A Multilevel Memetic Approach for Improving Graph k-Partitions

Una Benlic and Jin-Kao Hao

To appear in *IEEE Transactions on Evolutionary Computation*
15 March 2011

***Abstract*—Graph partitioning is one of the most studied NP-complete problems. Given a graph $G = (V, E)$, the task is to partition the vertex set V into k disjoint subsets of about the same size, such that the number of edges with endpoints in different subsets is minimized. In this work, we present a highly effective multilevel memetic algorithm, which integrates a new multi-parent crossover operator and a powerful perturbation-based tabu search algorithm. The proposed crossover operator tends to preserve the backbone with respect to a certain number of parent individuals, i.e. the grouping of vertices which is common to all parent individuals. Extensive experimental studies on numerous benchmark instances from the Graph Partitioning Archive show that the proposed approach, within a time limit ranging from several minutes to several hours, performs far better than any of the existing graph partitioning algorithm in terms of solution quality.**

***Index Terms*—Graph partitioning, multi-parent crossover, tabu search, backbone, landscape analysis.**

I. Introduction

Graph partitioning is one of the fundamental combinatorial optimization problems which is notable for its applicability to a wide range of domains, such as VLSI design [1], [39], data mining [47], image segmentation [37], etc. It is well known that the general graph partitioning problem is NP-complete [15], so approximate approaches are very useful to address this problem.

Evolutionary algorithms are among the most popular approaches for the graph partitioning problem. Some representative examples include: Mansour and Fox [29], who enforce the equi-partition constraint with a penalty term; Talbi and Bessiere [42], whose genetic algorithm is based on a cellular population structure; Bui and Moon [10], who additionally employ a preprocessing phase scheme that improves the space searching capability of the genetic algorithm; Gil et al. [16], who use the direct encoding for circuit partitioning; Kang and Moon [23], and Kim and Moon [27], who perform extensive experiments on graphs with up to 5,000 vertices that show an improvement over the local optimization approaches. In [43], von Laszewski employs a ‘structural genetic operator’ which copies subsets of vertices to the offspring. The current most effective population-based approach is the one reported by Soper et al. [38], which employs a multilevel heuristic algorithm to provide an effective crossover. Although this approach requires considerable computing time (up to one week), it achieves partitions significantly better than those generated by the state-of-art graph partitioning packages.

Moreover, a great number of well-known graph partitioning approaches are based on other popular metaheuristics including Tabu Search [12], [36], [4], Simmulated Annealing [21], Neural Networks [2], Swarm Intelligence [41], etc.

To handle very large graphs, the so-called multilevel paradigm has shown to be very effective for partitioning graphs [3], [19], [25], [44], [31]. The basic idea of multilevel approaches is to first coarsen the original graph G down to a certain number of vertices, generate a partition of this much smaller graph, and then project this partition back towards G by successively refining the partition.

In this paper, we introduce a new multilevel memetic algorithm which combines a dedicated multi-parent crossover operator based on the notion of backbone and a perturbation-based tabu search algorithm. This work extends thus a preliminary memetic algorithm presented in [5], where a different crossover operator and a hill-climbing based local search algorithm are used. Compared to this previous work, the new algorithm presented in this paper ensures better exploration with the new multi-parent crossover operator, and better exploitation provided by an effective tabu search algorithm. Furthermore, this paper additionally includes: (a) more extensive experimental evaluations on a set of benchmark instances from the Graph Partitioning Archive; (b) a detailed analysis on several key issues such as the distribution of local optima and the backbone size; (c) a comparison of the proposed crossover operator with the traditional uniform crossover, and analysis on the impact of perturbation within the proposed crossover operator; and (d) an analysis on the impact of the employed local search mechanism on the overall performance of our memetic algorithm.

The paper is organized as follows. In the next section, we recall the definition of the graph partitioning problem and some basic notations used. In Section III, we describe the multilevel paradigm, as well as the general scheme of the proposed multilevel approach. In Section IV, we present the memetic algorithm, which is the partition refinement mechanism of the multilevel approach. In Section V, we provide computational results of extensive experiments on benchmark instances from the Graph Partitioning Archive. In Sections VI, we show a landscape and backbone analysis, and based on the observations made, provide a motivation for our proposed multi-parent crossover operator. In Section VII, we compare the proposed

Una Benlic and Jin-Kao Hao (Corresponding author) are with the LERIA, University of Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France (e-mail: {benlic,hao}@info.univ-angers.fr)

crossover with a conventional uniform crossover operator, and study the impact of perturbation strength within our multi-parent crossover operator. In Section VIII, we analyze the impact of local search on the overall algorithm performance before concluding in Section IX.

II. 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-partition of G can be defined as a mapping (partition function) $\pi : V \rightarrow \{1, 2, ..., k\}$ that distributes the vertices of V among k disjoint subsets $S_1 \cup S_2 \cup ... \cup S_k = V$.

Let $\{S_1, S_2, ..., S_k\}$ be a partition of V obtained by π, E^c the set of all the cutting edges of G induced by π, i.e. $E^c = \{\{x, y\} \in E \mid x \in S_i \text{ and } y \in S_j \text{ and } i \neq j\ \}$, and let φ be the set of all the partition functions of G. The graph k-partitioning problem consists in determining $\pi^* \in \varphi$ such that the partition $\{S_1, S_2, ..., S_k\}$ given by π^* minimizes the number of cutting edges in E^c while ensuring that each S_i, $i \in \{1, 2, ..., k\}$ is of roughly equal size.

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 III-B, the multi-level approach generates intermediate (coarsened) graphs with weighted edges and vertices. It is then useful to define the notion of edge and vertex weight.

Let $|v|$ denote the weight of a vertex v in a coarsened graph, which corresponds to the number of aggregated vertices of the initial graph. Then, the weight $W(S_i)$ of a vertex 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 weight of a set of edges in the coarsened graph can similarly be defined.

In this paper, we are essentially interested in finding almost evenly balanced partitions. The notion of balance is defined as follows. Let $W_{opt} = \lceil |V|/k \rceil$ be the optimal subset weight, where $\lceil x \rceil$ represents the first integer $\geq x$, then the quantity $\varepsilon = max_{i \in \{1..k\}} W(S_i)/W_{opt}$ defines the degree of imbalance among the k subsets of a partition $\{S_1, S_2, ..., S_k\}$. $\varepsilon = 1$ means that the partition is perfectly balanced while $\varepsilon > 1$ indicates an imbalanced partition with larger ε corresponding to larger imbalance.

The optimization objective f of our graph partitioning algorithm is to find a k-partition with the smallest number of edge cuts in E^c, such that each partition subset is of almost equal size ($\epsilon = 1.00$).

III. Multilevel memetic algorithm for graph partitioning

A. General procedure

Our multilevel memetic approach follows the general multilevel paradigm [9], [3], [19], [45]. Graph G is first coarsened down to a certain number of vertices (coarsening phase), an initial partition of this much smaller graph is generated (initial partitioning phase), and then this partition is projected back towards the original graph (uncoarsening phase) followed by partition refinement.

The proposed multilevel approach, which employs memetic partition refinement at each uncoarsening step, is presented in Algorithm 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 k partition of G_0

```
i := 0
while |V_i| > coarsening_threshold do
    G_{i+1} = Coarsen(G_i) /*Section III-B*/
    i := i + 1
end while
POP_i = Initial_Partition(G_i) /*Section III-C*/
POP_i = Short_Tabu_Search(POP_i) /*Section IV-C*/
POP_i = Memetic_Refinement(POP_i) /*Section IV*/
while i > 0 do
    i := i − 1
    POP_i = Project(POP_{i+1}, G_i) /*Section III-D*/
    POP_i = Short_Tabu_Search(POP_i)
    POP_i = Memetic_Refinement(POP_i)
end while
```

B. Coarsening phase

Let $G_0 = (V_0, E_0)$ be the initial graph. Creating a coarser graph $G_{i+1} = (V_{i+1}, E_{i+1})$ from $G_i = (V_i, E_i)$ consists in finding an independent subset of edges (matching) $\Gamma \subset E_i$, and then collapsing the two vertices of each edge in Γ to form a new vertex in V_{i+1}. Any vertex that is not part of Γ is simply copied over to G_{i+1} (see Fig. 1 for an illustrative example).

When two vertices $v_1, v_2 \in V_i$ are collapsed to form a new vertex $v_a \in V_{i+1}$, the weight of the resulting vertex v_a is set equal to the sum of weights of vertices v_1 and v_2. Therefore, the weight of a vertex of a coarsened graph equals the number of aggregated vertices of the initial graph G_0.

Similarly, let $v_a, v_b \in V_{i+1}$ be two vertices formed by collapsing $\{v_1, v_2\} \in \Gamma$ and $\{v_3, v_4\} \in \Gamma$. All the edges incident to $\{v_1, v_2\}$ and $\{v_3, v_4\}$ are merged to form a new edge $\{v_a, v_b\} \in E_{i+1}$ with a weight that is set equal to the sum of weights of edges incident to $\{v_1, v_2\}$ and $\{v_3, v_4\}$.

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 maximal matching of the graph [32]. 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 an approximate maximal matching using a fast heuristic called heavy-edge matching (HEM), which has $O(|E|)$ time complexity [24]. 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 maximal among all the edges incident to v. An example of vertex and edge aggregation with HEM of an initial graph with seven vertices is provided in Figure 1.

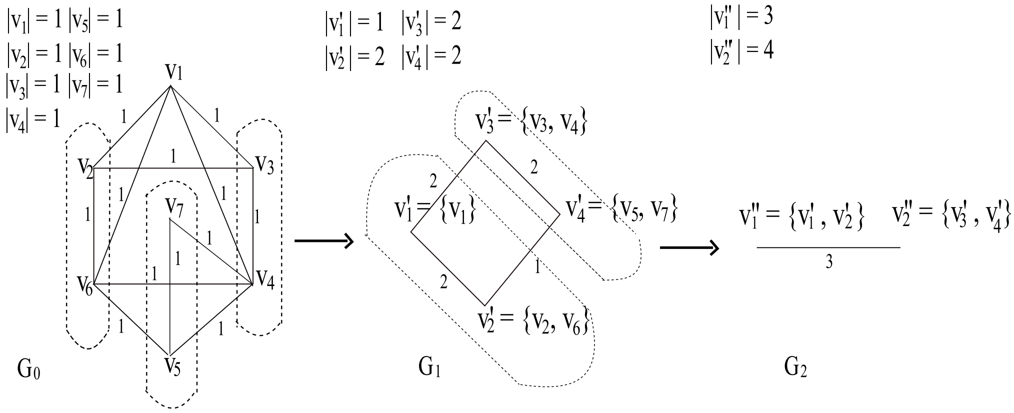


Fig. 1. An example of coarsening with HEM of an initial unweighted graph G_0 with seven vertices. The weight of a vertex $|v|$ of a coarsened graph G_i equals the number of aggregated vertices of the initial graph G_0. The weight of the resulting edge, which is incident to the collapsed vertex $v_c = \{u, v\}$ is set equal to the sum of weights of all the edges incident to u and v minus the weight of edge $\{u, v\}$.

C. Initial partition and its refinement

To create each individual of the initial population in the second phase (line 6 of Alg. 1), we first assign randomly the vertices of the coarsest graph $G_m = (V_m, E_m)$ to subsets $S_i \in \{S_1, S_2, ..S_k\}$, such that each subset is as evenly balanced as possible, i.e. each S_i, $i \in \{1, ..., k\}$ has a similar weight $W(S_i)$. Afterwards, we apply a short run of the perturbation-based tabu search, previously presented in [6], to improve all individuals of this initial population (see Section IV-C), followed by the memetic refinement which is described in Section IV (lines 7–8 of alg. 1).

This refinement step is essential for our approach to improve progressively the quality of partitions. It should be noted that for certain graphs, it may be impossible to obtain a perfectly balanced initial partition, since weights of vertices in the coarsest graph are often greatly inhomogeneous. This imbalance is gradually reduced throughout each uncoarsening step, and (usually) completely eliminated by the end of the algorithm execution.

D. 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 same 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, 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 the perturbation-based tabu search to improve all the individuals of the population, which is immediately followed by the memetic refinement (lines 12–13 of alg. 1). Experiments show that the local optimization applied on the initial population before memetic refinement (lines 7 and 12 of Alg. 1) influences favorably the final result, though this influence is not very important.

As the uncoarsening-refining process proceeds, the partition quality of a 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.

IV. The memetic refinement

The general idea behind memetic approaches is to combine advantages of both crossover that discovers unexplored promising regions of the search space, and local search that finds good solutions by concentrating the search around these regions. Given an initial population which consists of locally optimal solutions, a memetic approach generates a new set of improved local optima by applying a crossover operator and/or mutation to the population, followed by local refinement. The success of this method depends critically on the crossover operator that discovers new promising regions of the search space by performing ‘jumps’ from one local optimum to another. These jumps need to be far enough to escape from the basin of attraction of the current local optimum, but still not too far to degenerate into a simple random search algorithm. Furthermore, in order to perform a directed ‘jump’, a crossover operator should be able to recognize what elements must be preserved through the recombination and what elements can be perturbed.

Motivated by the observation that high quality partitions share many groupings of vertices (backbone, see below), we propose in this work an original backbone-based multi-parent crossover (BBC) which preserves the backbone with respect to a certain number of parent individuals. After the offspring has been generated by the proposed crossover, it is refined with a perturbation-based tabu search algorithm. Finally, we apply a replacement strategy, which takes into consideration both the partition quality and the distance between individuals in the population.

The general architecture of our memetic approach is described in Algorithm IV. The main components are detailed in the following subsections.

