Evolutionary Algorithms for Overlapping Correlation Clustering

Carlos E. Andrade
University of Campinas
Av. Albert Einstein, 1251
Campinas, SP, Brazil
andrade@ic.unicamp.br

Mauricio G. C. Resende
AT&T Labs Research
200 Laurel Avenue
Middletown, NJ 07748 USA
mgcr@research.att.com

Howard J. Karloff
Yahoo Labs
111 West 40th Street
New York, NY 10018 USA
karloff@yahoo-inc.com

Flávio K. Miyazawa
University of Campinas
Av. Albert Einstein, 1251
Campinas, SP, Brazil
fkm@ic.unicamp.br

ABSTRACT

In Overlapping Correlation Clustering (OCC), a number of objects are assigned to clusters. Two objects in the same cluster have correlated characteristics. As opposed to traditional clustering where objects are assigned to a single cluster, in OCC objects may be assigned to one or more clusters. In this paper, we present Biased Random-Key Genetic Algorithms for OCC. We present computational experiments such results outperformed the state of art methods for OCC.

Categories and Subject Descriptors

I.2.8 [Artificial Intelligence] [**Problem Solving, Control Methods, and Search**]: [Heuristic methods]

Keywords

Data mining; Correlation Clustering; Biased Random-Key Genetic Algorithm

1. INTRODUCTION

One of the most fundamental tasks in data analysis is to categorize objects in different sets such that two objects in the same set have certain characteristics that are correlated. This correlation is usually measured by a similarity value. The standard way of clustering is the creation of a partition of these objects. However, for some applications, it is natural that an object belong to two or more clusters since it can share characteristics with objects in more than one cluster. In this case, to partition the ground set does not make sense. Instead, it is more appropriate to assign the objects to clusters with possible overlapping. Several scenarios with these properties can be addressed: in social networks, a user belongs to several communities; in mobility analysis, persons share trajectories with respect to time and space; in biology, a protein belongs to several protein complexes having similar expressions.

We can model scenarios like these using Overlapping Correlation Clustering (OCC), introduced in [5]. This problem is closely related to Correlation Clustering (CC) [2] but allows overlapping of the clusters. Another major difference is in the relation among the objects. In OCC, this relation is expressed as a value in range $[0, 1]$ while in CC it takes on one of two discrete values in the set $\{0, 1\}$. This enables the utilization of different types of similarity functions leading to a more sophisticated analysis.

This paper presents Biased Random-Key Genetic Algorithms associated with local search procedures aiming to solve the overlapping correlation clustering problem under an optimization point of view. Section 2 formalizes the problem. Section 3 presents high level algorithms to deal with OCC. Section 4 depicts the experimental results and Section 5 presents the concluding remarks.

2. DEFINITIONS

Let $V = \{v_1, \ldots, v_n\}$ be a set of n objects such that there exists a symmetric function $s : V \times V \to [0, 1]$ which gives the similarity $s(u, v)$ between two objects u and v in V. Let $L = \{1, \ldots, k\}$ be the set of k available labels (clusters). Since we allow an object to be in several clusters, we define $\ell : V \to 2^L \setminus \emptyset$ with $\ell(v)$ being the set of labels assigned to object v. Note that we require that each object have at least one label. If the set $\ell(v) = \{c_1, \ldots, c_s\}$ of labels is given to object v, then we assume that v is in clusters $c_1, \ldots, c_s$. Consider also $\mathcal{H} : 2^L \times 2^L \to [0, 1]$, a symmetric function which gives the similarity between two sets of labels. Higher "similarity" of E and F should correspond to higher $\mathcal{H}(E, F)$, and $\mathcal{H}(E, E)$ should be 1 for all E. In Overlapping Correlation Clustering (OCC), the task is to find a multi-labeling function ℓ that minimizes the cost function

$$\frac{1}{2} \sum_{(u,v) \in V \times V, u \neq v} |\mathcal{H}(\ell(u), \ell(v)) - s(u, v)|. \qquad (1)$$

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2576768.2598284.

Note that Objective Function (1) represents the absolute error between the labeling and the similarity measure. In this sense, the objective is to find a labeling as close as possible to the given similarities.

This formulation enables the application in different contexts. However, we first need to measure the similarities that are context dependent. As the problem is general enough, we can use any type of comparison measure as long as it scales to the real interval $[0, 1]$. Another considerable task is to find the appropriate $\mathcal{H}$ function. As suggested in [5], we use two functions: The *Jaccard Similarity Coefficient* and the *Set-Intersection Indicator*. The Jaccard similarity coefficient, also known as the Jaccard index, is a well-known measure of similarity between two sample sets, widely used in biology and machine learning. Let E and F be two sets, not both empty. The Jaccard index of E and F is defined as $\mathcal{J}(E, F) = \frac{|E \cap F|}{|E \cup F|}$. The set-intersection indicator function is a simple function defined as $\mathcal{I}(E, F) = 1$ if $E \cap F \neq \emptyset$ or $\mathcal{I}(E, F) = 0$, otherwise.

Some scenarios require that objects be assigned a restricted number of labels among those available. A good example is in the analysis of mobility patterns, where in spite of the fact that trajectories can have a large number of characteristics, we restrict ourselves to only a few of the most meaningful ones. In these cases, we introduce a parameter p such that $|\ell(v)| \leq p$ for all $v \in V$.

It is easy to see that, if we consider $\mathcal{H} = \mathcal{I}$ and $p = 1$, each object will belong to a single cluster and the similarity between pairs of objects will be an indication as to whether they share the same cluster. In this case, if s follows the set-intersection function, we face the original correlation clustering problem, which is $\mathcal{NP}$-hard [2]. In [5], hardness proofs for the other cases are presented.

