EBIC.JL - an Efficient Implementation of Evolutionary Biclustering Algorithm in Julia

Paweł Renc
AGH University of Science and Technology
30-059 Krakow, Poland
rencpawe@gmail.com

Patryk Orzechowski*†
University of Pennsylvania
Philadelphia, PA 19104, USA
patryk.orzechowski@gmail.com

Aleksander Byrski
AGH University of Science and Technology
30-059 Krakow, Poland
olekb@agh.edu.pl

Jarosław Wąs
AGH University of Science and Technology
30-059 Krakow, Poland
jarek@agh.edu.pl

Jason H. Moore
University of Pennsylvania
Philadelphia, PA 19104
jhmoore@upenn.edu

ABSTRACT

Biclustering is a data mining technique which searches for local patterns in numeric tabular data with main application in bioinformatics. This technique has shown promise in multiple areas, including development of biomarkers for cancer, disease subtype identification, or gene-drug interactions among others. In this paper we introduce EBIC.JL - an implementation of one of the most accurate biclustering algorithms in Julia, a modern highly parallelizable programming language for data science. We show that the new version maintains comparable accuracy to its predecessor EBIC while converging faster for the majority of the problems. We hope that this open source software in a high-level programming language will foster research in this promising field of bioinformatics and expedite development of new biclustering methods for big data.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; • **Computing methodologies** → **Cluster analysis**; *Search methodologies*; Bio-inspired approaches; • **Theory of computation** → *Massively parallel algorithms*;

KEYWORDS

biclustering, data mining, machine learning, evolutionary computation, parallel algorithms

*corresponding author
†Patryk Orzechowski is also affiliated with AGH University of Science and Technology, Department of Automatics and Robotics, al. Mickiewicza 30, 30-059 Krakow, Poland

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463197

ACM Reference Format:
Paweł Renc, Patryk Orzechowski, Aleksander Byrski, Jarosław Wąs, and Jason H. Moore. 2021. EBIC.JL - an Efficient Implementation of Evolutionary Biclustering Algorithm in Julia. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449726.3463197

1 INTRODUCTION

Evolutionary algorithms, since their introduction in the 1960s, have been primarily applied to global optimization tasks [10]. Later their applicability was extended to tackle multi-criteria optimization tasks [51], where special care is needed to upkeep the diversity of the search space sampling in order to produce viable results. Evolutionary approaches have been also successfully applied to multiple machine learning tasks, including construction of decision trees [48], feature selection [50], feature construction [17], symbolic model identification [40], parameter estimation [49], image feature manipulation [12]. In recent years, evolutionary algorithms have also gained much traction in the machine learning community, as multiple evolutionary approaches have led to competitive with state-of-the-art results in tasks of regression [18, 31], deep learning models optimization [41], neural networks architecture selection [23, 26] or automating machine learning (AutoML) [28]. A detailed survey on evolutionary machine learning may be found in [1].

Another area of research in which evolutionary approaches have achieved leading results is biclustering. This unsupervised machine learning technique, also named co-clustering or subspace clustering, aims at discovering relevant local patterns in input data by extracting biclusters - submatrices with specific properties, e.g., with same values in certain rows or/and columns, correlated rows, or rows and columns, shift, scaled, or shift and scaled values in rows [9, 24, 30, 36]. Since its first application to gene expression data over 20 years ago [7], biclustering and its algorithms have led to some meaningful discoveries in biology and biomedicine, including identification of biomarkers for cancer [42, 43], diseases subtypes [8, 22, 47] or adverse drug effects [11].