A. Encoding and fitness function

Given a graph $G_i = (V_i, E_i)$ at level i and an integer k, an individual I corresponds to a partition of V_i into k disjoint

Algorithm 2 Memetic refinement of our multilevel approach

Require: Population POP_i at graph level i
Ensure: Refined population POP_i

```
1:  I* ← Best(POP_i) /* the best individual found so far */
2:  for n := 1 to number of crossovers θ do
3:      Select p (p ≥ 2) individuals {I^1, ..., I^p} with the
        tournament selection strategy
4:      I^0 ← BBC(I^1, ..., I^p) /* Section IV-B */
5:      I^0 ← Tabu_Search(I^0) /* Section IV-C */
6:      if (f(I^0) < f(I*)) then
7:          I* ← I^0 /* update best individual found so far */
8:      end if
9:      POP_i ← Pool_Updating(I^0, POP_i) /* Section IV-D2
        */
10: end for
```

groups or subsets $I = \{S_1, ..., S_k\}$, such that each S_j, $j \in \{1, ..., k\}$ is composed of vertices that are assigned to the j^{th} subset.

The optimization objective of our k-partitioning problem is to minimize the cutting edges in E^c (see Section II), while maintaining the best possible balance between partition subsets.

The fitness function $f(I)$ of our memetic algorithm is directly related to the optimization objective and sums up the cutting edge weights of a k-partition (individual) $I = \{S_1, ..., S_k\}$. More formally,

$$f(I) = \sum_{\{u,v\} \in E_i} \varrho_{u,v}(I) \tag{1}$$

$$\varrho_{u,v}(I) = \begin{cases} w_{u,v} & \text{if } u \in S_x \text{ and } v \in S_y \ (x \neq y); \\ 0, & \text{otherwise.} \end{cases}$$

where $w_{u,v}$ represents the weight of edge $\{u, v\} \in E_i$, i.e. the number of unit cost edges of the original graph $G_0 = (V_0, E_0)$ that are aggregated within $\{u, v\} \in E_i$ during the coarsening phase.

Then, individual I^A is considered better than individual I^B only if $f(I^A) < f(I^B)$ (lines 6-8 of algo. 2).

Since our goal is to find perfectly balanced partitions ($\varepsilon = 1.00$), the partition balance is imposed as a constraint rather than an objective. However, as mentioned earlier, it is sometimes impossible to establish perfect balance in coarsened graphs since vertex weights can be extremely inhomogeneous. It is during the partition refinement of levels which are closer to the original graph that the balance condition is (usually) completely satisfied. More precisely, the tabu search procedure of our memetic algorithm employs two move operators that take care of partition imbalance by transferring vertices to subsets of smaller weight (see Section IV-C). In addition, the proposed backbone-based crossover operator insures that the balance is not degraded during the crossover process (see Section IV-B).

B. Backbone and crossover

1) Notion of backbone: Our backbone-based multi-parent crossover described in this section tries to preserve the backbone with respect to a number of parent individuals while redistributing with a certain probability vertices that do not belong to the backbone.

For optimization problems, the term *backbone* is usually used to define a set of variables B having the same value assignment throughout all the global optima, while the *backbone size* corresponds to the number of elements in B. The similar idea has been used in several contexts [13], [26], [46], [48].

For our graph partition problem, the notion of backbone can be defined as follows.

Definition 1 (Backbone): Let G be a graph, Ω the set of all optimal k-partitions of G. The backbone B of G is a set of k subsets of vertices $\{B_1, ..., B_k\}$ such that each B_i, $i \in \{1, ..., k\}$ is the subset of vertices that are grouped together throughout all the optima of Ω.

Definition 2 (Backbone size): Given a backbone $B = \{B_1, ..., B_k\}$, its size $|B|$ equals $|B_1 \cup ... \cup B_k|$.

Such a definition cannot be applied in practice given that the optimal solutions are unknown (our goal is to find such a solution). Therefore, in this paper, we use a relaxed definition of backbone by considering a set of locally optimal (high quality) solutions. Therefore, if a set of vertices are shared through the set of selected k-partitions, these vertices are considered to have a high chance to be part of the backbone.

2) The backbone-based multi-parent crossover operator (BBC): Given the set $P = \{I^1, ..., I^p\}$ of p parent individuals, BBC constructs the offspring $I^0 = \{S_1^0, ..., S_k^0\}$ in k passes (one for each subset of the partition). In each pass μ it performs the following steps:

1) Select a subset S_j^i of I^i such that the weight $W(S_j^i)$ is maximal across the subsets $j \in \{1..k\}$ of each individual $I^i \in P$, i.e. $max_{i \in \{1..p\}, j \in \{1..k\}}\{W(S_j^i)\}$, with the constraint that at most $\lceil k/p \rceil$ subsets can be chosen from each individual $I^i \in P$ (line 5 of alg. 3).
2) Given I^i and S_j^i determined in Step 1, for *each* individual $I^t \in P$ $(t \neq i)$, let $\prod_t$ contain the largest number of vertices that are shared by the subset S_j^i of I^i and a subset S_η^t of I^t, i.e. $\prod_t = \{S_j^i \cap S_\eta^t | max_{\eta \in \{1..k\}} |S_j^i \cap S_\eta^t|\}$. Then, $\prod = \{\prod_1, .., \prod_{p-1}\}$ forms a set of these vertex subsets (lines 6–9 of alg. 3).
3) Set $S_\mu^0 = \prod_1 \cap \prod_2 \cap ... \cap \prod_{p-1}$. S_μ^0 is the largest subset of vertices that are shared by all the parent individuals. For each vertex $v \in S_j^i$ and $v \neq S_\mu^0$, v is assigned to subset S_μ^0 of I^0 if $c(v)/p - 1$ is greater than or equal to some random real number in the range $[0, 1]$, where $c(v)$ is the number of subsets of $\prod$ in which v occurs (lines 11–15 of alg. 3).
4) When a vertex v is assigned to subset S_μ^0 of I^0 in the μ^{th} pass, v is removed from all the parent individual subsets in which it occurs, and the weights of these subsets are adjusted accordingly (lines 17–18 of alg. 3).

After the previous four steps, the last step handles the unassigned vertices. Any vertex v missing from I^0 is placed at random to a subset S_r of I^0 such that $W(S_r \cup \{v\}) \leq W_{opt}$ (lines 21–26 of alg. 3), where W_{opt} is defined in Section II. This step introduces a degree of diversification in the crossover

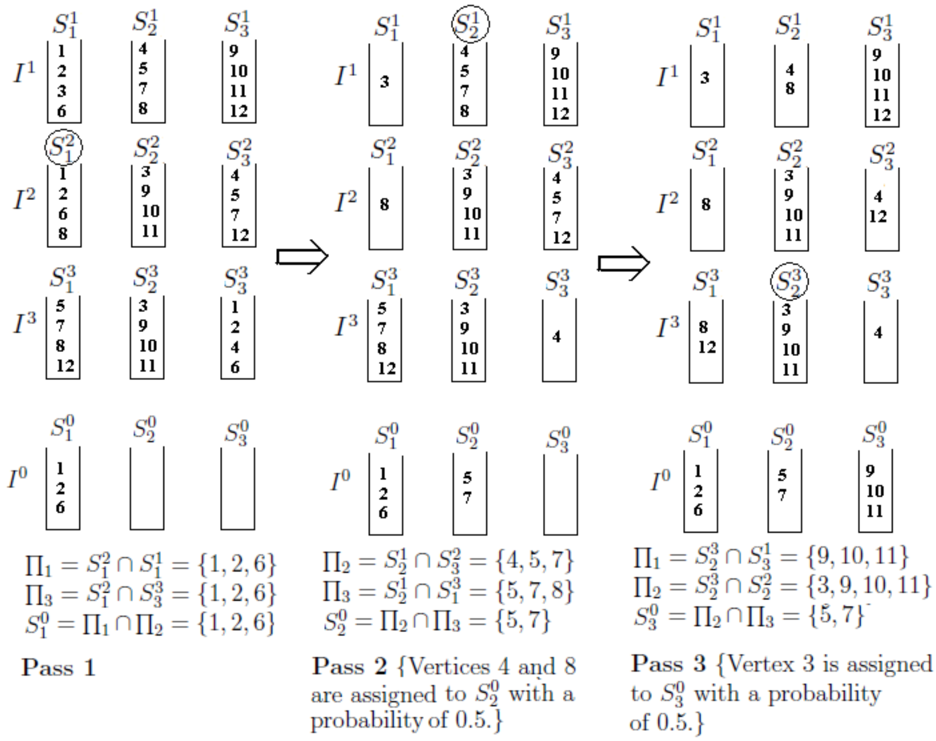


Fig. 2. An illustration of the BBC crossover with three parents. A circled subset of a parent corresponds to the subset chosen in the μ^{th} pass, i.e. the subset of maximal weight across all the parent individuals with the constraint that at most $\lceil k/p \rceil$ subsets can be chosen from each individual.

process.

Notice that the proposed BBC operator never degrades the balance with respect to the set of parent individuals P, since given a subset S_j^i of individual I^i which is chosen in the μ^{th} pass (see line 5 of alg. 3), at most $|S_j^i|$ vertices can be transmitted to the subset S_μ^0 of offspring I^0. In addition, an unassigned vertex v in I^0 is assigned to a subset S_r^0 only if adding v to S_r^0 does not exceed the expected optimal subset weight W_{opt}.

The complexity of the proposed crossover is $O(p * k * |V_i|)$, where $|V_i|$ is the number of vertices in graph $G_i(V_i, E_i)$.

To determine the subset $P \subset POP$ of p parent individuals, we employ the tournament selection strategy. Let λ be the size of the tournament pool. We select each individual $I^i \in P$ in the following way: randomly choose λ individuals from POP; among the λ chosen individuals, place the best one into P if it is not already present in P.

An example of this crossover with three parent individuals $(p = 3)$ for $k = 3$ is provided in Figure 2.

C. *Perturbation-based Iterated Tabu Search improvement*

To improve the newly generated offspring, we apply an iterated tabu search algorithm [6] whose basic components are briefly described in this section.

Basically, the TS algorithm uses two neighborhood relations (call them N_1 and N_2) which are explored in a token-ring way. That is, we repeatedly apply one neighborhood search to the best local optimum produced by the other neighborhood. The algorithm incorporates as well a perturbation mechanism in order to bring diversification into the search.

1) Neighborhood relations: Given a subset S_i of a k-partition $I = \{S_1, S_2, ..., S_k\}$, the basic idea of the neighborhood relations is to move a vertex v from another subset to S_i. Such a move is constrained such that v must be a border vertex relative to S_i, i.e. $v \notin S_i$ has at least one adjacent vertex in S_i. Note that in this way, the size of the neighborhoods is limited, since the set of border vertices relative to S_i is generally of small size. In addition, such a neighborhood allows the search to concentrate 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 the vertex with the highest gain, as well as the updates needed after each move, are achieved efficiently by using an adaptation of bucket sorting [6] that was originally proposed in [14] 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, and $S_{max} = \{S_i | max_{i \in \{1..k\}}\{W(S_i)\}\}$ the subset with the maximum vertex weight. The neighborhood relations N_1 and N_2 can be explained by the two move operators given below.

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 the *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.

Algorithm 3 Backbone-based multi-parent crossover (BBC)

Require: Set $P = \{I^1, ..., I^p\}$ of p parent individuals
Ensure: An offspring $I^0 = \{S_1^0, ..., S_k^0\}$
Initialize offspring $I^0 = \phi$
Set for each $I^e \in P$ the number of times it has been selected: $q(I^e) = 0$
Determine subset weights $W(S_b^e)$ of each $I^e \in P$, $\forall b \in \{1..k\}$
for $\mu := 1$ to k **do**
  /* **Step 1** */
  Select an individual I^i and its subset S_j^i such that: $max_{i\in\{1..p\},j\in\{1..k\}}\{W(S_j^i)\}$ and $q(I^i) \leq \lceil k/p \rceil$
  /* **Step 2**: Determine $\prod = \{\prod_1, .., \prod_{p-1}\}$ */
  for each $I^t \neq I^i$, $I^t \in P$ **do**
    $\prod_t = \{\{S_j^i \cap S_\eta^t\} | max_{\eta\in\{1..k\}} |S_j^i \cap S_\eta^t|\}$
  end for
  $S_\mu^0 \leftarrow \prod_1 \cap \prod_2 \cap ... \cap \prod_{p-1}$
  /* **Step 3**: Assign vertices */
  Count the number of occurrences $c(v)$ of each vertex $v \in S_j^i$ in $\prod$
  for all vertices $v \in S_j^i$ **do**
    if $(v \notin S_\mu^0$ and $c(v)/p - 1 \geq random[0..1])$ **then**
      $S_\mu^0 = S_\mu^0 \cup \{v\}$
    end if
  end for
  $q(I^i) = q(I^i) + 1$
  /* **Step 4** */
  Remove all vertices $v \in S_\mu^0$ from each $I^e \in P$
  Update subset weights $W(S_b^e)$ of each $I^e \in P$, $\forall b \in \{1...k\}$
end for
/* **Step 5**: Handle unassigned vertices */
Compute subset weights $W(S_b^0)$ of I^0, $\forall b \in \{1..k\}$
for all vertices $v \in V$ **do**
  if $v \notin \cup_{i=1}^k S_i^0$ **then**
    Assign v to a random subset S_r^0, $r \in \{1..k\}$ such that $W(S_r^0 \cup \{v\}) \leq W_{opt}$
    Update subset weight $W(S_r^0)$
  end if
end for
return $I^0 = \{S_1^0, S_2^0, ..., S_k^0\}$

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 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.

It is important to note that these move operators progressively lead the search toward a balanced partition since they basically constraint (partially with Move 2) vertex migration from heavy weight subsets to light weight subsets. Indeed, with Move 1 and the first choice of Move 2, a vertex can never be moved to a subset of the highest weight. The second choice of Move 2 is allowed to bring some diversification into the search.

Let $V_{cand} \subset V(S_m)$ be the set of the highest gain vertices which are considered for migration to subset S_m. The selection of vertex v, which is moved to S_m, is based on several pieces of history information.

This selection strategy is first conditioned by the tabu status (see IV-C2). It also 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. 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.

2) 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 v is tuned adaptively according to the number of border vertices relative to S_c,

$$tt(S_c)(v) = |V(S_c)| * \alpha,$$

where $|V(S_c)|$ is the number of border vertices relative to S_c, and α a parameter that takes randomly a value in the range $[0.05, ..., 0.2]$.