3. BIASED RANDOM KEY GENETIC ALGORITHM AND LOCAL SEARCH

To solve *Overlapping Correlation Clustering*, we opt to use a Biased Random-Key Genetic Algorithm (BRKGA) mainly because of its recent success in solving several problems, such as routing [1], packing [8], among others. The major features that distinguish BRKGAs from other genetic algorithms is the standard chromosome encoding and a well-defined and parameterized evolutionary process. The biased component of a BRKGA contributes to increasing the likelihood that high-quality solutions will pass down their genes to future generations. The decoding process links the genetic algorithm with the problem itself, indirectly mapping the chromosome space $[0, 1]^n$ onto the set of feasible solutions. See [8] for more details regarding BRKGAs.

3.1 Representation

In most BRKGA implementations, a chromosome is encoded as a real vector $\mathbf{x}' \in [0, 1]^n$, following the procedure given in [4]. However, other encodings are possible. In this paper, we use two forms of chromosomes to represent a labeling. Let n be the number of objects and k the number of available clusters. The first representation, called *compact*, is an integer vector $\mathbf{x^c} \in \mathbb{N}^n$, and the second, called *extended*, is a binary vector $\mathbf{x^e} \in \{0, 1\}^{nk}$.

For the compact chromosome $\mathbf{x^c}$, each $\mathbf{x_i^c}$ is a positive integer that represents the clusters to which object v_i belongs. In this case, we consider each bit as a set indicator in which the least-significant bit corresponds to the first cluster, and so on. Although this representation limits the number of possible clusters ($k < 64$ on a 64-bit machine)[1], it enables very fast set operations since they are done bitwise, and for most applications, this limit is sufficient. In the extended chromosome $\mathbf{x^e}$, the subvector $\tilde{\mathbf{x}}^\mathbf{e} = \tilde{\mathbf{x}}^\mathbf{e}_{(i-1)k+1}, \ldots, \tilde{\mathbf{x}}^\mathbf{e}_{ik}$ is an indicator vector where, for each object v_i, $\tilde{\mathbf{x}}^\mathbf{e}_j$ is 1 if v_i is in cluster j, or 0 otherwise. This representation can be used with any $n > 1$ and $k > 1$.

Although both compact and extended representations are quite similar, they affect the evolutionary mechanism of BRKGA differently. In the compact representation, each allele is the full labeling of an object and therefore, in the mating process, the offspring inherits these full labelings. Learning occurs through the combination of entire sets of labels of each object. In the extended representation, each allele represents a specific label of an object. In the mating process, the offspring inherits each label individually, enabling learning to occur at the level of each label for each object.

3.2 Decoding a solution

To decode a solution from a chromosome using the representations of Section 3.1, we use a two-phase decoder in which the first phase extracts a solution and the second phase is committed to local search procedures. We assume that the input vector to the decoder is in compact representation. Note that in case of the extended representation, we need to "compactify" the vector obtaining from each subvector $\tilde{\mathbf{x}}^\mathbf{e} = \tilde{\mathbf{x}}^\mathbf{e}_{(i-1)k+1}, \ldots, \tilde{\mathbf{x}}^\mathbf{e}_{ik}$, for each object v_i, the integer that can be described by this sequence of bits. Algorithm 1 describes the first phase.

First of all, we must guarantee that the number of clusters of each object is limited to p. If the number $|\ell(v)|$ of clusters containing object v is greater than p, we repair the chromosome by removing $|\ell(v)| - p$ clusters from v uniformly at random. Lines 1–2 of Algorithm 1 summarize this procedure, in which by "$s \in \mathbf{x}$" we mean that "$s = x_h$ for some h," and by "$|s|$," the number of bits equal to one in the binary representation of s.

After the repair, we have a feasible solution whose value is computed in lines 4–11. We compare each pair of objects (u, v), calculating the Jaccard or set-intersection similarity and adding the error to the solution value, according to Equation (1). Note that in line 5 we abuse notation: $l(u) \leftarrow \mathbf{x}_u$ gets the labeling of object u. We also split the amount of error caused by u and v into two parts: the positive error e^+, indicating that labels assigned to u and v are too similar, and the negative error e^-, indicating otherwise. These errors are used in the local search phase to improve the solution. Note that vectors $e^+, e^- \in \mathbb{R}^+$. After local search, the solution may be improved and, in such case, we consolidate the change of the solution in the chromosome (line 13).

The running time complexity of the first phase is $O(kn^2)$ when k is unrestricted. Note that the repair phase runs in $O(kn)$ time. The solution value computation has a quadratic factor due to the comparison of each pair of objects. The time complexity of function $\mathcal{H}$ depends directly on k and the type of function. Assuming k is unrestricted, when $\mathcal{H} = \mathcal{J}$,

[1]If a specialized bitset structure like C++'s `bitset<>` template is used, this limitation can be overcome although at the expense of increased complexity.

Algorithm 1: Decoder – Phase 1

Input: a vector $\mathbf{x} \in \mathbb{N}^n$, integers $k, p \in \mathbb{N}$, and set V.
Output: modified $\mathbf{x}$ and the fitness $\mathcal{C}$ of $\mathbf{x}$.

foreach $s \in \mathbf{x}$ *such that* $|s| > p$ **do**
    Remove $|s| - p$ elements from s uniformly at random;
$\mathcal{C} \leftarrow 0$;
foreach *pair* $(u, v) \in V \times V$ **do**
    $l(u) \leftarrow \mathbf{x}_u$; $l(v) \leftarrow \mathbf{x}_v$;
    $error \leftarrow \mathcal{H}(\ell(u), \ell(v)) - s(u, v)$;
    $\mathcal{C} \leftarrow \mathcal{C} +$ `absval`$(error)$;
    if $error > 0$ **then**
        $e_u^+ \leftarrow e_u^+ + error$; $e_v^+ \leftarrow e_v^+ + error$;
    else
        $e_u^- \leftarrow e_u^- - error$; $e_v^- \leftarrow e_v^- - error$;
`LocalSearch()`;
Rewrite the improved solution on the chromosome $\mathbf{x}$;
return *modified* $\mathbf{x}$ *and solution value* $\mathcal{C}$;

we need to compute the union and intersection of the two sets of labels, which takes $O(k)$ time each. In the case of $\mathcal{H} = \mathcal{I}$, we only need to calculate the intersection, which takes $O(k)$ time. However as observed in the previous section, for most practical applications, $k < 64$, which enables the representation using a 64-bit integer. The intersection can be implemented using a single machine `AND` operation, the union using a single machine `OR` operation, and the bit counting running in $O(1)$ using a lookup table[2]. Using this implementation, the time complexity of phase 1 is $O(n^2)$, if $k < 64$.

