone is used to define a set of variables $B$ having the same value throughout each solution from the set of all the global optima $S_{opt}$, while the backbone size corresponds to the number of elements in $B$. Then, in the graph partitioning problem, the backbone is the set of vertices which are grouped together throughout each solution of $S_{opt}$.

Since it is impossible to determine the exact backbone even for the smallest instances of graph partitioning, for this work we use a more relaxed notion of backbone, which is defined by the set of vertices that are grouped in the same way throughout each high quality solution.

The new recombination operator is based on the observation that, given a set $Q$ of high quality partitions of a graph instance, the backbone size can sometimes even exceed 90% of the total number of vertices. If there is a significant number of vertices that are grouped together throughout each high quality solution, there is a strong chance that they are also grouped together in the globally optimal solution. Therefore, the idea of our proposed operator is to preserve vertex grouping that is common to a number of population individuals, and perturb (with a certain probability) only vertices that do not belong to this grouping.

TABLE II
BACKBONE SIZE FOR 7 GRAPHS EXPRESSED AS A PERCENTAGE OF $|V|$.

| Graph | 4 | 8 | 16 | 32 |
|---|---|---|---|---|
| data | 95.2 | 64.7 | 42.2 | 50.9 |
| 3elt | 97.3 | 69.6 | 77.2 | 61.5 |
| uk | 68.4 | 51.4 | 30.0 | 61.5 |
| crack | 98.9 | 95.4 | 79.6 | 33.8 |
| wing-nodal | 91.5 | 96.0 | 33.2 | 15.5 |
| fe-4elt2 | 88.0 | 62.4 | 73.5 | 34.3 |
| vibrobox | 40.7 | 52.4 | 7.5 | 3.2 |

Table II reports for seven graphs of different types from the Graph partitioning archive, the backbone size with respect to the best known solution and best local optima found after 1500 independent runs of an effective iterated tabu search algorithm. The backbone size is expressed as the percentage of the number of vertices $|V|$.

*2) Backbone-based recombination operator (BRO):* Given a number of individuals $p$ ($p \geq 2$) which take part in the recombination process, we select a subset $P$ of size $p$ from population $POP$ with the tournament selection strategy. Let $\lambda$ be the size of the tournament pool. We select each individual $I^i \in P$ in the following way: randomly choose $\lambda$ individuals from $POP$; among the $\lambda$ chosen individuals, place the best one into subset $P$ if it is not already present in $P$. We then set the $p^{th}$ individual to be the reference individual $I^R = \{S_1^R, S_2^R ... S_k^R\}$ for the recombination operator which follows.

Given the partition encoding from Section IV-A, BRO can be formally described as follows. Let $\prod = \{\prod_1, \prod_2, ..., \prod_{p-1}\}$ be the set of vertex subsets, where each element $\prod_i$, $i = 1, 2, .., p-1$, contains the largest number of vertices that are shared by $I^R$ and individual $I^i = \{S_1^i, S_2^i ... S_k^i\} \in P, I^i \neq I^R$, i.e.

$$\prod_i = \{\{S_1^R \cap S_{\sigma(1)}^i\} \cup \{S_2^R \cap S_{\sigma(2)}^i\} \cup ... \cup \{S_k^R \cap S_{\sigma(k)}^i\} | max_{\sigma \in \Gamma} \sum_{j=1}^{k} |S_j^R \cap S_{\sigma(j)}^i|\},$$

where $\Gamma$ is the set of all the possible permutations of $\{1, 2, ..., k\}$, and $|S_j^R \cap S_{\sigma(j)}^i|$ is the number of elements shared by two subsets of individuals $I^R$ and $I^i$. Then, the subset of vertices $B \subset V$ which are grouped together throughout each