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

Let S_{max} be the set 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 γ times (perturbation strength γ is set in this paper to 2% of the total number of vertices).

Making such moves introduces naturally more diversification into the search.

D. Population updating based on distance

After offspring I^0 has been obtained with the proposed multi-parent crossover operator, we improve it with the perturbation-based tabu search algorithm from Section IV-C, and then decide whether I^0 should be inserted into the population. To base this decision, our algorithm combines the ideas presented in [34] and [28] and considers both the solution quality and the distance between individuals in population. Therefore, we first formally define the notion of distance between two individuals before presenting the used replacement strategy.

1) 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 partition distance [17] (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 $max_{\sigma \in \Psi} \sum_{i=1}^{k} M_{i,\sigma(i)}$, where Ψ 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.

Algorithm 4 Pool updating strategy

Require: Population $POP = \{I^1, ..., I^m\}$ and offspring I^0
Ensure: Updated population $POP = \{I^1, ..., I^m\}$
1: Tentatively add I^0 to population: $POP' = POP \cup \{I^0\}$
2: **for** $i := 0$ to m **do**
3: Calculate the minimum distance $D_{i,POP'}$ between I^i and any individual in POP' according to Eq.(2)
4: Calculate the goodness score $H_{i,POP'}$ of I^i according to Eq.(3)
5: **end for**
6: Select individual I^w with the largest H score $max_{j \in \{0..m\}}\{H_{j,POP'}\}$
7: Determine the minimum distance between two individuals: $D_{min} = min_{j \in \{0..m\}}\{D_{j,POP'}\}$
8: **if** ($I^0 \neq I^w$) and ($D_{0,POP} > D_{min}$ or $f(I^{best}) > f(I^0)$)) **then**
9: Replace I^w with I^0: $POP = POP \cup \{I^0\} \setminus \{I^w\}$
10: **end if**

---

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

$$D_{i,POP} = min\{d_{i,j} | I^j \in POP, j \neq i\} \tag{2}$$

Offspring I^0 is then inserted into POP if it is of the best quality relative to the population, or if $D_{0,POP} > min_{i \in \{1..m\}}\{(D_{i,POP})\}$, 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 [34], and has shown to be very effective in ensuring the population diversity.

To determine the individual that is to be replaced by I^0, we adopt the strategy proposed in [28]. This strategy uses a quality-and-distance scoring function H to rank the individuals of the population.

$$H_{i,POP} = f(I^i) + \beta / D_{i,POP} \tag{3}$$

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

Our pool updating strategy consists thus of three phases: for each individual $I^i \in POP$, calculating $D_{i,POP}$ and the corresponding $H_{i,POP}$ score (lines 2–5 of alg. 4); identifying the minimum distance D_{min} between any two individuals as well as the worst individual I^w (lines 6–7 of alg. 4); and updating the pool (lines 8–10 of alg. 4). This pool updating strategy contributes to maintaining a healthy diversity of the population.

V. Experimental results

A. Benchmark instances

To evaluate the efficiency of our proposed memetic approach, we carry out extensive experiments on a set of graphs that are frequently used to assess graph partitioning algorithms. These benchmark graphs are samples of small to medium scale real-life problems arising in different applications. They can be downloaded from Walshaw's Graph Partitioning Archive at: http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/ in the same format as used by JOSTLE [44], METIS [25] and CHACO [18]. These graphs have unit vertex and edge weights. Table I shows the main characteristics of the graphs.

B. Experimental protocol

There is generally a trade-off between execution time and partition quality. The preference of time vs. quality is problem dependent. For instance, in the context of network layout or VLSI design, even a slight improvement of partition quality can be of significant importance. For these applications, it is worthwhile to employ a partition algorithm able to obtain excellent quality solutions even if the algorithm is computationally intensive. On the other hand, in other cases like sparse matrix-vector multiplication, a very fast algorithm is indispensable since the computing time required for the partitioning task has to be less than the time needed by a fast vector multiplication algorithm.

Our MMA algorithm is designed to produce excellent quality partitions with the possibility to be used to generate solutions of various qualities depending on the amount of computing time allowed. We thus report computational results of two experiments with short and long runs of MMA. For the first experiment, we parameterize our MMA such that each run lasts from one second to 15 minutes depending on the size of the graph (see Table II and Section V-C). The second experiment aims to assess our MMA approach with respect to the best partitions reported at the Graph Partitioning Archive. For this experiment, we use a set of parameter values that lengthens each run of the MMA algorithm (see Section V-D). The second experiment allows us to test the limit of MMA and to obtain the best results possible with more computing budgets. Given the stochastic nature of MMA, computational statistics are based on 20 or 30 independent runs of MMA on each graph.

The proposed multilevel memetic algorithm is programmed in C++, and compiled with GNU gcc on a Xeon E5440 with 2.83 GHz and 8GB. The parameter settings applied in both experiments are reported in Table II. We fix experimentally the number of parents p for BBC relative to k: $p = 3$ for $k = 16$; $p = 4$ for $k = 2, 32, 64$; $p = 5$ for $k = 4, 8$.

TABLE I
THE LIST OF BENCHMARK GRAPHS TOGETHER WITH THEIR CHARACTERISTICS

Grap h	Size $\|V\|$	$\|E\|$	Degree *Max*	*Min*	*Avg*	*Type*
add20	2395	7462	123	1	6.23	20-bit adder
data	2851	15093	17	3	10.59	3D FEM
3elt	4720	13722	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	291583	140	19	66.74	3D stiffness matrix
whitaker3	9800	28989	8	3	5.92	2D nodal graph
crack	10240	30380	9	3	5.93	2D nodal graph
wing-nodal	10937	75488	28	5	13.80	3D nodal graph
fe-4elt2	11143	32818	12	3	5.89	2D FEM
vibrobox	12328	165250	120	8	26.8	Sparse matrix
bcsstk29	13992	302748	70	4	43.27	3D stiffness matrix
4elt	15606	45878	10	3	5.88	2D nodal graph
fe-sphere	16386	49152	6	4	5.99	3D FEM
cti	16840	48232	6	3	5.73	3D semi-structured graph
memplus	17758	54196	573	1	6.10	Memory circuit
cs4	22499	43858	4	2	3.90	3D nodal graph
bcsstk30	28924	1007284	218	3	69.65	3D stiffness matrix
bcsstk31	35588	572914	188	1	32.197	3D stiffness matrix
fe-pwt	36519	144794	15	0	7.93	3D FEL
bcsstk32	44609	985046	215	1	44.1636	3D stiffness matrix
fe-body	45097	163734	28	0	7.26	3D FEM
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
finan512	74752	261120	54	2	6.99	stochastic programming matrix
fe-tooth	78136	452591	39	3	11.58	3D FEM
fe-rotor	99617	662431	125	5	13.30	3D FEM
598a	110971	741934	26	5	13.37	3D FEM
fe-ocean	143437	409593	6	1	5.71	3D dual graph

TABLE II
SETTINGS OF IMPORTANT PARAMETERS.

Parameters	*Description*	*Values for Comp. 1*	*Values for Comp. 2*
k	number of partition subsets	$[2, 4, 8, 16, 32, 64]$	$[2, 4, 8, 16, 32, 64]$
POP_s	size of population	10	30
p	number of parents involved in crossover	$[3, 4, 5]$	$[3, 4, 5]$
λ	size of tournament pool	6	6
θ	number of crossover operations	10	30
sr	number of TS iter. before crossover (line 7 of alg. 1)	$\|V\|$	$10 * \|V\|$
lr	number of TS iter. after crossover	$5 * \|V\|$	$100 * \|V\|$
ct	coarsening threshold	200	200
p_{str}	perturbation strength	$0.02 * \|V\|$	$0.02 * \|V\|$
γ	non-improvement TS iter. before perturbation	$0.01 * \|V\|$	$0.01 * \|V\|$

C. Computational results with short running time

In this section, we show computational results of the first experiment and compare our results with those of the latest versions of METIS (METIS-4.0) [25] and CHACO (CHACO-2.2) [18] 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. Notice however that the purpose of this experiment is not to show a rigorous comparison of MMA with METIS and CHACO, given that MMA is a computationally intensive stochastic algorithm while METIS and CHACO are based on very fast heuristics (order of second) whose computing time cannot be tuned. Instead, we want to assess whether MMA can obtain good partitions with a reduced running time (one second to 15 minutes). Only for this purpose, we use the results of METIS and CHACO as our references. We do not claim that MMA can be a substitute for the existing fast partition packages. Therefore, this comparison should be interpreted with caution.

The computational results of the first experiment are shown in Table III. Columns two and three report respectively the partition quality obtained by pMetis and CHACO, while columns MMA_B and MMA_{Av} provide respectively the result of the best and average partition obtained with MMA (based on 30 independent runs per graph). We indicate the MMA's average partition in bold if it is better than the partitions obtained by both pMetis and CHACO. The last column shows the average time (in seconds) needed by our approach to generate the reported partition.

From Table III, we observe that the best partitions obtained with our MMA approach within a time limit ranging from less than one second up to 15 minutes are of far better quality in almost every case. In addition, the average quality of partitions obtained with MMA are also generally better than those of pMetis and CHACO.

However as k increases, MMA (but also pMetis) fails to generate partitions of perfect balance in some cases. For imbalanced partitions, we indicate in parentheses the degree of imbalance or '-' if the resulting partition has an imbalance degree greater than 1.07. For these cases, partition balance can not be completely established since the tabu search procedure does not move vertices strictly from the highest to the lightest weight subsets (see Section IV-C1). Although Move 1 of the

TABLE III

COMPARISON OF OUR MMA APPROACH WITH PMETIS AND CHACO FOR $k \in \{2, 4, 8, 16, 32, 64\}$. PMETIS IS PART OF THE METIS FAMILY OF MULTILEVEL PARTITIONING ALGORITHMS. CHACO IS A PACKAGE THAT INTEGRATES A VARIETY OF ALGORITHMS FOR GRAPH PARTITIONING. BESIDE PROVIDING THE VALUES OF PARTITIONS OBTAINED WITH PMETIS AND CHACO, WE SHOW THE BEST (MMA_B) AND AVERAGE (MMA_{Av}) PARTITIONS OBTAINED WITH MMA AFTER 30 RUNS, AS WELL AS THE AVERAGE TIME IN SECONDS. IF THE PARTITION IS IMBALANCED, WE REPORT THE DEGREE OF IMBALANCE BETWEEN PARENTHESES.

	k=2					k=4					k=8				
Graph	pMetis	CHACO	MMA_B	MMA_{Av}	time	pMetis	CHACO	MMA_B	MMA_{Av}	time	pMetis	CHACO	MMA_B	MMA_{Av}	time
add20	729	742	697	**709.0**	0.9	1292	1329	1179	**1205.1**	3.6	1907	1867	1708	**1730.7**	8.4
data	218	199	189	**195.0**	0.8	480	433	383	**409.7**	3.0	842	783	674	**699.6**	3.2
3elt	108	103	90	103.2	1.5	231	234	201	**208.4**	4.8	388	389	348	**359.5**	4.9
uk	23	36	20	24.4	1.5	67	69	43	**50.8**	4.9	101	119	93	102.0	5.1
add32	21	11	10	12.8	1.9	42	56	33	**39.5**	7.5	81	115	66	**76.9**	7.1
bcsstk33	10205	10172	10171	10224.4	17.1	23131	23723	21748	**22119.0**	37.6	40070	39070	34443	**34585.1**	51.3
whitaker3	135	131	127	**127.3**	5.1	406	425	383	**394.3**	17.1	719	765	659	**669.3**	12.7
crack	187	225	184	188.4	6.5	382	445	367	**372.2**	14.1	773	777	685	**711.2**	14.2
wing_nodal	1820	1823	1708	**1721.1**	8.0	4000	4022	3582	**3625.1**	16.9	6070	6147	5445	**5534.6**	19.5
fe_4elt2	130	144	130	148.6	10.4	359	402	349	**354.2**	17.7	654	718	613	**635.7**	15.4
vibrobox	12427	11367	11184	11404.6	15.1	21471	21774	19288	**19664.0**	41.5	28177	33362	24790	**24975.7**	47.3
bcsstk29	2843	3140	2843	3030.1	20.0	8826	9202	8495	**8663.1**	37.3	16555	18158	15760	16587.6	43.9
4elt	154	158	139	179.4	14.3	406	433	327	**360.6**	25.6	635	688	548	**581.3**	26.4
fe_sphere	440	424	386	**386.0**	18.1	872	852	771	**773.8**	26.2	1330	1302	1212	**1231.8**	23.0
cti	334	372	334	340.4	13.6	1113	1117	970	**994.4**	29.4	2110	2102	1801	**1876.6**	31.2
memplus	6337	7549	5556	**5645.4**	29.3	10559	11535	9687	**9916.6**	63.5	13110	14265	12438	**12546.3**	73.8
cs4	414	517	374	**379.0**	28.6	1154	1166	977	**1003.9**	18.3	1746	1844	1497	**1532.7**	43.8
bcsstk30	6458	6563	6394	9664.5	91.6	17685	17106	16681	20002.4	127.7	36357	37406	35909	38441.3	113.9
bcsstk31	3638	3391	2767	3404.1	74.1	8770	9199	7699	**8314.2**	125.3	16012	15551	13465	**15088.7**	150.4
fe_pwt	366	362	340	428.8	85.8	738	911	707	**722.6**	91.2	1620	1670	1452	**1486.5**	92.9
bcsstk32	5672	6137	4667	**5611.8**	170.4	12205	15704	9386	**11203.7**	175.3	23601	25719	21790	**23546.6**	208.6
fe_body	311	1036	262	**291.5**	88.9	957	1415	672	**802.6**	188.8	1348	2277	1115	**1290.5**	137.0
t60k	100	91	84	111.3	176.7	255	235	221	256.7	199.6	561	524	490	524.6	205.2
wing	950	901	814	**842.5**	182.8	2086	1982	1696	**1740.1**	331.4	3205	3174	2595	**2668.6**	228.7
brack2	738	976	731	819.5	173.0	3250	3462	3087	**3199.6**	318.0	7844	8026	7246	**7641.1**	274.4
finan512	162	162	162	194.4	343	324	325	324	448.2	427.8	810	648	648	734.4	429.7
fe_tooth	4297	4642	3822	**4019.1**	277.8	8577	8430	6941	**7110.9**	435.1	13653	13484	11688	**11966.6**	344.2
fe_rotor	2190	2151	2098	**2102.3**	426.2	8564	8215	7310	**7745.1**	525.2	15712	15244	13026	**13693.3**	518.8
598a	2504	2465	2398	**2405.5**	504.8	8533	8975	8044	**8240.1**	645.7	17276	17530	16061	**16524.2**	598.5
fe_ocean	505	499	464	647.2	977.4	2039	2110	1897	**1910.5**	894.6	4516	5309	4210	**4313.1**	984.1