In the second phase, a local search procedure is applied to the solution found in phase 1. We developed a local search that explores neighborhood solutions based on error reduction. This procedure is detailed on Section 3.3. We also use the local search methods proposed in [5]. They developed two local searches, one for OCC using the Jaccard index as the $\mathcal{H}$ function and another using the set-intersection function. Section 3.4 details these algorithms.

3.3 Error Reduction Local Search

Algorithm 2 depicts the local search procedure based on error reduction, which we call `OLS`. In line 4, we sort the objects v in nonincreasing order of $e_v^+ + e_v^-$, such that we start with the object whose labeling leads to the most error in the entire clustering process. For the first given $\tau \leq n$ objects in this order, `OLS` attempts to reduce or augment the total similarity driven by the values of e^+ and e^-. This is done by removing or adding labels that impact total similarity. For a given object v, if $e_v^+ \geq e_v^-$, we try to find a most common label between v and all other objects, and remove it from v (line 16). Otherwise, if $e_v^+ < e_v^-$, we try to add to v a most common label that was not assigned to it (line 18). If none of these *add* or *remove* operations can be done, we remove a label from v uniformly at random and replace it with a label not assigned to v, also chosen uniformly at random (line 10–14). In this case, in each exchange we only select a new label not previously chosen in the previous iterations. In the worst case, we try all k labels. If an improvement is

[2]Another approach is to use the special hardware instruction `POPCNT` found in modern processors. For details, see [10].

Algorithm 2: Error Reduction Local Search – `OLS`

Input: labeling function ℓ; vectors $e^+, e^- \in \mathbb{R}^+$; integers $p, \tau \in \mathbb{N}$; set V.

1 Let $\mathcal{C}$ be the cost of ℓ computed with Obj. Func. (1);
2 *impr* $\leftarrow$ **True**;
3 **while** *impr* = **True do**
4 Let V' be the set of objects $v \in V$, sorted in nonincreasing order of $e_v^+ + e_v^-$;
5 *impr* $\leftarrow$ **False**; $i \leftarrow 0$;
6 **while** $i \leq \tau$ **do**
7 Take the next $v \in V'$ in the given order;
8 $\hat{\ell}(v) \leftarrow \ell(v)$;
9 *local_impr* $\leftarrow$ **True**;
10 **while** *local_impr* = **True do**
11 $\hat{\mathcal{C}} \leftarrow \mathcal{C}$;
12 *local_impr* $\leftarrow$ **False**;
13 **if** $e_v^+ \geq e_v^-$ **and** $|\ell(v)| = 1$ **then**
14 Exchange the unique label $c \in \ell(v)$ for a $c' \notin \ell(v)$ chosen uniformly at random;
15 **if** $e_v^+ \geq e_v^-$ **and** $|\ell(v)| > 1$ **then**
16 Remove from $\ell(v)$ the $c \in \ell(v)$ which corresponds to the largest cluster containing v;
17 **if** $e_v^+ < e_v^-$ **and** $|\ell(v)| < p$ **then**
18 Add to $\ell(v)$ the c which corresponds to the largest cluster not containing v;
19 **foreach** $u \in V \setminus \{v\}$ **do**
20 $\delta \leftarrow \mathcal{H}(\ell(u), \hat{\ell}(v)) - \mathcal{H}(\ell(u), \ell(v))$;
21 **if** $\delta > 0$ **then**
22 $e^+(v) \leftarrow e^+(v) + \delta$; $e^+(u) \leftarrow e^+(u) + \delta$;
23 **else**
24 $e^-(v) \leftarrow e^-(v) + \delta$; $e^-(u) \leftarrow e^-(u) + \delta$;
25 $\mathcal{C} \leftarrow \mathcal{C} + \delta$;
26 **if** $\mathcal{C} \leq \hat{\mathcal{C}}$ **then**
27 *impr* $\leftarrow$ **True**; *local_impr* $\leftarrow$ **True**;
28 **else**
29 $\mathcal{C} \leftarrow \hat{\mathcal{C}}$;
30 **if** *local_impr* = **True then** go to line 3; **else** $\ell(v) \leftarrow \hat{\ell}(v)$; i**++**;
31 **return** *labeling* ℓ *and solution value* $\mathcal{C}$;

reached, the solution value and the errors of each object are updated (lines 19–25) and a new iteration begins. After τ iterations without improvement, the local search stops.

The time complexity of `OLS` is tricky to compute since it depends on the starting solution. The innermost loop (starting in line 10) iterates at most k times since each label is added or removed only once. Note that adding (respectively, removing) a label that was removed (respectively, added) in earlier iterations (of the loop starting on line 10) does not lead to an improvement in solution quality. The exchange on line 14 can be done in $O(1)$ time. The operations in lines 16 and 18 can be done in $O(\log k)$ time using a red-black tree [3] data structure for the histogram of labels that we use to keep track of the most used labels. However, note that for small values of k, it is worthwhile to use a naive linear approach running in time $O(k)$. The update operations in lines 19–25 take $O(n)$ time since we need to recalculate the function $\mathcal{H}$ for all pairs (v, w) with fixed v and $w \neq u \in V$. Therefore, the loop starting on line 6 has time complexity

$O(\tau(k \log k + kn))$. If $k < 64$ and $\tau \leq n$, then the time complexity is $O(n^2)$. As observed earlier, it is hard to estimate the complexity of the loop starting on line 3, but its main components are $O(n \log n)$ from line 4 and the complexity of the loop starting on line 6. Therefore, each iteration has time complexity $O(n^2)$.