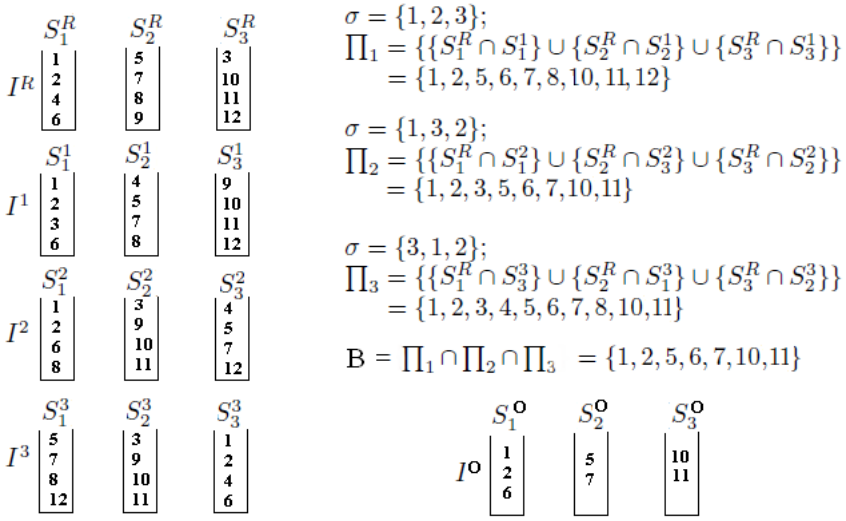


Fig. 1. Recombination with four individuals

individual of $P$ is $B = \prod_1 \cap \prod_2 \cap ... \cap \prod_{p-1}$.

Let $S_m$ be the subset of vertex $v$ in the reference individual $I^R$. If $v \in B$, then $v$ will automatically be placed to subset $S_m$ of the offspring individual $I^O$. Otherwise, it will be placed to subset $S_m$ of $I^O$ if $c/p - 1$ is greater than or equal to some random real number in the range of $[0, 1]$, where $c$ is the number of subsets of $\prod$ in which $v$ occurs.

If after this procedure, a vertex $v$ is left unassigned, it is placed to a random subset $S_r$ of $I^O$ such that $W(S_r \cup \{v\}) \leq W_{opt}$.

The complexity of the proposed recombination operator is $O(p * k * |V|)$.

An example of recombination with four individuals is given in Fig. 1. Note that in the new offspring individual $I^O$, vertices from $B$ are placed into the same subset as in $I^R$. The rest of the vertices in $I^O$, i.e. vertices $\{3, 4, 8, 9, 12\}$ will be assigned to the same subset as in $I^R$ with a certain probability as previously described.

### D. Local search improvement

To improve the newly generated offspring, we apply local search which is based on two neighborhood relations (call them $N_1$ and $N_2$) combined in a token-ring way, i.e. one neighborhood search is applied to the local optimum produced by the previous one. We also periodically apply a simple perturbation mechanism to bring some diversification into the search.

*1) Neighborhood relations:* Given a subset $S_i$, the basic idea of the neighborhood relations is to move a vertex $v$ to $S_i$ only if $v$ is a border vertex relative to $S_i$. $v \notin S_i$ is a border vertex relative to subset $S_i$ if and only if $v$ has at least one adjacent vertex in $S_i$. Note that in this way, the size of the neighborhoods is largely reduced, since the set of border vertices relative to $S_i$ is generally of small size. In addition, the search is concentrated only around these critical vertices.

The key concept related to the two neighborhoods is the *move gain*, which represents the change in the optimization objective. It expresses an estimate on how much a partition could be improved if a vertex $v$ is moved to another subset $S_n$. Given a vertex $v$ from subset $S_c$, the gain $g(v, n)$ can be computed for every other subset $S_n$, $n \neq c$. The selection of a vertex with the highest gain is achieved efficiently by using an adaptation of bucket sorting that was originally proposed in [3] for graph bisection.

Let $I = \{S_1, S_2, ..., S_k\}$ be a $k$-partition, $V(S_i)$ the set of border vertices relative to subset $S_i$, $S_{max} = \{S_i | max_{i \in \{1..k\}}\{W(S_i)\}\}$ the subset with the maximum vertex weight. The neighborhood relations $N_1$ and $N_2$ can be explained by the two following move operators.

> **Move 1**: Move one highest gain vertex $v_m$. Choose randomly a subset $S_m \in \{S_1, S_2, ..., S_k\} - \{S_{max}\}$. Then, select a *highest gain* vertex $v_m \in V(S_m)$ whose current subset is $S_c$, such that $S_c \in \{S \in I | W(S) > W(S_m)\}$. Move the selected vertex $v_m$ to subset $S_m$.
>
> **Move 2**: Move two highest gain vertices $v_m$ and $v_n$. Choose vertex $v_m$ and its new subset $S_m$ as in the first move operator. Choose randomly a new subset $S_n \in \{S_1, S_2, ..., S_k\} - \{S_{max}, S_m\}$. Then, select a highest gain vertex $v_n \in V(S_n)$ whose current subset is $S_c$, such that $S_c \in \{S \in I | S \neq S_n\}$. Move $v_m$ to $S_m$, and $v_n$ to $S_n$.