	k=16					k=32					k=64				
Graph	pMetis	CHACO	MMA_B	MMA_{Av}	time	pMetis	CHACO	MMA_B	MMA_{Av}	time	pMetis	CHACO	MMA_B	MMA_{Av}	time
add20	2504	2297	2113	**2113.8**	16.3	–	2684	2447(1.01)	**2439.4**	20.6	3433(1.07)	3349	3050(1.05)	**3068.7**	38.5
data	1370	1360	1154	**1168.3**	3.4	2060(1.01)	2143	1859	**1881.9**	3.9	3116(1.03)	3145	–	–	–
3elt	665	660	579	**589.9**	5.4	1093	1106	978	**988.8**	5.6	1710	1722	1574	**1583.9**	6.4
uk	189	211	164	**182.1**	4.7	316(1.01)	343	288(1.01)	**307.8**	11.2	495(1.02)	540	513	508.2	5.9
add32	128	174	117	129.3	7.9	288(1.01)	303	212(1.01)	**224.9**	7.7	626(1.02)	730	572	**574.8**	8.2
bcsstk33	59791	61890	55522	**55800.7**	54.1	86008	84613	78844	**79374.2**	80.5	116203(1.01)	115530	125407	125275.0	142.4
whitaker3	1237	1218	1101	**1121.6**	11.9	1891	1895	1727	**1750.7**	12.8	2796(1.01)	2811	2594	**2621.2**	14.9
crack	1255	1253	1101	**1142.6**	13.9	1890	1962	1730	**1767.4**	13.71	2847(1.01)	2904	2609(1.01)	**2640.7**	15.8
wing_nodal	9290	9273	8437	**8508.7**	24.3	13237	13258	11990	**12064.7**	23.5	17899(1.01)	17783	16075(1.01)	**16178**	32.9
fe_4elt2	1152	1135	1015	**1041.8**	15.7	1787	1796	1655	**1681.8**	15.1	2765(1.01)	2781	2574	**2585.6**	17.5
vibrobox	37441	43064	33919	**34839.1**	73.2	46112	51006	42579	**45100.9**	94.2	53764(1.01)	58392	55189	54632.1	135.8
bcsstk29	28151	28629	24508	**25711.3**	51.2	41190	42935	36330	**37265.4**	57.9	62891(1.01)	63576	58272(1.01)	**58607.5**	93.2
4elt	1056	1083	951	**983.3**	24.2	1769	1766	1597	**1650.5**	27.3	2953	2921	2640	**2692.9**	28.0
fe_sphere	2030	2037	1752	**1806.3**	26.8	2913	2920	2638	**2686.7**	24.6	4191	4151	3803	**3834.5**	29
cti	3181	3083	2921	**2989.6**	29.9	4605	4532	4243	**4335.3**	28.3	6461	6334	6014	**6070.8**	35.3
memplus	14942	16433	13361	**13558.5**	273.8	17303	17936	14778	**15110.4**	569.6	19140(1.01)	18978	–	–	–
cs4	2538	2552	2160	**2221.7**	44.8	3579	3588	3057	**3111.6**	39.6	4791	4817	4219	**4278.9**	49.8
bcsstk30	77293	81069	76258	**76954.5**	141.9	131405	128694	119413	**123824.0**	267.3	191691	191445	184829	204726.0	662.3
bcsstk31	27180	28557	24934	**26192.0**	147.3	42645	45354	40742	**41573.5**	123.1	66526	68375	61778	**63207.6**	266.1
fe_pwt	2933	3200	2839	**2864.5**	83.4	6029	6036	5783	**5966.2**	92.8	9310	9231	8532	**8577.7**	96.8
bcsstk32	43371	47829	38361	**40966.4**	214.2	70020	73377	64186	**68541.7**	406.7	106733	108855	101861	**106247.0**	711.8
fe_body	2181	2947	2118	2201.7	140.4	3424	4194	3385	3516.2	137.4	5843	6326	5576	**5683.7**	142.5
t60k	998	977	899	**922.9**	228.9	1613	1594	1488	**1549.3**	194.5	2484	2506	2331	**2397.7**	201.7
wing	4666	4671	4076	**4154.1**	235.1	6700	6843	5896	**6001.0**	224.1	9405	9308	8065	**8185.9**	220.6
brack2	12655	13404	12055	**12322.4**	269.6	19786	20172	17855	**18411.1**	295.6	28872	29223	27056	**27853.8**	308.5
finan512	1377	1296	1296	1368.9	391.2	2592	2592	2592	**2592.0**	362.9	10842	11962	10764	10978.4	350.3
fe_tooth	19346	20887	17857	**18204.5**	348.7	29215	29849	25787	**26179.6**	345.5	40162	40306	35864	**36055.7**	456.3
fe_rotor	23863	23936	20694	**21398.3**	460.6	36225	36367	32034	**33831.1**	559.2	53623	52497	48518	**50043.5**	943.2
598a	28922	29674	26361	**26807.0**	578.0	44760	45780	39470	**40244.3**	615.5	64307	65094	58483	**58985.7**	928.9
fe_ocean	9613	9690	7908	**8206.6**	930.8	14613	15059	13237	**13571.8**	908.7	23317	22692	21143	**21554**	1010.3

tabu search procedure generally reduces the partition imbalance after each iteration, the imbalance may not be decreased after an iteration of Move 2 since the balance constraint is only partially imposed. As it can be seen from the experimental results, the balance is always established for $k \leq 16$. For larger k, there is generally a large number of feasible moves implying more freedom for vertex migrations. As a result, it is more difficult to establish a perfect partition balance via the two move operators.

D. *Comparisons with the best known results*

To better assess the performance of our MMA approach, we show in this section experimental results under relaxed time constraint. We prolong the running time from several minutes up to 5 hours for the largest graph (notice that the current most effective evolutionary approach by Soper et al. [38] requires computing time of up to one week to produce state-of-the-art results). The main purpose of this experiment is to know whether our MMA algorithm can improve further the current best solutions.

For comparison, we use the *current best partitions* reported at the Walshaw's Graph Partitioning Archive. The majority of these best partitions are generated with the hybrid evolutionary algorithm presented by Soper et al. [38], which uses JOSTLE multilevel procedure as a black box. Since each run of Soper et al.'s algorithm 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 current best partitions are produced with NetWorks, which is a commercialized version of JOSTLE. The remaining best results are obtained with several other approaches [20], [31], [11]. Since the experimental conditions to obtain the current best results are not available, we focus on comparing solution quality based on the best objective value.

Table IV summarizes the current best results from the Graph Partitioning Archive (column 'Best')[1], the best results obtained by MMA (column 'MMA')[2], as well as the average and standard deviation of partitions obtained by MMA (column 'Avg(Std)'). The last row with heading 'Total' shows the number of times MMA succeeds to improve the current best partition. All the comparisons are carried out between partitions of the same balance. In most cases, the partitions of MMA are perfectly balanced (i.e. $\varepsilon = 1.00$, this is the default balance). For the cases where MMA produces imbalanced partitions ($k \in \{32, 64\}$), we indicate the imbalance in parentheses next to the objective value and compare the partitions with the same imbalance.

The results show that, in the case of bisection, our MMA approach succeeds to reach the same solution quality of more than two thirds of the best balanced bisections reported at the archive. It also improves the best bisection in three cases, and produces, only in four cases, bisections that are less good than the current best ones. More importantly, as k increases ($4 \leq k \leq 64$), MMA improves even 63%, 90%, 93%, 83% and 77% of the current best k-partitions from the archive when k is equal to 4, 8, 16, 32 and 64 respectively.

[1] Results retrieved in June 2010

[2] Our best results are available at: http://www.info.univ-angers.fr/pub/hao/MMAbest.html

VI. Landscape and structural analysis

In this section, we wish to obtain some insight on the search space and provide motivations for the proposed multi-parent crossover operator. For this purpose, we employ the fitness distance analysis (FDA) [22], which investigates the correlation between quality (fitness) of local optima and their distances to the optimum. Additionally, we analyze structural similarity between local optima in terms of backbone size.

A. *Analysis protocol*

We perform the analysis on seven graph partitioning instances of different types, with the cardinal number k set to 4, 8, 16 and 32. The results reported for each graph and k are based on 1500 independent runs of the multilevel perturbation-based tabu search algorithm from Section IV-C, and using the distance measure introduced in Section IV-D1. Since the optimal solutions for the selected instances are not known, we use instead the best local optima found to compute fitness-distance correlation. Table V contains the data to which we will be referring in the following sections.

B. *FDA for selected graph partitioning instances*

The fitness distance correlation (FDC) coefficient ρ_{fdc} [22] is a well-known tool for landscape analysis and can provide useful indications about the problem hardness, even if such an analysis has some known shortcomings and limits. FDC estimates how closely related are the fitness and distance to the nearest optimum. For a minimization problem, if the fitness of a solution decreases with the decrease of distance from the optimum, then it would be easy to reach the target optimum for an algorithm that concentrates around the best candidate solutions found so far, since there is a "path" to the optimum via solutions with decreasing (better) fitness. A value of $\rho_{fdc} = 1$ indicates perfect correlation between fitness and distance to the optimum. For correlation of $\rho_{fdc} = -1$, the fitness function is completely misleading. FDC can also be visualized with the FD plot, where the same data used for estimating ρ_{fdc} is displayed graphically. Such plots have been used to estimate the distribution of local optima for a number of problems including for instance the TSP problem [7], graph bipartitioning problem [30] and flow-shop scheduling problem [35].

In column 'ρ_{fdc}' of Table V, we report FDC coefficients ρ_{fdc} for the 7 selected graphs. For illustrative purpose, FD plots of only two graphs (*3elt* and *vibrobox*) are given in Figure 3 for $k \in \{4, 8, 16, 32\}$. To make the difference in fitness distribution more obvious, we "normalize" the actual fitness values in the FD plots by subtracting from them the best objective value.

As it can be seen from Table V, there is a signification fitness distance correlation in many cases. However, the FDA analysis also reveals the existence of several cases among the selected instances for which there is virtually no correlation

TABLE IV

COMPARISON WITH THE BEST RESULTS FROM THE GRAPH PARTITIONING ARCHIVE (COLUMN 'BEST') AND THE BEST RESULTS OBTAINED BY MMA (COLUMN 'MMA') OVER 20 INDEPENDENT RUNS FOR $k \in \{2, 4, 8, 16, 32\}$. COLUMN 'AVG(STD)' PROVIDES THE AVERAGE AND STANDARD DEVIATION OF PARTITIONS OBTAINED WITH MMA. IF THE PARTITION IS IMBALANCED, WE REPORT THE DEGREE OF IMBALANCE BETWEEN PARENTHESES.

	k=2			k=4			k=8		
Graph	Best	MMA	Avg(Std)	Best	MMA	Avg(Std)	Best	MMA	Avg(Std)
add20	**596**	678	708.5 (20.6)	1203	**1159**	1187.6 (17.1)	1714	**1696**	1705.5 (14.1)
data	**189**	**189**	189.0 (0.0)	383	**382**	391.6 (8.4)	679	**669**	675.6 (4.3)
3elt	**90**	**90**	90.0 (0.0)	**201**	**201**	202.4 (2.2)	348	**345**	346.8 (1.0)
uk	20	**19**	20.8 (0.9)	43	**42**	43.1 (0.7)	89	**84**	87.1 (2.2)
add32	11	**10**	10.3 (0.46)	34	**33**	34.8 (1.89)	75	**66**	68.9 (4.0)
bcsstk33	**10171**	**10171**	10171.0 (0.0)	**21719**	21730	22193.6 (431.8)	34579	**34455**	34491.3 (34.9)
whitaker3	**127**	**127**	127.0 (0.0)	**382**	**382**	382.1 (0.2)	661	**658**	659.4 (1.3)
crack	**184**	**184**	184.0 (0.0)	368	**366**	366.0 (0.0)	687	**679**	686.6 (5.7)
wing-nodal	**1707**	**1707**	1707.8 (0.4)	3581	**3578**	3608.6 (27.0)	5443	**5438**	5481.8 (42.1)
fe-4elt2	**130**	**130**	130.0 (0.0)	**349**	**349**	349.0 (0.0)	610	**609**	615.8 (5.6)
vibrobox	**10343**	**10343**	10984.5 (265.4)	19245	**19138**	19534.1 (217.2)	24715	**24583**	24747.8 (74.7)
bcsstk29	**2843**	**2843**	2846.0 (3.3)	**8159**	8475	8484.9 (13.4)	**14322**	15340	15905.8 (247.0)
4elt	**139**	**139**	139.2 (0.7)	**326**	**326**	329.6 (4.3)	548	**547**	548.5 (2.5)
fe-sphere	**386**	**386**	386.0 (0.0)	**770**	**770**	771 (0.9)	1193	**1165**	1182.2 (18.3)
cti	**334**	**334**	334.0 (0.0)	963	**955**	970.0 (5.3)	1812	**1795**	1841.1 (21.4)
memplus	**5513**	5524	5587.6 (53.2)	**9643**	9646	9792.5 (56.6)	**11872**	11879	12040.1 (132.9)
cs4	**371**	**371**	374.0 (1.6)	964	**934**	962.1 (14.6)	1496	**1455**	1474.9 (12.3)
bcsstk30	**6394**	**6394**	6394.0 (0.0)	**16652**	**16652**	16856.0 (220.9)	34921	**34910**	34948.4 (34.9)
bcsstk31	**2762**	**2762**	2768.1 (6.5)	7469	**7355**	7621.6 (140.2)	13812	**13370**	13755.3 (383.3)
fe-pwt	**340**	**340**	358.1 (5.3)	709	**707**	718.8 (5.7)	1465	**1450**	1465.1 (22.1)
bcsstk32	**4667**	**4667**	4679.5 (23.5)	9492	**9318**	9383.4 (53.0)	22757	**21119**	22377 (786.4)
fe-body	**262**	**262**	262.0 (0.0)	703	**624**	661.5 (13.1)	1234	**1055**	1086.9 (30.5)
t60k	**79**	83	85.5 (1.2)	**213**	218	222.3 (1.3)	476	**474**	486.9 (11.1)
wing	**791**	798	806.4 (5.3)	1666	**1644**	1672.9 (22.0)	2589	**2525**	2564.3 (24.7)
brack2	**731**	**731**	731.0 (0.0)	3090	**3084**	3100.5 (24.7)	7269	**7151**	7268.2 (104.9)
finan512	**162**	**162**	162.0 (0.0)	**324**	**324**	336.2 (28.9)	**648**	**648**	656.1 (24.3)
fe-tooth	3850	**3819**	3876.5 (78.7)	7142	**6919**	6969.1 (67.6)	11935	**11475**	11680.5 (173.3)
fe-rotor	**2098**	**2098**	2103.8 (10.0)	7480	**7277**	7630.7 (195.5)	13292	**12912**	13152.4 (139.0)
598a	**2398**	**2398**	2398.9 (1.0)	8154	**8016**	8072.6 (43.3)	16884	**15938**	16160.2 (115.6)
fe-ocean	**464**	**464**	467.6 (1.2)	1902	**1895**	1898.9 (2.9)	4299	**4205**	4233.5 (16.5)
Total	4	3		4	19		2	27	