3.4 Bonchi et al. Local Search

Bonchi et al. [5] propose two local search algorithms to deal with OCC using the Jaccard index and set-intersection functions. Their framework is based on the relabeling of objects, one at time, by solving a system of linear equations in the case of Jaccard, and by applying a simple greedy algorithm in the case of set intersection. The main idea is to find a good labeling of a single object given a fixed labeling of the other objects.

Let v be an object and ℓ be a labeling for all objects. The error incurred by v is defined as

$$C_v(\ell(v)|\ell) = \sum_{u \in V \setminus \{v\}} |\mathcal{H}(\ell(v), \ell(u)) - s(v,u)|, \qquad (2)$$

and, consequently, the objective function (1) can be rewritten as

$$\frac{1}{2} \sum_{v \in V} C_v(\ell(v)|\ell). \qquad (3)$$

Using these observations, Bonchi et al. proposed Algorithm 3, which is a simple local search algorithm.

In the case of the Jaccard index, the following approach is used. Let v be an object to be relabeled and ℓ be a fixed labeling for all $u \in V \setminus \{v\}$. Let x^v be an indicator vector such that $x^v_j = 1$ if label j is assigned to object v, and $x^v_j = 0$, otherwise. Assume that the number of labels assigned to v is t, that is,

$$\sum_{j \in L} x^v_j = t. \qquad (4)$$

Ideally we would like to have $\mathcal{J}(\ell(v), \ell(u)) = s(v,u)$ for all $u \in V \setminus \{v\}$. This can be written as

$$\mathcal{J}(\ell(v), \ell(u)) = \frac{\sum_{j \in \ell(u)} x^v_j}{|\ell(u)| + t - \sum_{j \in \ell(u)} x^v_j} = s(v,u),$$

which is equivalent to

$$(1 + s(v,u)) \sum_{j \in \ell(u)} x^v_j - s(v,u)t = s(v,u)|\ell(u)| \qquad (5)$$

for all $u \in V \setminus \{v\}$. Although Equations (4) and (5) are linear with respect to the unknowns x^v_j and t, these variables are integral and therefore the system seemingly cannot be solved in polynomial time. In [5], the authors applied a nonnegative least-squares optimization method which led to possibly fractional x^v and t values. They then sort x^v in nonincreasing order, breaking ties arbitrarily. Let π^v be the permutation of labels induced by this order and consider the p sets $\{\pi^v_1, \ldots, \pi^v_i\}$ for $i = 1, \ldots, p$. The new set of labels for v is the set $\{\pi^v_1, ..., \pi^v_i\}$ that minimizes Equation (2).

For the set-intersection indicator, Bonchi et al. present a greedy approach which starts from an empty labeling for a given object and fixes the labeling for the other objects as is done in the approach for the Jaccard index. In each iteration, the label that causes the least error while improving the solution value is chosen. If such a label cannot be found, the algorithm stops the search for this object and goes on to the next. In fact, this greedy search solves $C_v(\ell(v)|\ell)$ in line 6 of Algorithm 3.

Algorithm 3: Bonchi et al. Local Search – `BSL`

Input: labeling function ℓ; set V.

```
1  Let C be the cost of ℓ computed with Obj. Func. (1);
2  improvement ← True;
3  while improvement = True do
4      improvement ← False;
5      foreach v ∈ V do
6          Find the labeling L that minimizes C_v(L|ℓ);
7          Let Ĉ be the cost of (L|ℓ);
8          if Ĉ < C then
9              ℓ(v) ← L; Ĉ ← C;
10             improvement ← True;
11 return labeling ℓ and solution_value;
```

As in Algorithm 2, the time complexity of Algorithm 3 is tricky to determine because of its dependency on the starting solution. For the Jaccard function, the dominant factor is the nonnegative least squares that can be computed in $O(m^3)$ time, where m is the major dimension of the matrix [11]. Note that the matrix resulting from Equation (5) has dimension $n \times (k+1)$. As observed earlier, if $k < n$, then the complexity of an iteration of the main loop (starting on line 3) is $O(n^4)$. For the set-intersection function, each label search takes $O(k^2)$ time, which is done at most p times. As this operation is performed for each object, each iteration of the main loop takes $O(nk^3)$ time.

4. EXPERIMENTAL RESULTS

4.1 Instances

We used several datasets, grouped in two major categories, to evaluate our algorithms. The first category has instances with known multi-label assignments. For this case, we computed the similarities between the objects using the Jaccard index. These datasets were used to check the quality of the labelings produced by the algorithms when compared to the actual labelings. We used two datasets in this category. The first, named EMOTIONS, corresponds to psychological trials of people listening to music [17]. There are 593 objects (trials) and six available labels. The second dataset, named YEAST, is formed by micro-array expression data and phylogenetic profiles with 2417 genes in a learning set for which 14 functional classes (labels) are assigned [7].

The second category is instances with unknown multi-labelings. The first dataset in this category corresponds to animal trajectories from the Starkey Project [15][3]. We used an instance with 88 trajectories of elk, mule deer, and cattle, and classify them using five labels. To calculate the similarity between each trajectory, we used the approach presented in [6], which defines the *Edit Distance in Real Sequences* (EDR).[4] The similarity between two trajectories u and v is given by $s(u,v) = 1 - EDR(u,v)$.

The second group of instances in this category is from the field of biology. They consist of homologous groups of proteins from the SCOP taxonomy [13][5]. This taxonomy is a hand-made tree classification of functional proteins. We tested four databases with 669, 587, 567, and 654 proteins

[3]Available at `http://www.fs.fed.us/pnw/starkey`.