Let $V_{cand} \subset V(S_m)$ be the set of highest gain vertices which are considered for migration to subset $S_m$. Among the vertices from $V_{cand}$, we use two criteria to select a vertex which is moved to $S_m$. These two criteria are move frequency and vertex weight. We first give priority to a vertex which has been moved less often. Let $V_f \subset V_{cand}$ be the subset of vertices with the same move frequency. Then, we select vertex $v \in V_f$ which, when moved to subset $S_m$, minimizes the weight imbalance between $S_m$ and the subset $S_c$ from which $v$ is moved, i.e. $min_{v \in V_f}\{abs(W(S_m) + 2 * |v| - W(S_c))\}$ where $|v|$ is the weight of vertex $v$.

*2) Perturbation mechanism:* Since our local search procedure focuses its search only around border (critical) vertices, it can easily get trapped in a local optimum. Therefore, we periodically apply a simple perturbation which consists in moving a fixed number of vertices $\gamma$, including non border ones, in the following way.

Let $S_{max}$ be the subset of vertices with the maximum vertex weight, $S_{max} = max_{i\in\{1..k\}}\{W(S_i)\}$. Randomly select a subset $S_m \in \{S_1, S_2, ..., S_k\} - \{S_{max}\}$. Then, randomly choose a vertex $v_m$ whose current subset is $S_c$, such that $S_c \in \{S \in I | W(S) > W(S_m)\}$. Move the selected vertex $v_m$ to subset $S_m$. This operation is repeated $\gamma$ times (perturbation strength $\gamma$ is set in this paper to $2\%$ of the total number of vertices).

### E. Pool replacement strategy

When an offspring $I^0$ is obtained with the recombination operator, we improve it with the local search algorithm from Section IV-D, and then decide whether $I^0$ should be inserted into the population pool. To base this decision, our algorithm uses the ideas presented in [13] and [11].

Given a population $POP = \{I^1, I^2, ..., I^m\}$ of size $m$, and the distance $d_{i,j}$ (see Section IV-B) between any two individuals $I^i$ and $I^j$, $(i, j = 1...m$ and $i \neq j)$, the minimum distance between $I^i$ and any other individual in $POP$ is given by:

$$D_{i,POP} = min\{d_{i,j} | I^j \in POP, j \neq i\}$$

Offspring $I^0$ is inserted into $POP$ if it is of the best quality relative to the population, or if $D_{0,POP} > min(D_{i,POP})$, i.e. the minimum distance between $I^0$ and any other individual in the population is greater than the minimum distance between any two individuals in the population. This idea was originally proposed in [13], and has been shown to be very effective in ensuring the population diversity.

As proposed in [11], we take into consideration the goodness score for population when choosing an individual $I^i \in POP$, which is to be replaced by $I^0$. This score is based both on solution quality and minimum distance:

$$h_{i,POP} = f_1(I^i) + \beta / D_{i,POP},$$

where $f_1$ is the first evaluation function defined in Section IV-A and $\beta$ a parameter set to $\beta = 0.08 * |V|$.

## V. Experimental results

### A. Experimental protocol

Our partition algorithm is programmed in C++, and compiled with GNU gcc on a Xeon E5440 with 2.83 GHz and 8GB. We test our approach on the same set of benchmark graphs described in Section 2, which was also used in [14] for evaluating one of the currently most effective graph partitioning algorithms. As in [14], we report results with the number of partition subsets $k$ set to 4, 8, 16 and 32. To evaluate the performance of the proposed multilevel memetic algorithm we perform two comparisons. The first comparison is with two state-of-art graph partitioning packages (METIS [9] and CHACO [8]). In the second one, we compare the partitions generated by the proposed approach with the best partitions reported in the literature. In addition, we compare our partitions that are (usually) of perfect balance ($\varepsilon = 1.00$), with the imbalanced partitions obtained by two of the currently most powerful graph partitioning algorithms reported in [14] and [12]. It is very important to mention that allowing a larger degree of imbalance usually leads to partitions of better quality in terms of the number of cutting edges $|E_c|$.

The parameter settings used in both comparisons are reported in Table III.