	k=16			k=32			k=64		
Graph	Best	MMA	Avg(Std)	Best	MMA	Avg(Std)	Best	MMA	Avg(Std)
add20	2149	**2064**	2073.6 (7.5)	2493(1.03)	**2387**(1.03)	2402.9 (8.9)	3152(1.03)	**3021**(1.03)	3021.1 (7.9)
data	1162	**1135**	1146 (6.3)	**1802**(1.03)	1824(1.02)	1836.8 (7.0)	**2798**	–	–
3elt	581	**573**	575.6 (2.4)	**969**(1.01)	**969**(1.01)	972.3 (2.57)	1564(1.01)	**1554**(1.01)	1557.2 (2.2)
uk	159	**153**	158 (2.6)	**258**(1.01)	264(1.01)	273.0 (4.5)	**438**(1.01)	454(1.01)	460.7 (5.0)
add32	121	**117**	122.4 (5.9)	**212**(1.01)	**212**(1.01)	215.4 (8.6)	**493**	499	514.2 (7.7)
bcsstk33	55136	**54763**	55250.5 (337.7)	78132	**61047**	61984 (552.3)	108505(1.01)	**107862**(1.01)	108144 (219.7)
whitaker3	1108	**1095**	1102.3 (4.3)	1718	**1697**	1708.4 (4.3)	2569	**2552**	2563.1 (8.3)
crack	1108	**1094**	1111.4 (10.3)	1728	**1693**	1704.6 (5.1)	2566(1.01)	**2561**(1.01)	2574.3 (6.4)
wing-nodal	8422	**8359**	8404.1 (29.4)	12080	**11828**	11891.1 (34.55)	16134(1.01)	**15888**(1.01)	15911.1 (28.7)
fe-4elt2	1018	**1010**	1013.1 (3.4)	1657	**1633**	1643.8 (7.3)	2537	**2519**	2533.2 (6.4)
vibrobox	32654	**32532**	33207.3 (249.7)	42187	**40098**	40607.2 (282.0)	49521(1.01)	**48040**(1.01)	48794 (1006.3)
bcsstk29	**22869**	24106	25167.5 (694.5)	36104	**35637**	36100.3 (239.3)	57054(1.01)	**56792**(1.01)	57640.1 (448.8)
4elt	956	**942**	950.2 (6.096)	1592	**1563**	1577.7 (9.5)	2636	**2596**	2603.8 (6.4)
fe-sphere	1750	**1734**	1739.4 (3.1)	2567	**2542**	2565.3 (12.4)	3663	**3625**	3655.7 (15.7)
cti	2909	**2837**	2894.5 (27.8)	4288	**4142**	4200.0 (33.1)	5955	**5818**	5862.8 (36.0)
memplus	13516	**13054**	13099.0 (31.1)	14634	**14501**	14601.6 (75.5)	17446	–	–
cs4	2206	**2107**	2136.8 (15.1)	3110	**2938**	2979.7 (16.5)	4223	**4051**	4095.7 (19.8)
bcsstk30	72007	**70910**	71978.7 (411.9)	119164	**113788**	115716 (1030.4)	**173945**(1.01)	174982(1.01)	176496 (1066.3)
bcsstk31	24551	**23807**	24152.2 (226.3)	38484	**37927**	38432.7 (447.0)	60724	**58241**	58651.4 (230.4)
fe-pwt	2855	**2838**	2845.0 (6.1)	5758	**5663**	5693.1 (24.3)	8495	**8338**	8358.4 (15.0)
bcsstk32	38711	**36518**	37225.7 (506.1)	63856	**60898**	61670.1 (538.7)	95199	**91863**	93633.2 (826.3)
fe-body	2057	**1834**	1890.6 (36.0)	3371	**3060**	3101.7 (35.1)	5460	**4903**	5021.7 (47.7)
t60k	**866**	881	890.1 (6.3)	1440	**1431**	1453.5 (10.6)	**2233**	2260	2273.7 (7.9)
wing	4198	**3921**	3960.9 (27.2)	6009	**5643**	5703.7 (33.2)	8132	**7690**	7752.7 (33.8)
brack2	12323	**11689**	11859.5 (92.6)	18229	**17398**	17612.7 (135.8)	27178	**25997**	26154.6 (108.6)
finan512	**1296**	**1296**	1356.8 (35.1)	**2592**	**2592**	2592.0 (0.0)	**10560**	**10560**	10662.3 (82.6)
fe-tooth	18382	**17428**	17636.3 (96.3)	26346	**24985**	25292.4 (178.0)	35980	**34433**	34688.9 (100.9)
fe-rotor	21241	**20438**	20711 (129.7)	32783	**31369**	31720.5 (257.7)	49381	**45984**	46364.3 (209.0)
598a	26427	**25783**	26095.5 (147.1)	41538	**38682**	38939.2 (161.1)	59708	**56260**	56574.5 (163.4)
fe-ocean	8622	**7803**	7944.7 (98.8)	14277	**12903**	13032 (75.7)	22301	**20146**	20331.3 (148.1)
Total	2	27		2	25		5	23	

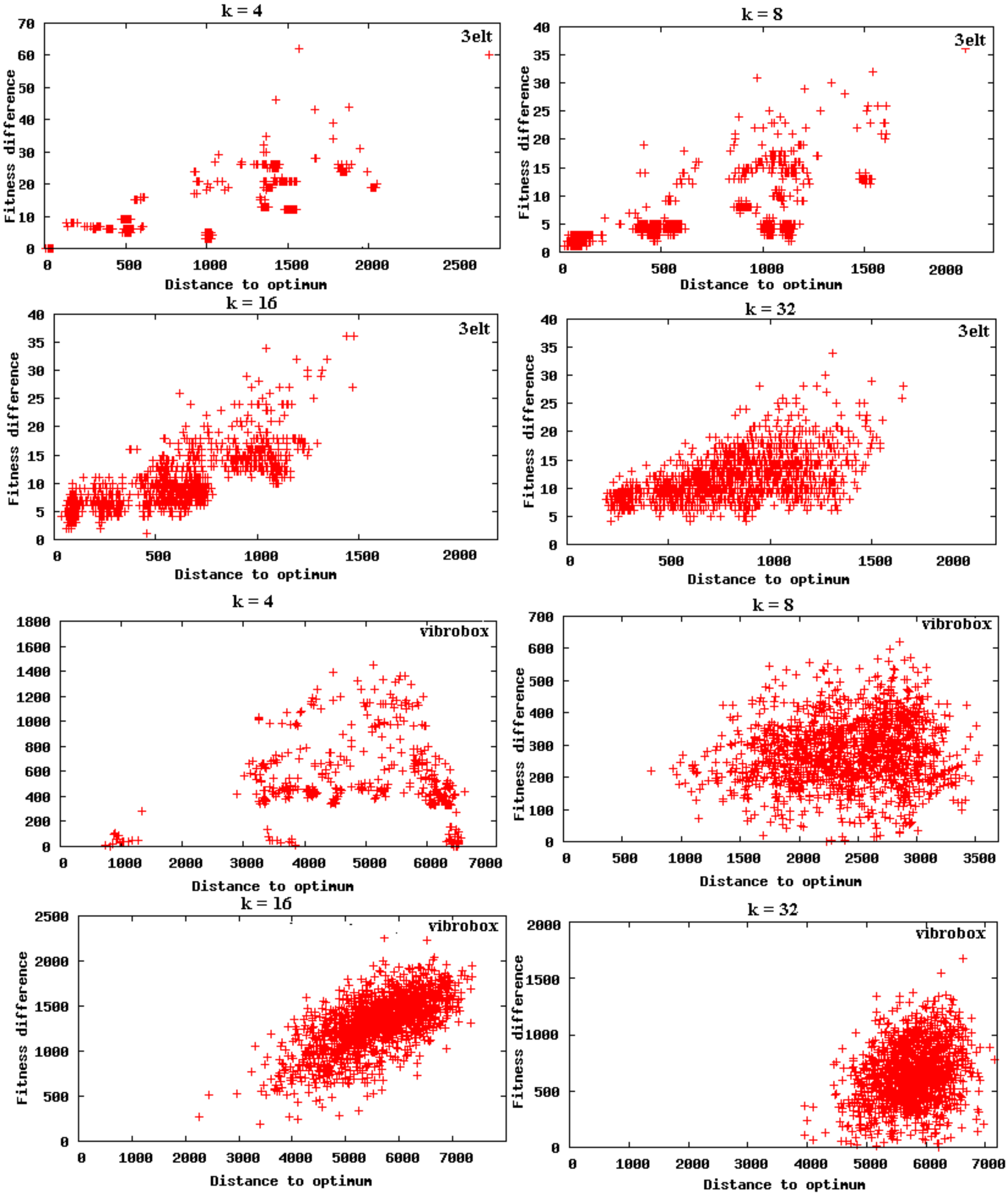


Fig. 3. FD correlation plots with respect to the normalized solution fitness and distance to the optimum for *3elt* and *vibrobox* when $k \in \{4, 8, 16, 32\}$. The first four plots are related to the *3elt* graph, while the last four are related to the *vibrobox*.

between fitness and distance, i.e. cases where $\rho_{fdc} < 0.15$. Indeed, from plots in Figure 3, it is clear that there is practically no correlation for 'vibrobox' when $k \in \{4, 8\}$. In addition, the correlation is weak for 'vibrobox' when $k = 32$. On the other hand, the strongest correlation is observed for graph '3elt' when $k \in \{4, 16\}$ and 'vibrobox' when $k = 16$. The presence of significant FD correlation in many cases explains to some extent why the local optimization engine (tabu search) used in MMA is extremely powerful.

The strong correlation between solution quality (fitness) and its distance to the reference solution also indicates the presence of a big valley structure in the search landscape around the selected local optimum [8]. Intuitively, in this structure the global optimum (in our case, the best local optimum) is surrounded by many local optima whose fitness values deteriorate with the increase of distance from the optimum. To investigate the existence of the big valley structure, we provide in Figure 4 plots with respect to solution fitness and *average* distance between any two solutions of a given set of local optima. As it can be expected, the plots give further evidence for the big valley structure in cases of graph '3elt'