The development of highly effective biclustering methods capable of capturing multiple patterns in a large volume of data is

an emerging challenge. Up to this date tens or hundreds of biclustering methods have been proposed, including Plaid [19], ISA [3], Bimax [38], cMonkey [39], QUBIC [20], FABIA [13], Unibic [44], just to name a few. Not until 2018 has any of them been capable of providing high detection accuracy of multiple patterns, popular in biclustering research, in reasonable time. The landscape has changed with the introduction of Evolutionary search-based Biclustering (EBIC), which managed to solve all of the problems with average accuracy exceeding 90%, and additionally offered unprecedented scalability [35]. The algorithm, which takes its strength from GPU utilization, was originally dedicated for a single GPU but was later scaled to multi-GPU systems [32]. Implemented in highly performing C++ and CUDA, EBIC became a reference point for newly developed algorithms, such as QUBIC2 [46] and RecBic [21].

Limited availability of methods implemented in high productive programming languages is one of the main obstacles slowing down biclustering research. As the biclustering problem is considered NP-hard, the compromise between the ease of development and rapid prototyping vs high performance is hard to accomplish. This is the place where modern programming languages developed with a built-it parallelism paradigm come with help, as they allow fast and easy prototyping and testing improvements of new solutions, while not compromising on efficiency and running times.

Julia, an emerging programming language for Data Science developed at MIT, has been specifically designed for high-performance numerical computing. It represents multi-paradigm language, uniting in itself features of different paradigms: functional, imperative and object-oriented. Julia programs compile using just-in-time (JIT) strategy to efficient native code using Low Level Virtual Machine (LLVM). Availability of multiple machine learning libraries, e.g., FLUX.jl [15], and MLJ [6], as well as wrappers, e.g. TensorFlow.jl [25], ScikitLearn.jl [1] increase productivity in creating modern AI or machine learning applications. All aforementioned features make this specialized high-level programming language very convenient for rapid prototyping.

As Julia becomes more prevalent in the data science world, there is an emerging need for expanding the spectrum of available algorithms in this programming language. The release of multiple packages, such as CUDA.jl [4, 5] allows to conveniently utilize NVidia graphics cards and further speed computation. After over 6 years since its first release in 2014, the library is mature and, provides, due to Julia's reflection and metaprogramming capabilities, an abstraction that allows direct interaction with CUDA API.

In this paper we present *EBIC.jl*, an optimized and updated GPU-based implementation of EBIC biclustering algorithm in Julia. To our best knowledge, this might be the first available and thoroughly tested implementation of any biclustering algorithm in this modern programming language for data science. Our method is benchmarked on two large collections of datasets that have been previously used for measuring performance of biclustering methods. The results on 275 different datasets show that EBIC.jl offers higher adaptability to newly introduced sets of problems than the original algorithm (vanilla EBIC) and converges in multiple cases much faster and to better solutions. We also demonstrate that in vast majority of scenarios EBIC.jl is highly competitive with the leading method in the field, RecBic.

[1] https://github.com/cstjean/ScikitLearn.jl

2 METHODS

In this section we present the design of the study and describe in more details two collections of the datasets used for benchmarking, the methods included in this study as well as the metric, which was used to measure their performance.

2.1 Datasets

In order to deliver a more thorough picture describing performance of the newly developed method, two different benchmarks were used for evaluation of the performance: collection from Wang et. al with 119 datasets grouped into 8 different sets of problems [44], which we called *UniBic benchmark*, and collection from Xie et al. with 156 datasets grouped into 14 different sets of problems [45], which we named *RecBic benchmark*. It was the authors' decision to reuse the existing datasets in the domain, instead of proposing another test suite, similarly to the authors of UniBic or RecBic.

Unibic benchmark. UniBic [44] is a synthetically generated set of the most biologically meaningful types of data containing biclusters. It is divided into three test groups, each of which corresponds to a different problem commonly encountered in gene expression analysis.

The first one is evaluated to check an algorithm's ability to identify six popular data patterns: trend-preserving, column-constant, row-constant, shift, scale, and shift-scale, often referred as Type I, Type II, Type III, Type IV, Type V and Type VI respectively. The tests assess how accurately algorithms detect differently-sized square biclusters in relatively small input data (up to $300x200$). The tests include three different sizes of biclusters, 5 replicates for each size, 90 datasets in total.