### B. Comparison with state-of-art partitioning packages

For this comparison, we use the latest versions (METIS-4.0, CHACO-2.2) available at the time of writing. 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 execute our algorithm only once even though multiple runs would generate improved results.

Table IV reports the partition quality obtained by pMetis, CHACO, and our multilevel memetic approach (MMA). We also indicate the time (in seconds) required by MMA to reach the reported partitions. The last row with heading '*Total*' shows the number of times each approach produced the best partition.

From Table IV, we observe that the computation time of our approach is quite longer compared to those of the partitioning packages, which is often less than a second. However, in terms of partition quality, it performs far better than pMetic and CHACO.

As $k$ increases, the proposed approach and pMetis in some cases fail to generate partitions of perfect balance. Next to the imbalanced partitions, we indicate the degree of imbalance in parentheses. Our approach sometimes requires more computing time to establish good balance. On the other hand, CHACO always generates perfectly balanced partitions since it uses recursive bisection.

### C. Comparison with the best partitions and partitioning approaches reported in the literature

The best reported partitions generated with the proposed approach are obtained after 20 executions of our algorithm, within the time limit of 90 minutes for the largest instance (i.e. *brack2*). Before each recombination, the number of parent individuals is chosen randomly from a range [2,6]. In this section, we compare these partitions with the *best known* balanced partitions ($\epsilon = 1.00$) reported at the Graph Partitioning Archive, which are obtained by several different algorithms. Unfortunately, the conditions used to obtain these results, such as the number of algorithm executions or running time, are not reported. In addition, we compare our balanced partitions with the imbalanced ones generated with the two effective graph partitioning approaches proposed by Osipov and Sanders [12], and Soper et al. [14].

TABLE III
SETTINGS OF IMPORTANT PARAMETERS FOR THE FIRST COMPARISON FROM SECTION V-B AND THE SECOND COMPARISON FROM SECTION V-C.

| *Parameters* | *Description* | *Values for Comp. 1* | *Values for Comp. 2* |
|---|---|---|---|
| $k$ | number of partition subsets | $[4, 8, 16, 32]$ | $[4, 8, 16, 32]$ |
| $POP_s$ | size of population | 10 | 40 |
| $p$ | number of parents involved in recombination | 4 | $random(2, 6)$ |
| $\lambda$ | size of tournament pool | 6 | 6 |
| $\theta$ | number of recombination operations | 10 | 20 |
| $br$ | number of LS iter. before recombination | $\|V\|$ | $10 * \|V\|$ |
| $ar$ | number of LS iter. after recombination | $5 * \|V\|$ | $100 * \|V\|$ |
| $ct$ | coarsening threshold | 100 | 100 |
| $p_{str}$ | perturbation strength | $0.02 * \|V\|$ | $0.02 * \|V\|$ |
| $\gamma$ | non-improvement LS iter. before perturbation | $0.01 * \|V\|$ | $0.01 * \|V\|$ |

TABLE IV
COMPARISON BETWEEN THE STATE-OF-ART PACKAGES AND OUR PROPOSED MULTILEVEL MEMETIC APPROACH FOR CARDINAL NUMBERS OF 4, 8, 16 AND 32.