[4]Please, refer to the supplementary material for a comprehensive description.

[5]Available at `http://scop.mrc-lmb.cam.ac.uk/scop`.

(objects) for which we assigned at most with five labels. The similarities were calculated as follows: For a node u in the protein tree, let $d(u)$ be the depth of u in the tree. If u is the root, $d(u) = 0$. For any pair of nodes u and v, let $lca(u, v)$ denote the lowest common ancestor of u and v. Let V be the set of leaves of the classification tree. The similarity between different objects $u \in V$ and $v \in V$ is defined as

$$s(u, v) = \frac{d(lca(u, v))}{\max(d(u), d(v)) - 1}.$$

The third group in this category consists of 1,000 newsgroup messages spread over 20 newsgroups [14]. The similarities are calculated using a base dictionary of the 500 most common words excluding stop words and common names but keeping political and religious references. For each message, we construct a characteristic vector to reflect the *term frequency – inverse document frequency (TF-IDF)* of each word in the dictionary [12]. Using these vectors, we applied a radial basis function to obtain the similarities $s(u, v) = e^{||\mathbf{u}-\mathbf{v}||^2}$ where $\mathbf{u}$ and $\mathbf{v}$ are the characteristic vectors of messages u and v, respectively.

4.2 Evaluated algorithms

Using the two representations of Section 3.1 and decoders from Sections 3.3 and 3.4, we consider the following variations of BRKGAs:

- `OLS-Comp`: BRKGA using the compact representation and local search from Section 3.3;
- `OLS-Ext`: Same as above but using the extended representation;
- `BLS-Comp`: BRKGA using the compact representation and Bochi et al. local search from Section 3.4;
- `BLS-Ext`: Same as above but using the extended representation;

We also tested the algorithms from [5]. Originally, each run of those algorithms starts with a simple random vector and ends when Algorithm 3 cannot find improvements. To fortify these algorithms, we built multi-start approaches around them, allowing each run to take several iterations, each starting with a different random vector. We always keep the best solution and the algorithms stop when a stopping criterion is reached. These modifications have a great impact on the original algorithms in terms of solution quality. As expected, the multi-start approaches outperformed the original algorithms. Consequently, we use them in this paper but still refer to them as `Bonchi`.

4.3 Computational environment and parameters

The experiments were conducted on identical machines with two 6-core Intel Xeon 2.4 GHz CPUs (two threads per core) and 32 GBytes of RAM running GNU/Linux. Running times reported are UNIX real wall-clock times in seconds, excluding the effort to read the instance. The algorithms are implemented in C++ and we use the GNU `g++` compiler version 4.8. Random numbers were generated by an implementation of the Mersenne Twister.For the nonnegative least-squares method, we use the Householder rank-revealing QR decomposition [11] provided by the Eigen library [9].

For the BRKGAs, we set the parameters following the advise of [8]. The population size was set to $p = 500$, the elite size to $p_e = \lceil 0.30p \rceil$, and the number of mutants to $p_m = \lfloor 0.15p \rfloor$. The probability of inheriting each allele from the elite parent was $\rho_e = 0.70$. We used the island model [16] with three independent and concurrent populations where every 100 generations each population exports its two best solutions to the other populations. After 300 generations without improvement, all populations are reset to vectors of random keys. We set $\tau = 2\sqrt{n}$ after performing some short tuning trials and analyze the quality of solutions as a function of optimization time. We use 12 simultaneous cores for decoding.

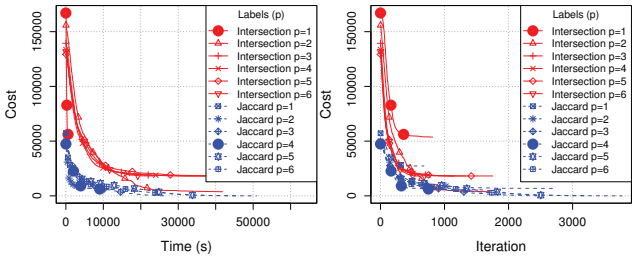
Thirty independent runs were performed for all five algorithms. Each algorithm ran for a given maximum amount of time (instance dependent) or at for most 1,000 generations (or iterations) without improvement of the best solution.

4.4 Defining maximum running times for BRKGA

Since we have several types of problems, a significant variation in BRKGA running times is expected on each problem. We carried out some preliminary experiments and observed that the BRKGAs obtained a large number of small improvements in solution quality. In fact, in the first hour of optimization, the BRKGAs improved the solution about every two or three generations. In light of this, we performed one long run for an instance of each scenario. For these long runs we did not set a time limit but rather stopped after 1,000 generations without improvement of the best solution. As we only did one long run per scenario, we cannot draw any statistically significant conclusion about comparisons of the BRKGAs with the different $\mathcal{H}$ functions. We limit ourselves to observing the general behavior of these BRKGAs.

Figure 1 depicts the convergence of the cost value as a function of time and iterations for the Emotions dataset. The Y-axis shows the cost value. The X-axis represents the time in seconds in Figure 1a and the number of iterations in Figure 1b. The solid red lines show the convergence for the BRKGA with the set-intersection function for values of p from one to six. The dashed blue line shows the same but for the Jaccard index function. We note that the BRKGA takes a long time with small improvements in the best solution and, after 30,000 seconds (about 8.3 hours), the improvements are even rarer. For the BRKGA with the Jaccard index function, values very close to zero, a lower bound on the optimum, are reached. Using the set-intersection indicator, the results are worse but the convergence behavior is similar. Looking at the number of iterations, one can note that the BRKGA with set-intersection function converges faster than it does with respect to time. We can conclude, therefore, that the local search spends more time using this function.