The second test set analyses the behavior of a method in scenarios with increasingly overlapping biclusters of size $25x25$. The patterns intersect up to $9x9$ elements creating four separate test cases with 5 replicates in each, 20 datasets in total.

The last set verifies biclustering methods' accuracy in very narrow pattern detection, i.e., ones with many rows and just several columns. The methods are requested to detect three biclusters with 10, 20, and 30 columns (3 test cases) and 100 rows implanted in the matrix of size $1000x100$.

The total number of datasets in this benchmark is 119.

RecBic benchmark. The second benchmark was downloaded from [21]. It consists of synthetically generated datasets with implanted biclusters, and similarly to the Unibic benchmark, the datasets are divided into several test groups. The tests for six different bicluster patterns are present in both benchmarks and were omitted by us in this one.

The benchmark comes with the following new challenges compared to the Unibic benchmark:

- *Noise.* The test case validates algorithms' robustness by checking their ability to handle poor-quality data. The resistance of the methods is tested under three different magnitudes of background noise (0.1, 0.2 and 0.3 with 0 being a sanity check).

- *Colincrease.* The test scenario verifies how well biclustering methods find a solution when biclusters the number of conditions constituting a biclusters increase (up to 120 columns).
- *1000Colin.* The test case examines how accurately algorithms discover biclusters when a background matrix is broadened from 500 to 2k columns, simultaneously maintaining the same size and position of true biclusters across all datasets.
- *Different.* In all other test cases, the data comes from Gaussian distribution with expectation 0 and deviation equal 1. In this test scenario, the influence of the various distributions of data constituting biclusters is checked by increasing the expectation, from 0 being a sanity check, up to 6.
- *20k-gene datasets.* Finally, to imitate real gene expression data in size, some of the aforementioned test cases (i.e., Six Types, Overlap, Different, Noise and Colincrease) were prepared in their larger variants with 20,000 rows and 250 columns.

Each test case comes with four variants resulting in the 156 datasets included, after removing cases overlapping with the previous benchmark.

2.2 Performance measure

Biclustering methods have been traditionally evaluated against ground truth using two performance measures: recovery and relevance [38]. Both measures are based on the Jaccard index between the rows [16] and reflect an average match of the maximum score for all rows of the biclusters to the ground truth. The relevance of biclusters shows how well detected biclusters represent true bicluster, whereas recovery shows the extent to which true biclusters are found by a biclustering method.

A much more intuitive measure of biclustering performance which involves both rows and columns of biclusters was proposed by Patrikainen and Meila and is called Clustering Error (CE) [37]. The name of this metric is a bit confusing - it doesn't measure an error, but the goodness of fit of two sets of biclusters. Thus, 1 indicates a perfect match and 0 denotes the worst possible fit. Before the metric is calculated, a confusion matrix is built based on the intersection of the biclusters, i.e. number of common elements shared between the detected and true biclusters. An optimal assignment between two sets of biclusters is found using Munkres algorithm [27]. Finally, the metric is calculated according to (1):

$$CE(S, S_1) = \frac{|U| - D_{max}}{|U|} \tag{1}$$

where D_{max} is the maximal sum on diagonal of confusion matrix between the ground truth and permutations of order of detected biclusters, and U is the union of biclusters. One of the properties of CE is much heavier penalization compared to recovery, or relevance for incorrect assignments of biclusters' either rows, or columns. Clustering error was also demonstrated to have more desired properties to objectively measure performance of the methods in biomedical studies in comparison with other measures [14, 30, 36].

2.3 Biclustering methods

In this study the following methods served as a benchmark: EBIC, RecBic, QUBIC2 and Runibic (parallel implementation of Unibic).