| | $k = 4$ | | | | $k = 8$ | | | |
|---|---|---|---|---|---|---|---|---|
| Graph | pMETIS | CHACO | MMA | Time(sec) | pMETIS | CHACO | MMA | Time(sec) |
| uk | 67 | 69 | **44** | 6.4 | **101** | 119 | 103 | 5.1 |
| add32 | 42 | 56 | **33** | 7.9 | 81 | 115 | **69** | 8.0 |
| crack | 382 | 445 | **376** | 15.5 | 773 | 777 | **703** | 15.1 |
| wing_nodal | 4000 | 4022 | **3706** | 14.7 | 6070 | 6147 | **5500** | 17.7 |
| vibrobox | 21471 | 21774 | **19314** | 48.5 | 28177 | 33362 | **25014** | 38.6 |
| 4elt | 406 | 433 | **339** | 24.7 | 635 | 688 | **573** | 25.3 |
| cti | 1113 | 1117 | **990** | 44.2 | 2110 | 2102 | **1961** | 29.0 |
| cs4 | 1154 | 1166 | **1025** | 51.8 | 1746 | 1844 | **1582** | 51.2 |
| bcsstk32 | 12205 | 15704 | **9935** | 186.5 | **23601** | 25719 | 23627 | 159.7 |
| t60k | 255 | 235 | **222** | 215.7 | 561 | 524 | **500** | 245.8 |
| wing | 2086 | 1982 | **1791** | 350.0 | 3205 | 3174 | **2706** | 322.8 |
| brack2 | **3250** | 3462 | 3427 | 502.1 | 7844 | 8026 | **7601** | 369.8 |
| Total | 1 | 0 | 11 | | 2 | 0 | 10 | |
| | $k = 16$ | | | | $k = 32$ | | | |
| Graph | pMETIS | CHACO | MMA | Time(sec) | pMETIS | CHACO | MMA | Time(sec) |
| uk | 189 | 211 | **178** | 5.2 | 316(1.01) | 343 | **313(1.03)** | 5.5 |
| add32 | **128** | 174 | 129 | 4.9 | **288(1.01)** | 303 | – | – |
| crack | 1255 | 1253 | **1126** | 13.4 | 1890 | 1962 | **1730** | 15.5 |
| wing_nodal | 9290 | 9273 | **8546** | 21.6 | 13237 | 13258 | **12089(1.01)** | 22.8 |
| vibrobox | 37441 | 43064 | **33645(1.02)** | 59.8 | 46112 | 51006 | **41276(1.02)** | 86.2 |
| 4elt | 1056 | 1083 | **983** | 23.9 | 1769 | 1766 | **1606** | 28.4 |
| cti | 3181 | 3083 | **2950** | 27.9 | 4605 | 4532 | **4354** | 29.4 |
| cs4 | 2538 | 2552 | **2234** | 37.4 | 3579 | 3588 | **3127** | 40.3 |
| bcsstk32 | 43371 | 47829 | **39200** | 218.6 | 70020 | 73377 | **62577** | 238.7 |
| t60k | 998 | 977 | **903** | 207.6 | 1613 | 1594 | **1475** | 194.1 |
| wing | 4666 | 4671 | **4215** | 251.2 | 6700 | 6843 | **6024** | 308.5 |
| brack2 | 12655 | 13404 | **12221** | 268.7 | 19786 | 20172 | **18425** | 305.2 |
| Total | 1 | 0 | 11 | | 1 | 0 | 11 | |

The recently proposed approach in [12] is a multilevel algorithm based on the idea to contract only a single edge on each level of the hierarchy, which results very few changes between two levels. In addition, it employs very effective data structures and fast local search improvement that can scale to large graph inputs. However, it cannot handle cases with the imposed perfect balance constraint. Therefore, we compare our balanced partitions with the best ones reported in [12] having 1% imbalance ($\epsilon = 1.01$), and which are obtained in a time limit of one hour.

The approach reported in [14] combines an evolutionary search approach with the JOSTLE multilevel procedure used as a black box. Although it generates high quality partitions, this approach requires very long run times (e.g. up to one week), since each run consists of 50,000 calls to JOSTLE. In [14], the authors report partitions with 3% of imbalance tolerance (i.e. $\epsilon = 1.03$). However, the exact running as well as the number of execution used to attain the results are not reported.

Table V shows the best know results reported at the Graph Partitioning Archive, the results generated with the $n$-Level graph partitioning algorithm from [12] (KaSPar), the results obtained with the multilevel evolutionary algorithm presented in [14] (ESA), and the results produced with our new multilevel memetic algorithm [1] (MMA). The partitions produced by KaSPar and ESA are with $\epsilon = 1.01$ and $\epsilon = 1.03$ imbalance respectively. For MMA, the reported partitions are perfectly balanced ($\epsilon = 1.00$), except for three cases where an imbalance of $\epsilon = 1.01$ is indicated next to the objective value. Colon '*Avg (Std)*' provides the average value and standard deviation of the results generated by our approach. The last row from Table V shows the number of times each colon '*Best*', '*KaSPar*', '*ESA*' report a partition that is better than our newly obtained partition.