For other instances, the curves have the same behavior and they can be found in the supplementary material. For the Yeast dataset, running times are much larger than those on Emotions. In this case, the BRKGA with the set-intersection function has slower convergence than it does with the Jaccard index but is able to obtain improvements even after running for 1,000,000 seconds (about 11 days). For the Starkey project dataset, the running times are very small for both $\mathcal{H}$ functions, mainly due to the size of the problem (88 objects). One notes that the BRKGA with set intersection is much worse than with the Jaccard index (except for the case of $p = 2$, in which they obtained similar



(a) (b)

Figure 1: Evolution of the cost for the EMOTIONS dataset.

results). For the SCOP dataset 1, we observe different behavior: although the convergence is similar, running times are smaller than for the previous scenarios and, more importantly, the BRKGA with set intersection shows poor results when compared to EMOTIONS and YEAST. Later in this paper, we discuss these results in more detail. For newsgroup messages, we omitted the curves for the BRKGA with set intersection since it obtained results 20 times worse than with the Jaccard index but with convergence about 400 times faster. This may indicate that set intersection is not appropriate for clustering in this type of instance.

From these experiments, we can estimate the running times for each scenario and set time bounds for the experiments to follow. For EMOTIONS and YEAST, we limit the experiments to at most eight hours (28,800 seconds) trying to balance running time and solution quality. This limit is less favorable to the BRKGA with set intersection which still finds solution improvements after 15 days (1,250,000 seconds). For Starkey and SCOP, we set the maximum running time to one hour (3,600 seconds). For newsgroup messages, the maximum running time is set to seven days (604,800 seconds).

4.5 Evaluating the quality of the algorithms on ground-truth instances

To evaluate the quality of the algorithms, we first applied them on instances in which we know the actual labeling *a priori*. We compare the costs of the final solutions produced by the algorithms and also make use of two metrics, precision and recall. Define $P(x) = \{\{u, v\}, u \neq v : x(u) \cap x(v) \neq \emptyset\}$, the set of unordered pairs of objects with at least one common label in x. Let g be the labeling of the ground truth. The *precision* of a labeling ℓ is

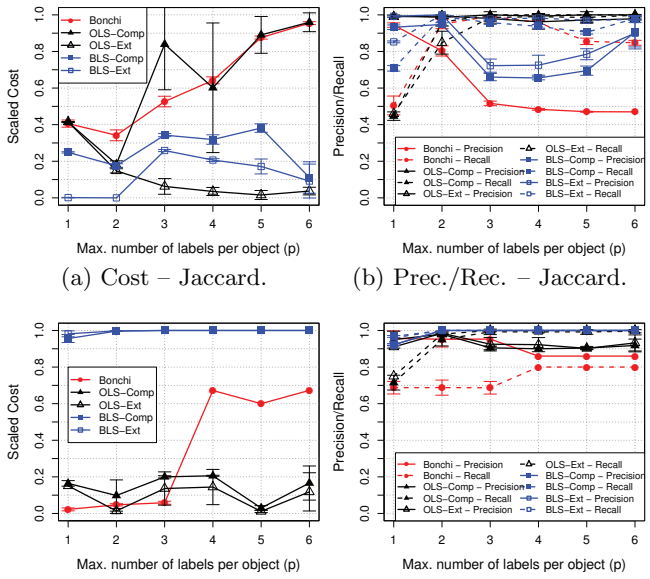
$$Precision_g(\ell) = \frac{|P(\ell) \cap P(g)|}{|P(\ell)|}. \quad (6)$$

The *recall* of a labeling ℓ is

$$Recall_g(\ell) = \frac{|P(\ell) \cap P(g)|}{|P(g)|}. \quad (7)$$

Note that for $\ell = g$, $Precision_g(\ell) = Recall_g(\ell) = 1$.

Figure 2 shows the performance of the algorithms for the EMOTIONS dataset. The X-axis on all plots corresponds to the maximum number of allowed labels per object. For Figures 2a and 2c, the Y-axis corresponds to the scaled solution costs (wth lower being better). In the plots, the red line with circles shows results for `Bonchi`, the black lines with triangles for the BRKGAs with the `OLS` decoder, while the blue lines with squares show results for the BRKGAs



(a) Cost – Jaccard. (b) Prec./Rec. – Jaccard.

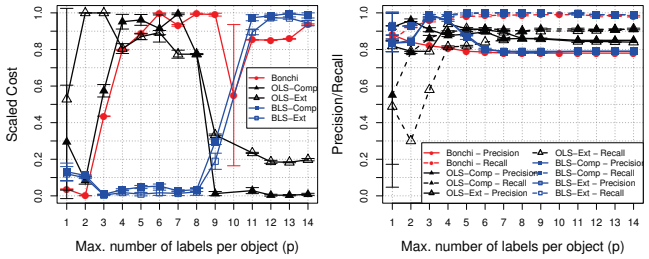
(c) Cost – Set-intersec. (d) Prec./Rec. – Set-intersec.

Figure 2: Comparison of costs, precision, and recall among the algorithms for the EMOTIONS dataset.

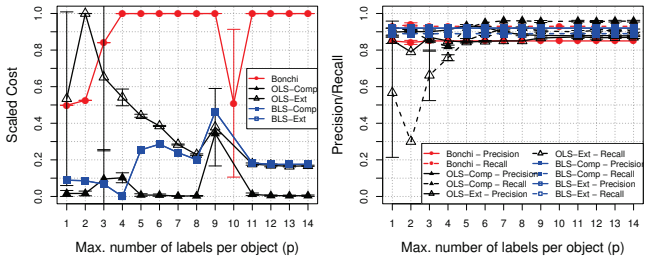
with the `BLS` decoder. Solid symbols correspond to the compact chromosome representation and hollow symbols correspond to the extended representation. The description of Figures 2b and 2d is similar, but there, solid lines show precision and dashed lines, recall (with higher being better). Figures 2a and 2b correspond to the Jaccard index function and Figures 2c and 2d, to the set-intersection function.