EBIC. EBIC [32, 35] is an algorithm relying on evolutionary search which supports distribution of computations across multiple graphic cards (GPUs). It was inspired by the concept of detecting order-preserving patterns introduced in OPSM [2]. Each individual in the population corresponds to a different bicluster, which is represented as a series of columns. The series defines an increasing order for the values in all of the rows that belong to the bicluster. The method uses simple genetic operations (such as insertion, deletion, substitution, swap, or crossover) in order to create new series of columns, or to merge two existing patterns. EBIC takes advantage from using GPUs in order to parallelize evaluation of fitness of the individuals. The evaluated scores are fetched in GPUs and transmitted back to CPU.

EBIC features multiple evolutionary strategies. Only a small fraction of best-fitting individuals is passed to the next generation (elitism), and its missing part is replenished by mutating other chromosomes. Tournament selection with exponential penalty from overusing same columns is used to increase diversity of the population (crowding). The individuals in new generations are prevented from being reevaluated by storing their hashes (tabu list). If a certain number of tabu list hits is recorded, the method finishes as it converged to the point where it is unwilling to search other solution subspaces. The enhancement prevents doing irrelevant iterations and hence remarkably reduces overall algorithm time.

EBIC was demonstrated to be a highly accurate method, but also very scalable up to 8 GPUs thanks to its use of Nvidia CUDA API [33].

RecBic. RecBic [21] rephrases the concept of monotonously increasing trends of columns, which was successfully implemented in EBIC. RecBic starts with biclusters that contain 3 columns and exhaustively greedily expands their series by adding more columns, provided that there is sufficient number of matching rows. RecBic, similarly to EBIC, excludes rows that violate monotonicity and allows certain degree of noise. RecBic is based on not very scalable source code of QUBIC [20].

QUBIC2. One of the recent biclustering methods dedicated to analyzing RNA-seq datasets is QUBIC2 [46]. This modification of QUBIC biclustering algorithm originally developed by Li et al. [20] was demonstrated as the most versatile method across different platforms, including synthetic, microarrays, bulk RNA-Seq and scRNA-Seq datasets. QUBIC2, similarly to its predecessor QUBIC, uses graph representation of biclusters and tries to find heavy subgroups in a graph with vertices representing rows, and edge weights - the level of similarity between the rows. The method comprises of three steps: discretization of input data, graph construction with seed selection, building core biclusters, which are further expanded and filtered. The source code of QUBIC2 is also based on its predecessor QUBIC.

Runibic. Runibic [34], another method that conceptually attempts to identify monotonous trends in the dataset, is a parallelized version of UniBic biclustering method by Wang et al. [44]. The method relies on evaluation of the longest common subsequence (LCS) of ranks between multiple pairs of rows. The ranks are determined by the increasing order of the values in each row, where the rank of k reflects the k-th smallest element in each row. Although the source

code of its predecessor is based on QUBIC, Runibic fixes some of its limitations by providing new and more modern implementation in R with C++ backend.

2.4 EBIC.jl

EBIC.jl proposed herein is a novel implementation of EBIC in Julia, and it is probably the first highly effective biclustering method available in this programming language. The selection of technology remarkably expedited the development process and enabled to locate issues due to the significantly lessened code complexity compared to the previous version. The algorithm works as follows:

(1) Initialize population with randomly generated short chromosomes represented by a vector of column numbers.
(2) Score every individual of the initial population by compressing the whole population and sending it on GPU where trends between rows are sought.
(3) Download numbers of trend-preserving rows corresponding to each chromosome, evaluate scores and update the top rank list.
(4) Initialize a new population with best-fitting individuals from the top rank list (elitism).
(5) Replenish the new population with mutated chromosomes from the old population penalizing for frequently used columns.
(6) Evaluate tabu list hits (the number of mutations after which the resulting chromosome was rejected) and if the count reaches a threshold, go to step 9.
(7) Score population and update the top rank list.
(8) Replace the old population with the newly created one and return to step 4.
(9) Send all individuals from the top rank list on GPU and download corresponding trend-preserving rows' indices. The pairs of sets of rows and columns create biclusters.