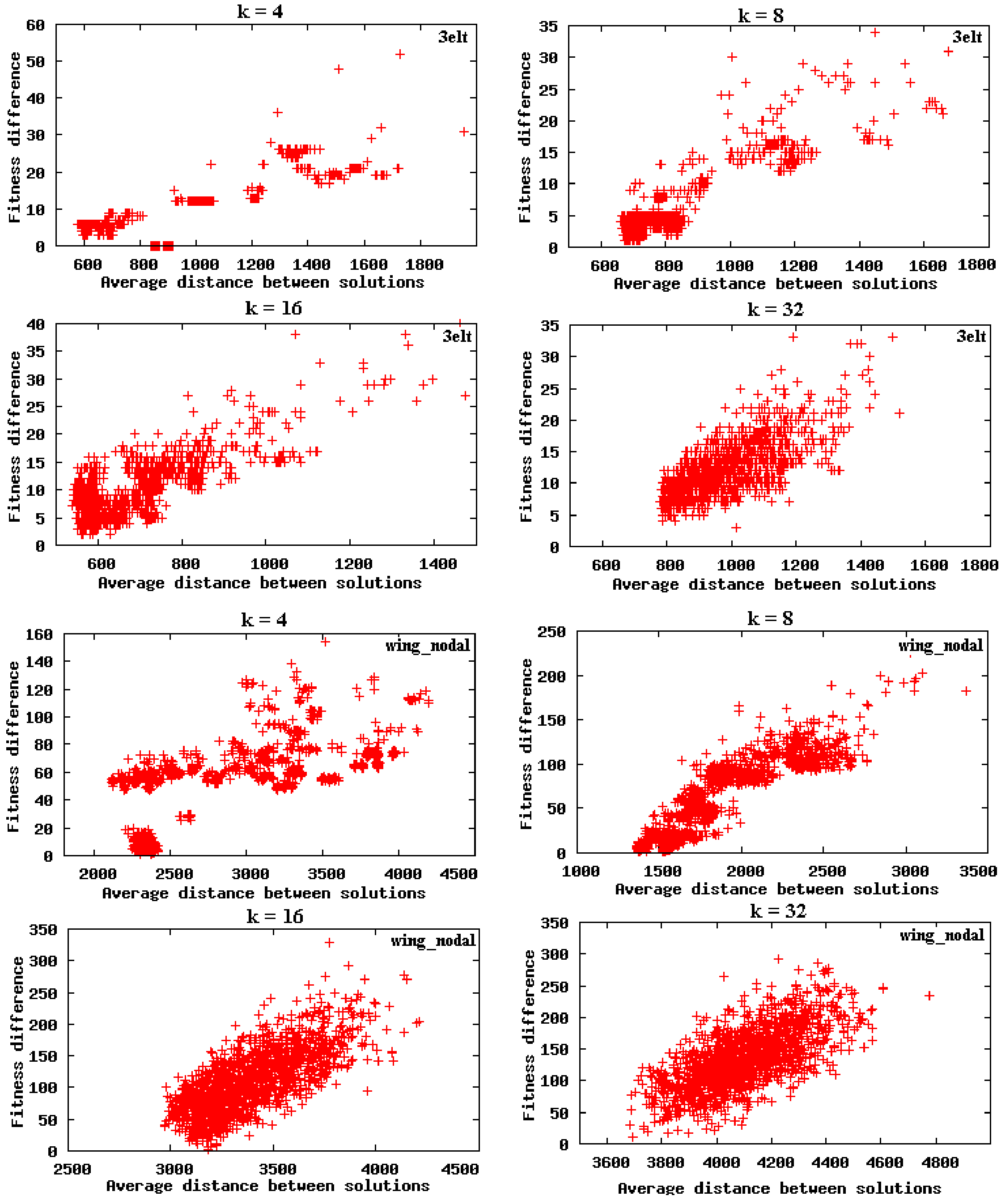


Fig. 4. FD correlation plots with respect to the normalized solution fitness and the average distance between any two solutions of a given set of local optima for *3elt* and *vibrobox* when $k \in \{4, 8, 16, 32\}$. The first four plots are related to *3elt*, while the last four are related to the *vibrobox*.

and 'vibrobox' when $k = 16$, i.e. high correlation between fitness and average distance between any two local optima. On the other hand, such correlation is not visible in case of 'vibrobox' when $k \in \{4, 8, 32\}$.

The big valley structure implies that high-quality local optima tend to be positioned centrally within the region of sampled local optima. Although we did not include fitness-distance plots for all the analyzed graph partitioning instances, except for some very rare cases, these plots confirm our observation on the distribution of local optima in the search space.

For informative purpose, columns '*avg* d_{lo}' and '*avg* d_{go}' from Table V report respectively the average distance between local optima and the average distance of local optima from the best local optimum, expressed as a percentage of $|V|$. Given that the maximum distance between any two solutions is $|V|$, these results also confirm that local optima are not uniformly distributed, but are rather concentrated within a limited number of regions in the search space.

C. Backbone analysis and motivation for the BBC operator

To evaluate the degree of similarity between local optima (including global optima which are technically speaking also local optima), we provide an analysis of the backbone size.

TABLE V

ANALYTICAL RESULTS FOR SEVEN GRAPH PARTITIONING INSTANCES WHEN $k \in \{4, 8, 16, 32\}$. COLUMNS 'd_{lo}' AND 'd_{go}' REPORT RESPECTIVELY THE AVERAGE DISTANCE BETWEEN LOCAL OPTIMA AND THE AVERAGE DISTANCE OF LOCAL OPTIMA FROM THE BEST LOCAL OPTIMUM, EXPRESSED AS A PERCENTAGE OF $|V|$. COLUMN 'ρ_{fdc}' SHOWS THE CORRELATION COEFFICIENTS WITH RESPECT TO FITNESS AND DISTANCE, WHILE COLUMNS '$avgB_{lo}$' AND 'B_{hq}' INDICATE RESPECTIVELY (IN PERCENTAGE OF $|V|$) THE AVERAGE BACKBONE SIZE WITH RESPECT TO 10 RANDOMLY CHOSEN LOCAL OPTIMA, AND THE BACKBONE SIZE WITH RESPECT TO BEST LOCAL OPTIMA FOUND DURING THE SEARCH.

	k=4					k=8				
Graph	*avg* d_{lo}	*avg* d_{go}	ρ_{fdc}	*avg* B_{lo}	B_{hq}	*avg* d_{lo}	*avg* d_{go}	ρ_{fdc}	*avg* B_{lo}	B_{hq}
data	30.5	34.8	0.57	21.5	95.2	17.8	16.0	0.68	46.6	64.7
3elt	19.1	18.7	0.7	50.0	97.3	17.2	14.6	0.53	58.2	69.6
uk	18	14.3	0.61	60.2	68.4	26.3	25.7	0.24	34.2	51.4
crack	3.5	2.2	0.89	98.3	98.9	22.5	19.6	0.51	59.8	95.4
wing-nodal	26.1	21.6	0.81	36.5	91.5	17.1	13.6	0.91	53.4	96.0
fe-4elt2	9.8	6.7	0.74	61.9	88.0	26.0	24.4	0.68	28.3	62.4
vibrobox	40.1	41.4	-0.02	21.7	40.7	22.4	19.7	0.03	54.1	52.4
	k=16					k=32				
Graph	*avg* d_{lo}	*avg* d_{go}	ρ_{fdc}	*avg* B_{lo}	B_{hq}	*avg* d_{lo}	*avg* d_{go}	ρ_{fdc}	*avg* B_{lo}	B_{hq}
data	22.5	23.7	0.08	42.3	42.2	24.9	23.1	0.6	32.4	50.9
3elt	14.0	12.1	0.75	61.8	77.2	20.5	17.1	0.53	43.3	61.5
uk	26.9	25.1	0.33	32.9	30.0	27.4	24.9	0.44	34.6	61.5
crack	27.7	22.9	0.74	24.8	79.6	28.1	26.3	0.58	24.5	33.8
wing-nodal	31.0	27.3	0.56	25.2	33.2	37.5	35.6	0.4	13.5	15.5
fe-4elt2	16.4	14.7	0.51	55.3	73.5	28.7	25.5	0.51	20.1	34.3
vibrobox	41.5	45.5	0.65	12.3	7.5	49.7	46.8	0.21	4.6	3.2

While the column '*avg* B_{lo}' from Table V reports the average backbone size with respect to 10 randomly chosen local optima, column 'B_{hq}' presents the backbone size with respect to best local optima found during the search. The backbone size is expressed as the percentage of $|V|$. From these results, we observe that except for very few cases, the backbone is generally of significant size. We also note that the values reported in 'B_{hq}' are generally higher than the ones in '*avg* B_{lo}', which indicates that the structure of a set of high quality solutions is very similar to the structure of the supposed global optimum. This suggests that if a significant number of vertices is grouped together throughout each of the high quality partitions, there is a strong chance that they are also grouped together in the global optimum. This observation constitutes the first motivation for the BBC crossover which tries to preserve the backbones through the search process.

On the other hand, the FD analysis above shows the presence of a big valley structure and high fitness-distance correlation in many cases. This provides justifications about why the tabu search based local optimization is important within our memetic approach. This additionally gives an argument for preferring a constructive crossover operator like BBC over highly destructive ones like uniform crossover. To enforce this comment, we show in Section VII a computational comparison between the BBC crossover and a uniform crossover, and study the influence of the perturbation strength within BBC.

VII. A COMPARATIVE ANALYSIS OF THE BBC OPERATOR

A. Comparison with a traditional crossover operator

We compare the performance of the BBC operator with an adapted uniform crossover on the set of 30 instances of the Graph Partitioning Archive for $k \in \{4, 8, 16, 32\}$. For the uniform crossover, each vertex in the offspring partition keeps with equal probability the subset of either parent partition, with the constraint that the subset weight in the offspring partition does not exceed W_{opt} (see Section II). In addition, we reinforce the randomness of the crossover by performing some vertex swaps after the uniform crossover.

In order to highlight the role of the crossover operators, we set the number of generations to 1000 and reduce the number of tabu search iterations to $0.5{*}|V_m|$, where $|V_m|$ is the number of vertices in the m^{th} graph level. We execute 20 times the two versions of our multilevel memetic algorithm, i.e. with BBC and uniform operators, and report the results in Table VI. For each value of k, columns 'b. BBC' and 'b. UNI' report respectively the best partition obtained with our approach integrating the BBC and uniform crossover, while columns 'BBC Avg(Std)' and 'UNI Avg(Std)' report respectively the average and standard deviation of the generated partitions when BBC and uniform crossovers are employed. A lower average value is indicated in bold. In addition, we perform a statistical analysis using the Welch's *t*-test, and report in column '*p*-value' the two tailed *p*-value over the two partition sets.

From Table VI we observe that there is no significant statistical difference between the solution sets for lower values of k, i.e. $k \in \{4, 8\}$. However, as k increases, we note that our BBC operator visibly outperforms the uniform crossover in almost each case for $k \in \{16, 32\}$. One explanation is that intuitively, given the semantics of the BBC crossover, larger k would favor the preservation of backbone information by BBC whereas the number of parts has a weak influence for the uniform crossover operator as to backbone preservation. Additionally, we compare in Table VII-A the average percentage of perturbed vertices over 20 generations with the two crossovers as well as with a variant of the BBC (see Section VII-B) for $k \in \{4, 8, 16, 32\}$. The perturbation strength is expressed as the minimum (set-theoretic partition) distance between a resulting individual and the parents participating in the crossover. As it can be expected, the perturbation introduced by the proposed crossover is always significantly weaker than the one introduced with the uniform crossover. Moreover, the degree of perturbation introduced with the uniform crossover increases with k. This may constitute another explanation why the BBC performs better than the uniform crossover for larger values of $k \in \{16, 32\}$, since according to the observations

TABLE VI

A comparison of partitions obtained with BBC and Uniform crossover operator on the set of 30 benchmark graphs for $k \in \{4, 8, 16, 32\}$. We report the best partition obtained with BBC (b. BBC) and Uniform operator (b. UNI) , the average and standard deviation of partitions obtained with BBC (BBC Avg(Std)) and Uniform crossover (UNI Avg(std)), and the p-value between the two partition sets. If the partition is imbalanced, we include the degree of imbalance in parentheses.