From the given results, we observe that the proposed multilevel memetic algorithm improves on the best known balanced partition in almost each case. Moreover, our bal-

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

TABLE V
A COMPARISON WITH THE BEST KNOWN BALANCED PARTITIONS AND TWO OF THE MOST EFFECTIVE GRAPH PARTITIONING ALGORITHMS REPORTED IN THE LITERATURE, FOR $k$ EQUAL TO 4, 8, 16 AND 32. THE BEST RESULTS ARE GIVEN IN BOLD.

| | $k=4$ | | | | | $k=8$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Graph | Best(1.00) | KaSPar(1.01) | ESA(1.03) | MMA(1.00) | Avg(Std) | Best(1.00) | KaSPar(1.01) | ESA(1.03) | MMA(1.00) | Avg (Std) |
| uk | 43 | **41** | **41** | **41** | 43.8 (0.87) | 89 | 92 | **83** | 84 | 86.8 (1.98) |
| add32 | 34 | **33** | **33** | **33** | 34.8 (3.71) | 75 | **66** | 69 | **66** | 69 (3.52) |
| crack | 368 | 370 | **361** | 366 | 368.3 (2.76) | 687 | 696 | **676** | 679 | 692.8 (7.53) |
| wing-nodal | 3581 | 3609 | 3590 | **3576** | 3613.6 (29.7) | 5443 | 5574 | **5424** | 5438 | 5457.8 (21.9) |
| vibrobox | 19245 | 19267 | 19245 | **19119** | 19579.7 (342.54) | 24715 | 25190 | 24874 | **24557** | 24798.4 (126.13) |
| 4elt | 326 | **325** | 320 | 326 | 333.8 (10.44) | 548 | 561 | **532** | 545 | 550.9 (7.61) |
| cti | 963 | 950 | **927** | 954 | 975.3 (21.86) | 1812 | 1815 | **1716** | 1790 | 1841.0 (26.08) |
| cs4 | 964 | 970 | **936** | 961 | 982.9 (10.42) | 1496 | 1520 | 1488 | **1465** | 1483.9 (11.26) |
| bcsstk32 | 9492 | **9247** | 9992 | 9318 | 9383.4 (53.0) | 22757 | **20855** | 21307 | 21589 | 22123.7 (508.3) |
| t60k | **213** | **213** | 215 | 216 | 220.9 (2.59) | 476 | 470 | **469** | 474 | 483.3 (7.04) |
| wing | 1666 | 1683 | 1672 | **1664** | 1703.0 (23.79) | 2589 | 2616 | **2551** | 2553 | 2593.8 (31.21) |
| brack2 | 3090 | **3027** | 2873 | 3129 | 3203.9 (43.45) | 7169 | 7144 | **7114** | 7225 | 7389.9 (95.31) |
| Total | 2 | 5 | 5 | | | 1 | 3 | 9 | | |
| | $k=16$ | | | | | $k=32$ | | | | |
| Graph | Best(1.00) | KaSPar(1.01) | ESA(1.03) | MMA(1.00) | Avg(Std) | Best(1.00) | KaSPar(1.01) | ESA(1.03) | MMA(1.00) | Avg (Std) |
| uk | 159 | 179 | 157 | **150** | 156.0 (3.19) | **258** | – | 266 | 266(1.01) | 273.0 (5.07) |
| add32 | 121 | **117** | **117** | **117** | 120.8 (5.53) | **212**(1.01) | **212** | 212 | **212** (1.01) | 215.6 (6.77) |
| crack | 1108 | 1183 | **1083** | 1090 | 1108.7 (8.17) | 1728 | – | 1699 | **1687** | 1704 (9.51) |
| wing_nodal | 8422 | 8624 | 8361 | **8348** | 8399.7 (39.8) | 12080 | – | 12024 | **11863** | 11879.3 (22.6) |
| vibrobox | 32167(1.01) | 35514 | 33676 | **32154**(1.01) | 32587.6 (322.93) | 42187 | 46331 | 43091 | **40085** | 41146.5 (403.56) |
| 4elt | 956 | 1009 | **916** | 939 | 950.8 (6.42) | 1592 | – | **1540** | 1562 | 1572.4 (7.06) |
| cti | 2909 | 3056 | **2859** | 2878 | 2922.4 (32.02) | 4288 | 5044 | 4438 | **4142** | 4191.7 (48.19) |
| cs4 | 2206 | 2285 | 2204 | **2113** | 2129.3 (10.47) | 3110 | 3521 | 3117 | **2970** | 3001.0 (19.26) |
| bcsstk32 | 38711 | 37372 | 38927 | **36518** | 37225.7 (506.1) | 63856 | 72471 | 64433 | **61431** | 62263.8 (555.6) |
| t60k | **866** | **866** | 886 | 868 | 884.5 (7.98) | 1440 | 1493 | 1478 | **1421** | 1448.2 (12.02) |
| wing | 4198 | 4147 | 5015 | **3945** | 4014.0 (39.45) | 6009 | 6271 | 6039 | **5690** | 5740.6 (26.23) |
| brack2 | 12323 | 11969 | 12009 | **11760** | 11958.9 (126.35) | 18229 | 18496 | 17952 | **17491** | 17780.7 (177.65) |
| Total | 1 | 1 | 3 | | | 1 | 0 | 1 | | |