The algorithms using the Jaccard index with $p = 1$ have similar performance. One notes that this case is very close to traditional clustering and results in a partition of the objects. But as more labels per object are allowed, the performance of `Bonchi` degrades while all but one of the BRKGAs perform well. The exception is `OLS-Comp`, which followed the performance of `Bonchi` and displayed a large variation in its results. The same decoder using the extended representation (`OLS-Ext`) obtained the best results on average. Analyzing Figure 2b, one can see that the precision and the recall of the BRKGAs are always above those of `Bonchi`. Even `OLS-Comp`, which obtained costs similar to those of `Bonchi`, showed better precision and recall. The algorithms that use the Bonchi et al. approach for set intersection showed worse performance than those using the `OLS` decoder. It is worthwhile to mention that the solutions obtained using the Jaccard index function have different cost values from those obtained using the set-intersection function, as the objective functions are different.

Figure 3 depicts results for the YEAST dataset. Its description is identical to that of Figure 2. Here, we observe a very different behavior for both the Jaccard index and set-intersection functions. For the Jaccard index, note that the algorithms using the `OLS` decoder perform poorly (Figure 3a) when the number of labels allowed for each object is small, while algorithms using `BLS` performed quite well. As p increases, `OLS` outperforms `BLS`. In particular, when p is between 8 and 11, the algorithms switch their behavior. However, it is interesting to notice that both precision and recall are relatively stable. For the set-intersection function,



(a) Cost – Jaccard. (b) Prec./Rec. – Jaccard.



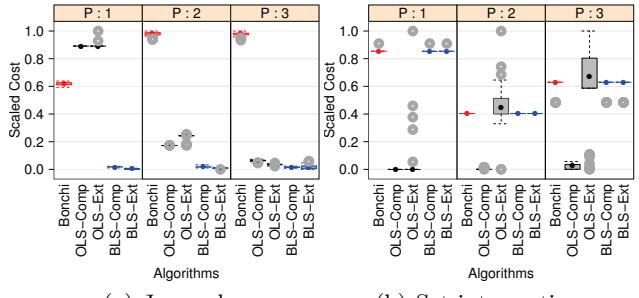
(c) Cost – Set-intersec. (d) Prec./Rec. – Set-intersec.

Figure 3: Comparison of costs, precision, and recall among the algorithms for the YEAST dataset.

the plots resemble the behavior of those for the Jaccard index function on the EMOTIONS dataset. Finally, note that the BRKGAs with `BLS` found better solutions on the YEAST dataset than they did on EMOTIONS. All BRKGAs obtained better precision on YEAST than they did on EMOTIONS.

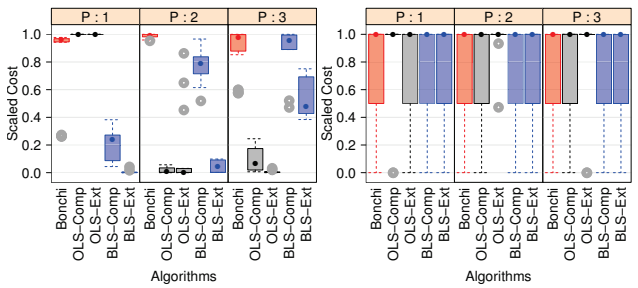
4.6 Evaluating the algorithms for instances with unknown multi-labeling

To evaluate the algorithms on instances with no multi-labeling given *a priori*, we first consider the Starkey project dataset. The algorithms were run for labelings of size $p = 1, 2, 3$, from a total of $k = 5$ available labels. Figure 4 shows the costs of the labelings obtained by the algorithms. For each scenario (consisting of an $\mathcal{H}$ function and a value of p), we scaled the costs into the interval $[0, 1]$ using the minimum and maximum costs of all algorithms. The box plots show the smallest cost (lowest whiskers), the first quartile (bottom box), the median cost (filled circles), the third quartile (upper box), the largest cost (highest whiskers), and the outliers (gray hollow circles). We observe that the sizes of the boxes for all configurations are very small, indicating that similar costs were found for all runs on a given algorithm.



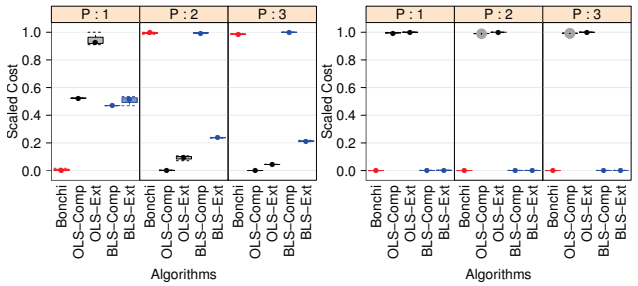
(a) Jaccard. (b) Set-intersection.

Figure 4: Boxplot of median and quartiles for each algorithm in Starkey dataset.



(a) Jaccard. (b) Set-intersection.

Figure 5: Boxplot of median and quartiles for each algorithm in SCOP dataset.



(a) Jaccard. (b) Set-intersection.

Figure 6: Boxplot of median and quartiles for each algorithm in newsgroup messages.

For the Jaccard index, `BLS-Comp` and `BLS-Ext` were able to produce the best results, although slightly different. For set intersection, `OLS-Comp` found better solutions, although in some cases `OLS-Ext` also found good solutions as its outliers suggest. To confirm these results, we applied the Wilcoxon-Mann-Whitney U test.This test assumes as the null hypothesis that the location statistics are equal in both distributions. Assuming a confidence interval of 95%, almost all algorithms presented significant difference in their results when compared to each other for the Jaccard index. The exception is the pair `BLS-Comp` and `BLS-Ext` for $p = 3$, whose p-value is 0.67 and we cannot thus assure a significant difference. For the set-intersection function, `OLS-Comp` was significantly better than the other algorithms. `Bonchi`, `BLS-Comp`, and `BLS-Ext` found solutions having the same value on all runs. The complete results are presented in the supplementary material.