	k=4					k=8				
Graph	b. BBC	BBC Avg(Std)	b. UNI	UNI Avg(Std)	p-value	b. BBC	BBC Avg(Std)	b. UNI	UNI Avg(Std)	p-value
add20	1159	1196.9 (23.5)	1154	**1188.8** (14.5)	0.199	1696	1729.2 (23.2)	1698	**1725.0** (24.5)	0.581
data	383	419.3 (21.1)	395	**410.2** (10.4)	0.095	681	**725.9** (29.7)	712	747.9 (35.6)	0.041
3elt	201	222.6 (20.9)	204	**220.5** (13.5)	0.708	348	**369.6** (23.0)	348	385.2 (27.1)	0.057
uk	43	49.2 (6.5)	45	**48.8** (4.2)	0.819	88	**96.0** (6.8)	102	115.4 (9.6)	0.000
add32	41	**42.0** (6.9)	33	42.8 (8.6)	0.748	67	**76.4** (7.2)	74	96.0 (17.1)	0.000
bcsstk33	21779	**22234.5** (402.4)	21779	22536.5 (605.0)	0.072	34430	34689.8 (471.8)	34480	**34683.0** (282.2)	0.956
whitaker3	384	403.1 (24.1)	381	**393.5** (15.0)	0.141	659	**670.4** (10.0)	662	685.0 (24.1)	0.019
crack	366	394.1 (32.5)	366	**385.5** (26.9)	0.368	687	**714.7** (15.8)	687	727.5 (34.5)	0.143
wing_nodal	3582	3646.3 (45.0)	3579	**3643.4** (33.6)	0.819	5457	5590.6 (115.9)	5454	**5515.6** (41.0)	0.012
fe_4elt2	349	379.8 (31.4)	349	**365.1** (26.3)	0.117	608	644.0 (28.2)	614	**640.6** (16.8)	0.646
vibrobox	19228	**19795.8** (336.0)	19170	19832.8 (331.6)	0.728	24935	25038.7 (311.2)	24739	**25011.0** (150.3)	0.723
bcsstk29	8475	**8613.0** (280.7)	8493	8981.2 (754.7)	0.052	15586	**16778.7** (709.3)	16025	16945.0 (554.8)	0.415
4elt	326	**364.0** (24.2)	326	366.1 (22.7)	0.779	546	**608.0** (36.0)	570	690.7 (66.6)	0.000
fe_sphere	770	**770.3** (0.6)	770	770.8 (0.9)	0.047	1208	**1225.5** (12.6)	1216	1242.4 (22.8)	0.007
cti	971	1052.0 (86.0)	954	**1028.4** (87.4)	0.395	1808	1927.9 (62.1)	1799	**1896.9** (53.8)	0.100
memplus	9677	9794.3 (98.6)	9628	**9688.9** (55.6)	0.000	12476	12533.5 (35.9)	12164	**12517.6** (178.4)	0.700
cs4	971	**995.5** (16.7)	970	995.7 (25.8)	0.977	1502	1531.6 (20.7)	1485	**1510.8** (15.8)	0.001
bcsstk30	16671	**19802.4** (4547.3)	16695	19878.4 (4598.1)	0.959	35810	**38342.3** (3451.6)	35904	38490.6 (3533.7)	0.894
bcsstk31	7642	8633.7 (994.1)	7396	**7955.0** (574.1)	0.013	13730	**15336.7** (1272.6)	13675	15447.5 (1224.8)	0.781
fe_pwt	722	744.3 (54.6)	707	**736.4** (59.7)	0.665	1462	1546.5 (98.2)	1453	**1506.2** (31.8)	0.094
bcsstk32	9499	13218.8 (2161.9)	9312	**10907.5** (1423.9)	0.000	21420	**23211.9** (788.3)	22403	24198.1 (1318.1)	0.007
fe_body	660	826.7 (99.7)	691	**821.5** (97.0)	0.868	1079	**1164.5** (60.9)	1122	1196.5 (78.4)	0.158
t60k	219	**248.3** (44.6)	223	252.1 (34.5)	0.765	484	**518.3** (17.5)	500	536.9 (26.6)	0.013
wing	1665	**1730.3** (55.7)	1680	1763.2 (55.1)	0.068	2575	**2666.8** (72.3)	2629	2716.1 (61.5)	0.026
brack2	3097	**3235.1** (208.7)	3084	3275.7 (417.4)	0.700	7491	7800.7 (207.3)	7222	**7756.4** (291.4)	0.583
finan512	324	**490.1** (129.3)	324	494.1 (72)	0.905	729	**866.7** (109.0)	729	907.2 (125.5)	0.283
fe_tooth	6933	**7273.5** (328.5)	6975	7304.3 (362.4)	0.780	11875	12188.5 (215.2)	11591	**11952.5** (221.6)	0.002
fe_rotor	7296	**7888.1** (419.2)	7495	7932.4 (429.9)	0.744	13184	**13784.2** (458.1)	13396	14039.2 (440.0)	0.081
598a	8071	8352.4 (313.2)	8058	**8321.7** (396.2)	0.787	16032	16899.1 (632.6)	16124	**16888.0** (670.6)	0.957
fe_ocean	1890	**2007.9** (270.2)	1890	2220.0 (425.5)	0.069	4224	4502.3 (257.2)	4211	**4389.7** (149.7)	0.100
	k=16					k=32				
Graph	b. BBC	BBC Avg(Std)	b. UNI	UNI Avg(Std)	p-value	b. BBC	BBC Avg(Std)	b. UNI	UNI Avg(Std)	p-value
add20	2073	**2094.9** (13.7)	2145(1.01)	2201.7 (30.6)	0.000	2436(1.03)	**2425.9** (13.8)	2541(1.03)	2620.3 (84.9)	0.000
data	1138	**1164.3** (18.5)	1348	1409 (39.4)	0.000	1833(1.07)	**1870.0** (20.7)	204 (1.01)	2074.5 (37.2)	0.000
3elt	600	**595.7** (10.0)	661	734.3 (30.2)	0.000	971	**988.6** (11.7)	1076	1116.3 (21.2)	0.000
uk	155	**180.2** (16.5)	211	227.3 (9.5)	0.000	330	**344.1** (8.3)	342(1.01)	353.7 (8.1)	0.001
add32	123	**134.4** (8.8)	172(1.01)	196.9 (17.5)	0.000	215	**238.7** (26.0)	275(1.01)	332.5 (24.3)	0.000
bcsstk33	55318	**55854.9** (486.9)	55411	56477 (650.5)	0.002	77990	**79380.1** (1013.7)	80642	82517.8 (1862.8)	0.000
whitaker3	1114	**1128.4** (11.5)	1204	1292.9 (40.5)	0.000	1862	**1802.9** (60.6)	1840	1905.4 (29.2)	0.000
crack	1109	**1151.9** (21.8)	1301	1384.3 (39.3)	0.000	1907	**1895.0** (78.1)	1890	1975.8 (46.4)	0.000
wing_nodal	8478	**8550.5** (61.8)	8832	8962.5 (279.3)	0.000	12291	**12383.8** (78.6)	12400	12561.8 (143.5)	0.000
fe_4elt2	1025	**1054.9** (18.6)	1114	1297.4 (83.4)	0.000	1673	**1765.0** (83.5)	1833	1902.1 (44.0)	0.000
vibrobox	33613	**35292.2** (784.2)	35975	37234.7 (563.2)	0.000	43861	**45060.1** (1090.2)	44462	45463.8 (1493.1)	0.336
bcsstk29	25567	25849.3 (705.3)	24456	**25586.9** (673.4)	0.236	36514	**37362.3** (876.1)	39819	41205.5 (748.5)	0.000
4elt	959	**992.8** (53.3)	1154	1223.8 (43.1)	0.000	1706	**1783.3** (68.3)	1789	1863.7 (37.6)	0.000
fe_sphere	1734	**1808.8** (45.2)	1969	2072.3 (59.5)	0.000	2636	**2825.8** (66.6)	2826	2886.2 (29.9)	0.001
cti	2936	**3020.0** (54.1)	2962	3085.9 (84.1)	0.024	4237	**4365.4** (78.3)	4870	5025.0 (88.7)	0.000
memplus	13733	**14061.1** (204.4)	13999	14183.5 (103.4)	0.024	14659(1.01)	**14907.7** (255.8)	–	–	–
cs4	2178	**2220.9** (18.0)	2198	2361.5 (110.7)	0.000	3110	**3273.8** (120.1)	3347	3424.8 (48.0)	0.000
bcsstk30	76258	**76954.5** (2185.0)	82533	87968.0 (4704.3)	0.000	124969	**128905.8** (2207.1)	131402	134148.5 (2429.8)	0.000
bcsstk31	24484	**25796.5** (849.1)	26360	30709.5 (2517.2)	0.000	41529	**43540.6** (1173.0)	43758	47259.5 (1593.3)	0.000
fe_pwt	2847	**2891.8** (41.1)	2869	3052.1 (94.5)	0.000	5872	**6268.7** (138.0)	6304	6356.3 (42.0)	0.013
bcsstk32	38535	**42422.7** (3105.7)	46771	51044.6 (3010.2)	0.000	66966	**71105.2** (2711.7)	70806	74726.5 (2168.4)	0.000
fe_body	1890	**2103.2** (131.7)	1986	2222.8 (135.1)	0.007	3444	**3532.1** (64.1)	3809	3697.8 (81.4)	0.000
t60k	900	**908.5** (11.8)	1049	1122.4 (49.5)	0.000	1673	**1702.2** (37.5)	1661	1732.1 (39.3)	0.019
wing	4097	**4155.6** (50.8)	4098	4653.4 (150.8)	0.000	6054	**6560.8** (242.0)	6572	6718.7 (108.9)	0.013
brack2	12019	**12259.5** (234.3)	13019	13695 (574.0)	0.000	18413	**19588.4** (541.7)	19613	20495.7 (504.5)	0.000
finan512	1296	**1458.0** (88.7)	1539	1680.8 (109.9)	0.000	2592	**2592.0** (0.0)	2592	**2592.0** (0.0)	–
fe_tooth	17993	**18342.5** (198.6)	18337	19521.8 (990.0)	0.000	27333	**27785.9** (986.0)	28373	29090.2 (472.9)	0.000
fe_rotor	21007	**21473.4** (380.2)	21392	24644.9 (1776.4)	0.000	34348	**35049.2** (2134.0)	39322	40929.3 (806.2)	0.000
598a	26194	**27014.8** (359.8)	26534	28206.1 (1385.0)	0.001	39711	**40351.5** (426.7)	41713	46046.0 (2017.5)	0.000
fe_ocean	7982	8262.4 (193.9)	7965	**8215.4** (169.6)	0.420	13158	13880.0 (447.3)	13321	**13519.8** (200.6)	0.003

TABLE VII

PERCENTAGE OF PERTURBED VERTICES WITH BBC, SBBC AND UNIFORM CROSSOVER (UNI) FOR $k \in \{4, 8, 16, 32\}$. FOR BBC AND SBBC THE NUMBER OF PARENTS $p = 6$, WHILE FOR UNI $p = 2$.

	k=4			k=8			k=16			k=32		
Graph	BBC	$SBBC$	UNI	BBC	$SBBC$	UNI	BBC	$SBBC$	UNI	BBC	$SBBC$	UNI
data	4.64	48.86	29.40	4.12	5.89	30.14	6.89	40.17	33.70	14.51	55.32	44.23
3elt	3.79	14.87	25.50	4.36	7.59	29.83	6.65	21.33	32.25	11.48	46.89	41.47
uk	8.36	26.35	25.92	5.38	39.38	31.52	24.13	60.04	32.71	10.82	69.14	35.28
crack	3.04	5.27	25.26	8.10	9.78	29.58	6.16	9.46	32.56	8.38	60.45	35.01
wing-nodal	2.88	3.62	25.58	2.26	5.26	29.64	9.90	9.06	31.75	12.73	61.00	34.75
fe-4elt2	5.17	9.58	25.83	4.92	51.24	29.94	6.65	15.65	32.21	8.80	71.99	35.14
vibrobox	2.12	57.77	25.38	2.90	6.07	29.58	31.14	75.16	32.40	17.20	94.15	56.26

from the analysis in Section VI, a weaker form of perturbation should be used because of the generally high FDC and the marked big valley structure in the landscape.

To see the convergence behavior of the proposed algorithm, we provide in Figure 5 evolution curves of the multilevel memetic algorithm respectively with the BBC and uniform crossovers. The curves are plotted over 300 generations performed on each graph level ($G_m, G_{m-1}, .., G_0$) for graph 'data' with $k = 4$ and 'fe-4elt2' when $k = 32$. The *x*-axis corresponds to the current number of generations, where the first 300 generations are performed on G_m, the 300–600 generations on G_{m-1}, etc. The *y*-axis shows the 'normalized' average fitness value obtained by subtracting the best fitness value over 10 independent executions. Both crossover operators start the refinement on a population of the same quality at the coarsest graph G_m. One observes that for graph 'data' with $k = 4$, the BBC is outperformed by the uniform crossover, while the situation is opposite for 'fe-4elt2' with $k = 32$. From the two curves, we also note that the algorithm converges in both cases toward its best partition after approximately the same number of generations. This is partially due to the quality-and-distance based population updating strategy of our MMA which ensures a healthily diversified population.

B. Perturbation strength of BBC operator

In this section, we provide an analysis on the impact of perturbation strength introduced by our new multi-parent crossover operator (BBC) by comparing the performance of BBC from Section IV-B2 with its slight variation (call it SBBC) which mainly differs from BBC in the degree of perturbation introduced in offspring. As explained in Section IV-B2, BBC preserves all the vertices from the backbone $B = \{B_1, ..., B_k\}$ with respect to p parent individuals, and perturbs with a certain probability a vertex v if it is not present in any subset of B (see alg. 3, lines 12–13). On the other hand, SBBC consists in preserving only the vertices $v \in \{B_1 \cup ... \cup B_k\}$ and assigning the rest of vertices to random subsets S_r of I^0, such that $W(S_r \cup \{v\}) \leq W_{opt}$. Consequently, SBBC implies much stronger perturbation in I^0 than BBC.

For this analysis, we use the same set of seven benchmark instances as in Section VI. To perform the statistical analysis, we use the Welch's t-test on sets of solutions obtained after 50 runs of our multilevel memetic algorithm. Table VII-A shows the average perturbation degree expressed as a percentage of $|V|$, which is introduced by both BBC and SBBC when $p = 6$ for $k \in \{4, 8, 16, 32\}$. As expected, we observe that the number of perturbed vertices is always significantly larger with SBBC than with BBC.

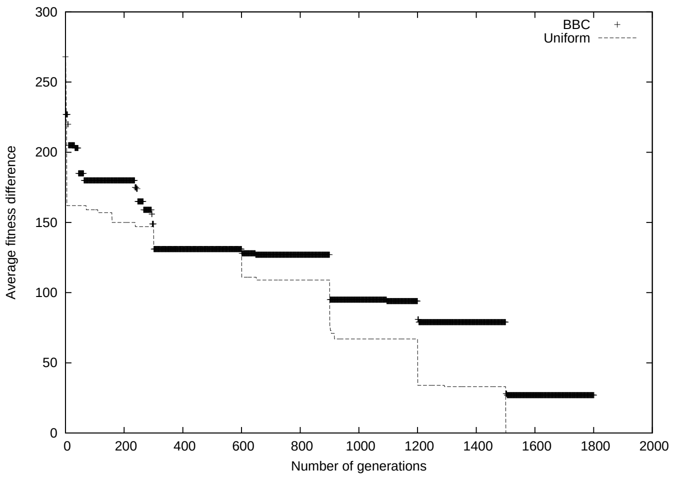
To analyze the difference in performance, we provide in Table VIII the t-value, the degree of freedom df, and the two tailed p-value over the two solution sets generated with BBC and SBBC respectively. We observe that the p-value is extremely significant (*p-value* < 0.001) except in one case, which suggests a statistically significant difference between the two solution sets. The negative t-value indicates that the mean value of partitions obtained by BBC is, except in two cases, significantly lower that the one produced by SBBC. These results imply that BBC outperforms in a more pronounced way SBBC, which suggests that a too strong perturbation introduced in offspring is not desirable. This result remains consistent with that observed when the uniform crossover is used.

VIII. INFLUENCE OF LOCAL OPTIMIZATION

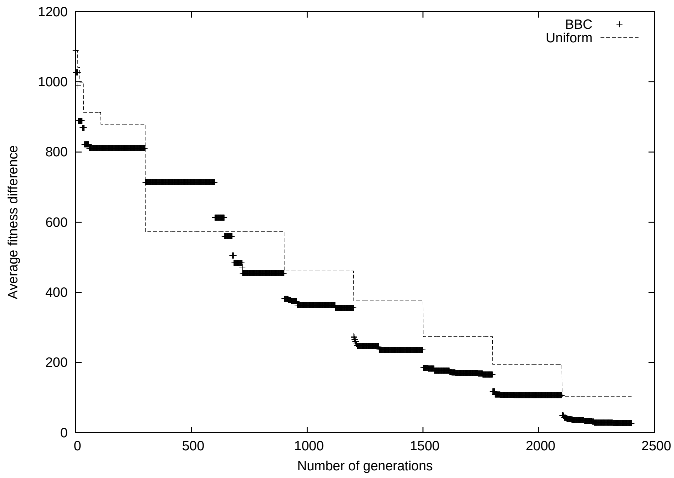
In this section, we briefly analyze the contribution of local optimization to the overall performance of the memetic algorithm by comparing the proposed perturbation-based tabu search procedure [6] with its hill-climbing version. The hill-climbing procedure is basically the same as the tabu search procedure with the tabu list disabled.