anced partitions are generally better than the imbalanced ones reported in [12]. As it can be seen from Table V, $n$-Level graph partitioning algorithm is sometimes unable to obtain a feasible solutions for larger values of $k$ (indicated by '-'). When compared to the imbalanced partitions generated with the evolutionary approach [14], our partitions are still better in many cases, especially for larger values of $k$.

All these results suggest that the newly proposed multilevel memetic algorithm is extremely effective when it comes to generating high quality balanced partitions.

## VI. CONCLUSION

We proposed an efficient multilevel algorithm with memetic refinement for the balanced graph partitioning problem. It integrates a new recombination operator based on the backbone with respect to a given number of parent individuals, which is motivated by the fact that there is always a high number of vertices that are grouped together throughout each high quality solution. When an offspring is created with the proposed operator, we refine it using local search with periodic perturbation that concentrates only around critical (border) vertices. Finally, to maintain a healthy population diversity, the algorithm integrates a highly effective pool replacement strategy, that takes into consideration both the solution quality and the distance between solutions. Although it requires longer computing time compared to some current state-of-art approaches, our algorithm is highly effective in terms of solution quality. Indeed, experiments on a collection of unweighted graph instances show that this multilevel memetic algorithm easily reaches or improves on almost all the best known balanced partitions reported so far. Moreover, it would be interesting to verify the efficiency of the proposed approach on a set of weighted benchmark graphs.

## ACKNOWLEDGMENT

This work was partialy supported by Angers Loire Métropole and the Region of Pays de la Loire within the RADAPOP and LigeRO Projects.

## REFERENCES

[1] C.J. Alpert and A.B. Kahng. Recent Directions in Netlist Partitioning: A Survey. *Integration, the VLSI Journal* 19(1–2): 1-81, 1995.

[2] T.N. Bui and C. Jones. A Heuristic for Reducing Fill-in in Sparse Matrix Factorization. In R.F Sincovec et al. (Eds), *Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 445–452, 1993.

[3] C. Fiduccia and R. Mattheyses. A Linear-time Heuristics for Improving Network Partitions. *Proceedings of the 19th Design Automation Conference*, 171–185, 1982.

[4] B. Freisleben and P. Merz. New Genetic Local Search Operators for the Traveling Salesman Problem. Proceedings of PPSN-96, *Lecture Notes in Computer Science 1141*, 890–899, Springer, 1996.

[5] M. Garey and D. Johnson. Computers & Intractability: A Guide to the Theory of NP-Completeness. *In W.H. Freeman and Company*, 1979.

[6] D. Gusfield. Partition-Distance: A Problem and Class of Perfect Graphs Arising in Clustering. *Information Processing Letters* 82(3): 159–164, 2002.

[7] B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. *Proceedings of Supercomputing'95*, San Diego, ACM Press, New York, 1995.

[8] B. Hendrickson and R. Leland. The Chaco User's Guide. Sandia National Laboratories, 2.0 edn, 2005.

[9] G. Karypis and V. Kumar. MeTiS 4.0: Unstructured Graphs Partitioning and Sparse Matrix Ordering System. *Technical Report*, Department of Computer Science, University of Minnesota, 1998.

[10] G. Karypis and V. Kumar. Multilevel k-Way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48(1): 96–129, 1998.

[11] Z. Lü and J.K. Hao. A Memetic Algorithm for Graph Coloring. *European Journal of Operational Research* 203(1): 241–250, 2010.

[12] V. Osipov and P. Sanders. n-Level Graph Partitioning. *Computing Research Repository*, abs/1004.4024, 2010.

[13] C.D. Porumbel, J.K. Hao, and P. Kuntz. An Evolutionaty Approach with Diversity Guarantee and Well-informed Grouping Recombination for Graph Coloring. *Computers and Operations Research* 37(10): 1822-1832, 2010.

[14] A.J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph-partitioning. *Journal of Combinatorial Optimization* 29: 225–241, 2004.