Figure 5 shows results on the four SCOP datasets. The structure is similar to that of Figure 4, except that the scaling was done for each dataset set separately and then combined in the plots for each configuration. For the Jaccard index, `BLS-Comp` and `BLS-Ext` found good solutions for $p = 1$, whereas `OLS-Comp`, `OLS-Ext`, and `BLS-Ext` had the best results for $p = 2$. In fact, the U test did not present significant difference among these three algorithms and value of p (for a confidence interval of 95%). For $p = 3$, `OLS-Ext` presented significantly better results than the other algorithms. For set intersection, the algorithms performed similarly and for most cases, no significant difference was found. Notice that the results are closer to 1.0 (worst solutions) which indicates that the algorithms converged to local minima frequently. But since the bottom whiskers are at 0.0, the algorithms

did find a good solution. Again, refer to the supplementary material for the complete tests.

Figure 6 shows results for the instance from the newsgroup messages dataset. For the Jaccard index function, the behaviors of the algorithms are similar to those on the other instances. For $p = 1$, the algorithms using the Bonchi approach are able to find better results, but for $p \geq 2$, `OLS-Comp` found significantly better solutions. For set intersection, the algorithms using the Bonchi approach found the best results with no significant difference among them.

Table 1 lists the algorithms that obtained the best results for each dataset and configuration. For the Jaccard index, both `OLS-Ext` and `BLS-Ext` presented the best results for most cases, which indicates that algorithms using the extended representation are able to find better solutions than those using the compact representation. The `OLS` approach obtained more best solutions than did the `BSL` approach. For set intersection, the performances of the algorithms were similar, reaching the best solution at least once in most cases. In the Starkey dataset, the algorithms using `OLS` performed better than those which did not. For the newsgroup message dataset, `Bonchi` found the best solutions.

Table 1: Algorithms that computed the best results for each instance, $\mathcal{H}$ function, and p.

Inst.	p	Jaccard	Intersec	Inst.	p	Jaccard	Intersec
Starkey	1	BLS-Ext	OLS-Comp/ OLS-Ext	SCOP3	1	BLS-Ext	All
	2	BLS-Ext	OLS-Comp		2	OLS-Comp	Except OLS-Ext
	3	BLS-Ext	OLS-Comp/ OLS-Ext		3	OLS-Ext	All
SCOP1	1	BLS-Ext	All	SCOP4	1	BLS-Ext	All
	2	OLS-Ext	All		2	OLS-Ext	All
	3	OLS-Ext	All		3	OLS-Ext	All
SCOP2	1	BLS-Ext	All	News.	1	Bonchi	Bonchi
	2	OLS-Ext	All		2	OLS-Comp	Bonchi
	3	OLS-Ext	All		3	OLS-Comp	Bonchi

5. CONCLUDING REMARKS

In general, the BRKGAs are effective at finding good solutions and are able to beat the Bonchi et al. approach on most cases when using the Jaccard index in the objective function. For set intersection, the algorithms based on Bonchi et al. approach presented better results than other algorithms in most scenarios. Also, the extended representation allows the BRKGAs to obtain better results when compared to the compact representation on most cases. On the negative side, running times to convergence for the BRKGAs can be high. We believe that this is not a major issue, since most applications of OCC are prospective in nature and therefore do not require real-time response.

Acknowledgments

Carlos E. Andrade is supported by São Paulo Research Foundation (FAPESP) grants 2010/05233-5 and 2012/08222-0. Flávio K. Miyazawa is supported by National Council for Scientific and Technological Development (CNPq).

6. REFERENCES

[1] C. E. Andrade, F. K. Miyazawa, and M. G. C. Resende. Evolutionary algorithm for the k-Interconnected Multi-Depot Multi-Traveling Salesmen Problem. In *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO '13, pages 463–470, New York, NY, USA, 2013. ACM.

[2] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004.

[3] R. Bayer. Symmetric binary B-Trees: Data structure and maintenance algorithms. *Acta Informatica*, 1:290–306, 1972.

[4] J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 2(6):154–160, 1994.

[5] F. Bonchi, A. Gionis, and A. Ukkonen. Overlapping correlation clustering. *Knowledge and Information Systems*, pages 1–32, 2012.

[6] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 491–502, New York, NY, USA, 2005. ACM.

[7] A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 681–687, 2001.

[8] J. F. Gonçalves and M. G. C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17:487–525, 2011.

[9] G. Guennebaud, B. Jacob, et al. Eigen v3. `http://eigen.tuxfamily.org`, 2010.

[10] I. S. Haque, V. S. Pande, and W. P. Walters. Anatomy of high-performance 2D similarity calculations. *Journal of Chemical Information and Modeling*, 51(9):2345–2351, 2011.

[11] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM*, 5(4):339–342, 1958.

[12] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[13] A. G. Murzin et al. SCOP: Structural classification of proteins, 2009. Accessed on Oct 16, 2013.

[14] J. Rennie et al. 20 newsgroups. `http://qwone.com/~jason/20Newsgroups`, 2008. Accessed on Oct 13, 2013.

[15] M. M. Rowland, P. K. Coe, R. J. Stussy, A. A. Ager, N. J. Cimon, B. K. Johnson, and M. J. Wisdom. The Starkey habitat database for ungulate research: Construction, documentation, and use. Technical report, U.S. Forest Service General Technical Report PNW-GTR-430, Portland, Oregon, USA, 1998.

[16] R. Tanese. Parallel genetic algorithms for a hypercube. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 177–183. L. Erlbaum Associates Inc., 1987.

[17] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. Vlahavas. Multilabel classification of music into emotions. In *Proceedings of International Conference on Music Information Retrieval*, ISMIR 2008, pages 325–330, 2008.