We perform a statistical analysis on seven graphs with $k \in \{4, 8, 16, 32\}$ using the Welch's t-test on solution sets obtained after 20 independent runs. For this analysis, we set the number of generations to 500, and the number of local optimization iterations before and after crossover to $0.5 * |V_i|$ and $5 * |V_i|$ respectively, where $|V_i|$ is the number of vertices at the i^{th} graph level.

Table IX shows the statistical result between solution sets generated with two versions of our algorithms, which integrate respectively the tabu search procedure (TS) and the hill climbing procedure (HC). For each $k \in \{4, 8, 16, 32\}$, columns '*t*-value', 'df' and '*p*-value' report respectively the *t*-value, degree of freedom and *p*-value over the two solution sets. In each case, the *t*-values are negative which indicates that the proposed multilevel approach always performs better when the TS algorithm is employed. From these very small *p*-values (< 0.001), one concludes that the difference is statistically significant. These analytical results, along with the FD analysis shown previously, provide clear evidence that the perturbation-based tabu search procedure plays an important role in the overall performance of our multilevel memetic algorithm.



(a) Graph 'data' when $k = 4$



(b) Graph 'fe-4elt2' when $k = 32$

Fig. 5. Evolutionary curves of the multilevel memetic algorithm with the BBC and uniform crossovers over 300 generations per graph level $G_m, G_{m-1}, ..., G_0$ for graphs 'data' with $k = 4$ and 'fe-4elt2' with $k = 32$. The *x*-axis corresponds to the current number of generations, while the *y*-axis shows the 'normalized' average fitness value over 10 independent executions. Both algorithms start with a population of the same quality.

TABLE VIII

STATISTICAL ANALYSIS USING THE WELCH'S t-TEST OVER TWO SOLUTION SETS GENERATED WITH BBC AND SBBC WITH $p = 6$ FOR k EQUAL TO 4, 8, 16 AND 32. A NEGATIVE t-VALUE MEANS THAT BBC OUTPERFORMS THE SBBC OPERATOR.

	k=4			k=8			k=16			k=32		
Graph	*t-value*	*df*	*p-value*	*t-value*	*df*	*p-value*	*t-value*	*df*	*p-value*	*t-value*	*df*	*p-value*
data	-6.547	93	0.000	-12.103	95	0.000	-18.154	91	0.000	-13.626	91	0.000
3elt	-6.314	72	0.000	-14.111	76	0.000	-9.535	93	0.000	-13.351	97	0.000
uk	-10.916	92	0.000	-8.175	98	0.000	-5.123	98	0.000	-3.716	93	0.000
crack	-4.214	81	0.000	-13.438	72	0.000	-10.535	97	0.000	-5.638	96	0.000
wing-nodal	3.539	95	0.001	3.756	94	0.000	-11.502	93	0.000	-7.492	96	0.000
fe-4elt2	-7.338	72	0.000	-13.236	70	0.000	-8.886	94	0.000	-9.757	96	0.000
vibrobox	3.390	98	0.001	-8.410	84	0.000	-5.165	92	0.000	-1.103	98	0.273

TABLE IX

STATISTICAL ANALYSIS USING THE WELCH'S t-TEST OVER SOLUTION SETS GENERATED WITH TWO VERSIONS OF OUR ALGORITHMS, WHICH INTEGRATE RESPECTIVELY THE PROPOSED PERTURBATION-BASED TABU SEARCH PROCEDURE (TS) AND A HILL CLIMBING PROCEDURE (HC). A NEGATIVE t-VALUE MEANS THAT THE ALGORITHM WITH TS PERFORMS BETTER THAN THE ALGORITHM WITH HC.

	k=4			k=8			k=16			k=32		
Graph	*t-value*	*df*	*p-value*	*t-value*	*df*	*p-value*	*t-value*	*df*	*p-value*	*t-value*	*df*	*p-value*
data	-4.440	38	0.000	-0.550	38	0.585	-2.144	38	0.039	-9.505	27	0.000
3elt	-4.712	29	0.000	-2.639	33	0.013	-1.773	34	0.085	-3.590	25	0.001
uk	-6.541	25	0.000	-4.728	32	0.000	-5.785	36	0.000	-13.136	26	0.000
crack	-11.441	23	0.000	-7.541	31	0.000	-4.428	38	0.000	-8.628	28	0.000
wing_nodal	-4.086	32	0.000	-3.919	35	0.000	-9.030	31	0.000	-10.938	26	0.000
fe_4elt2	-1.784	19	0.091	-7.541	31	0.000	-3.747	35	0.001	-7.188	28	0.000
vibrobox	-19.047	37	0.000	-8.222	21	0.000	-7.232	32	0.000	-6.693	27	0.000

IX. CONCLUSION

In this paper, we have presented a highly efficient multi-level memetic algorithm for the balanced graph partitioning problem. Our MMA algorithm uses an original backbone-based multi-parent crossover operator, a perturbation-based tabu search procedure as the local optimization engine, and a pool replacement strategy that takes into consideration both the solution quality and the distance between solutions. The backbone-based multi-parent crossover operator of MMA tries to preserve the elements which hopefully belong to the optimal partition while permitting limited perturbations within offspring solutions. The tabu search procedure (via its tabu list and occasional random moves), and the quality-and-distance based population updating strategy provide MMA with a healthy population diversity during its search.

We have proposed landscape analysis using FDC to put forward the existence of the big valley structure for some problem instances, and studied the backbone phenomenon within a set of high quality solutions which provided motivation for the design of our BBC crossover. We have also investigated the role of perturbation within the crossover operators and the influence of local optimization on the performance of the memetic algorithm.

We have assessed extensively the performance of the proposed memetic algorithm with both short and long run times, on a collection of benchmark graphs from the Graph Partitioning Archive, with the cardinal number k set to 2, 4, 8, 16, 32 and 64. We have shown that the results generated in short computing time (from less than one second to some 15 minutes) are very competitive with those produced by the two well-known graph partitioning packages METIS and CHACO. When the running time is prolonged (from several minutes to 5 hours), our approach succeeds even to improve more than two thirds of the best partitions of the given balance reported at the Graph Partitioning Archive.

This study focuses on obtaining perfectly balanced or slightly imbalanced partitions. It is known that allowing more imbalance may lead to partitions of better quality. Indeed, when we relaxed the balance constraint up to a certain degree, with a slight modification of our MMA algorithm described in this paper, we obtained imbalanced partitions (not reported here) which are highly competitive with the best-known partitions reported in the literature. However, we believe that further research needs to be realized in order to design a dedicated algorithm which could improve even more the quality of imbalanced partitions.

ACKNOWLEDGMENTS

This work was partially supported by "Angers Loire Métropole" and the Region of "Pays de la Loire" within the Miles, Radapop and LigeRO Projects.

REFERENCES

[1] C.J. Alpert and A.B. Kahng. Recent Directions in Netlist Partitioning. *Integration: the VLSI Journal*, 19(12): 1–81, 1995.

[2] A. Bahreininejad, B.H.V. Topping and A.I. Khan. Finite Element Mesh Partitioning Using Neural Networks. *Advances in Engineering Software*, 103–115, 1996.

[3] S.T. Barnard and H.D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice & Experience*, 6(2): 101–117, 1994.

[4] R. Battiti and A. Bertossi. Greedy and Prohibition-Based Heuristics for Graph Partitioning. *IEEE Transactions on Computers*, 48: 361–385, 1999.

[5] U. Benlic and J.K. Hao. An Effective Multilevel Memetic Algorithm for Balanced Graph Partitioning. *Proceedings of the 22th International Conference on Tools with Artificial Intelligence*, pages 121–128, 2010.

[6] U. Benlic and J.K. Hao. An Effective Multilevel Tabu Search Approach for Balanced Graph Partitioning. *Computers & Operations Research*, 38(7): 1066–1075, 2011.

[7] K.D. Boese. Cost versus Distance in the Traveling Salesman Problem. Technical Report TR-950018, UCLA CS Department, 1995.

[8] K.D. Boese. Models for Iterative Global Optimization. Tech. PhD thesis, University of California, Computer Science Department, Los Angeles, CA, USA, 1996.

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

[10] T.N. Bui and B.R. Moon. Genetic Algorithm and Graph Partitioning. *IEEE Transactions on Computers*, 45(7): 841–855, 1996.

[11] P. Chardaire, M. Barake, and G. P. McKeown. A PROBE based heuristic for Graph Partitioning. *IEEE Transactions on Computers*, 56: 1707–1720, 2007.

[12] M. Dell'Amico and F. Maffioli. A Tabu Search Approach to the 0-1 Equicut Problem. *Meta Heuristics 1995: The State of the Art, Kluwer Academic Publishers*, pages 361–377, 1996.

[13] O. Dubois and G. Dequen. A Backbone-Search Heuristic for Efficient Solving of Hard 3-SAT Formulae. *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 248–253, 2001.

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

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

[16] C. Gil, J. Ortega, A.F. Diaz and M.G. Montoya. Annealing-Based Heuristics and Genetic Algorithms for Circuit Partitioning in Parallel Test Generation. *Future Generation Computer Systems*, 14(5): 439–451, 1998.

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

[18] B. Hendrickson and R. Leland. The Chaco User's Guide Version 2.0. Technical Report, Sandia National Laboratories, 1995.

[19] B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In *S. Karin. (Eds) Proceedings of Supercomputing '95, San Diego*, ACM Press, New York, 1995. http://www.supercomp.org/sc95/proceedings/509_BHEN/SC95.HTM

[20] M. Holtgrewe, P. Sanders, and C. Schulz. Engineering a Scalable High Quality Graph Partitioner. Technical Report, 2009.

[21] D.S. Johnson, C.R. Aragon, L.A. Mcgeoch and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation; Part-I, Graph Partitioning. *Operations Research*, 37: 865–892, 1989.

[22] T. Jones and S. Forrest. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. *Proceedings of the 6th International Conference on Genetic Algorithms*, Morgan Kaufmann, pages 184–192, 1995.

[23] S. Kang and B.R. Moon. A Hybrid Genetic Algorithm for Multiway Graph Partitioning. *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 159–166, Morgan Kaufmann, 2000.

[24] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1): 359–392, 1998.

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

[26] P. Kilby and J. Slaney. Backbones and Backdoors in Satisfiability. *Proceedings of the 20 th American Association for Artificial Intelligence*, pages 1368–1373, 2005.

[27] J.P. Kim and B.R. Moon. A Hybrid Genetic Search for Multi-way Graph Partitioning Based on Direct Partitioning. *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 408–415, 2001.

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

[29] N. Mansour and G.C. Fox. Allocating Data to Distributed-memory Multiprocessors by Genetic Algorithms. *Concurrency: Practice & Experience*, 6(6): 485–504, 1994.

[30] P. Merz and B. Freisleben. Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning. *Evolutionary Computation*, 8(1): 61–91, 2000.

[31] H. Meyerhenke, B. Monien, and T. Sauerwald. A New Diffusion-based Multilevel Algorithm for Computing Graph Partitions of Very High Quality. *Journal of Parallel and Distributed Computing*, 69(9): 750–761, 2009.

[32] C. Papadimitriou and K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, 1982.

[33] F. Pellegrini. Scotch and libScotch 5.1 Users Guide, INRIA, University of Bordeaux I, 2010.

[34] 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.

[35] C.R. Reeves. Landscapes, Operators and Heuristic Search. *Annals of Operations Research*, 86: 473–490, 1997.

[36] E. Rolland, H. Pirkul and F. Glover. Tabu Search for Graph Partitioning. *Annals of Operations Research*, 63: 209–232, 1996.

[37] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 731–737, 1997.

[38] A.J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph-partitioning. *Journal of Global Optimization*, 29(2): 225–241, 2004.

[39] A. Slowik and M. Bialko. Partitioning of VLSI Circuits on Subcircuits with Minimal Number of Connections Using Evolutionary Algorithm. In L. Rutkowski et al. (Eds.), ICAISC 2006, LNAI 4029, pages 470–478, Springer-Verlag, 2006.

[40] A.G. Steenbeek, E. Marchiori, and A.E. Eiben. Finding balanced graph bi-partitions using a hybrid genetic algorithm. *1998 IEEE International Conference on Evolutionary Computation (ICEC98)*, pages 90–95, 1998.

[41] L. Sun and M. Leng. An Effective Refinement Algorithm Based on Swarm Intelligence for Graph Bipartitioning. *International Symposium*

on Combinatorics, Algorithms, Probabilistic and Experimental Methodologie, LNCS/LNAI, pages 1–12, Springer, Heidelberg, 2007.

[42] E. Talbi and P. Bessiere. A Parallel Genetic Algorithm for the Graph Partitioning Problem. *Proceedings of the 5th International Conference on Supercomputing*, ACM Press, New York, pages 312–320, 1991.

[43] G. von Laszewski. Intelligent Structural Operators for the k-Way Graph Partitioning Problem. *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 45–52, 1991.

[44] C. Walshaw and M. Cross. Mesh Partitioning : A Multilevel Balancing and Refinement Algorithm. *SIAM Journal on Scientific Computing*, 22(1): 63–80, 2000.

[45] C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. *Annals of Operations Research*, 131: 325–372, 2004.

[46] Y. Wang, Z. Lü, F. Glover, J.K. Hao. Backbone Guided Tabu Search for Solving the UBQP Problem, *Journal of Heuristics* (In Press), 2011. DOI:10.1007/s10732-011-9164-4

[47] H. Zha, X. He, C. Ding, H. Simon, M. Gu. Bipartite Graph Partitioning and Data Clustering. *Proceedings of the ACM 10th International Conference on Information and Knowledge*, pages 25–31, 2001.

[48] Q. Zhang, J. Sun, E.P.K. Tsang. Evolutionary Algorithm with the Guided Mutation for the Maximum Clique Problem, *IEEE Transactions on Evolutionary Computation*, 9(2), pp. 191–201, 2005.