Additionally, to increase the performance, the following modifications were incorporated compared to the original EBIC:

Using registers and atomic operations. Most of the optimizations that might have influenced the performance are strictly related to CUDA kernels. Firstly, shared memory used for storing local variables was replaced by registers as their access time is much shorter [29]. A significant change was not making an intermediate data storing point when acquiring GPU computations results and exploiting atomic operations, which are thread-safe.

Loop unrolling. Subsequently, the reduce design pattern was improved by applying the loop unroll. This technique is a simple and popular method of optimization in GPU computing. It enables to omit loop control overheads such as end-of-loop tests and branch penalties by expanding a loop to regular code with control statements. The improvement could not be applied entirely because, in the current version of CUDA.jl, there is no correspondence of the volatile keyword from CUDA C. This is an indication for a compiler that the access to the shared memory must be an actual memory read or write instruction. With its use, there would not be a need for additional expensive memory synchronization.

Other differences to 'vanilla' EBIC. Finally, the values of constant parameters were further tuned. The allowed number of tabu hits was reduced as the method, even though converging quickly, was looking further for solutions in irrelevant subspaces. Additionally, we adjusted the initial population settings, as a significant part of the randomly generated chromosomes was spawned unreasonably short and died in the first cycle of evolution. It quickly resulted in a poorly differentiated generation delaying its proper development.

The new prototype EBIC.jl does not support multiple GPUs at this point. Although this limitation still allows to analyze pretty large sizes of datasets, it is a notable disadvantage for big data analyses, limiting the size of the datasets to available memory of a single GPU. The feature certainly will be the subject of further works.

The input parameters of the method include options of switching on/off negative trends, setting a threshold for approximate trends, the level of overlap allowed between the trends, number of iterations and biclusters. All the parameter of the method are documented on the Github repository.

3 RESULTS

The selected biclustering methods were tested on the Unibic and RecBic benchmarks and their results quality was assessed using the Clustering Error (CE). Each of the test cases in these benchmark contains five replicates (datasets with the same characteristics). Therefore, due to a large number of test sets, initiated the method with multiple random seed was not deemed necessary. The biclustering experiments were conducted on two different workstations. CPU-based methods (i.e., RecBic, QUBIC2 and Runibic) were tested on a machine equipped with Intel Core i7-10700 CPU[2], whereas tests of GPU-based methods (i.e., EBIC and EBIC.jl) were carried on the workstation equipped with a bit slower Intel Core i7-9700KF CPU[3] and GeForce GTX 1050 Ti GPU. In our opinion, this experiment settings enables us to assess the methods' actual performance more objectively, as the former group benefits more from CPU multithreading, whereas the latter group from GPU parallelization.

The settings of the benchmarked algorithms are presented in Table 1. The methods were run with the settings suggested by the authors. For Runibic we tried both settings (default and legacy) and adopted one yielding better results on average (i.e. legacy). For QUBIC2, parameter N was used, which according to the algorithm's authors, refers to 1.0 biclustering (1.0 objective function + regular expansion) and in our trial runs yielded the best results on average. Additionally, all algorithms were requested to return exact number of biclusters, as implanted, for each test case.

Table 1: Settings of the methods.

Algorithm	Settings
EBIC.jl	-n 20000
EBIC	-n 20000
RecBic	[default]
Runibic	[dafeault] and useLegacy=True
QUBIC2	-N

[2]https://www.cpubenchmark.net/cpu.php?cpu=Intel+Core+i7-10700+%40+2.90GHz&id=3747
[3]https://www.cpubenchmark.net/cpu.php?cpu=Intel+Core+i7-9700KF+%40+3.60GHz&id=3428

3.1 Unibic benchmark

Performance of the methods across multiple problems in UniBic benchmark in terms of CE is presented in Figure 1. EBIC achieves the best score overall for most tested datasets with its median equal 1, whereas EBIC.jl and RecBic perform slightly worse, reaching 0.98 and 0.94 respectively. The median accuracy of Runibic in terms of CE on UniBic dataset is 0.78. QUBIC2 gives remarkably lower results (0.13).

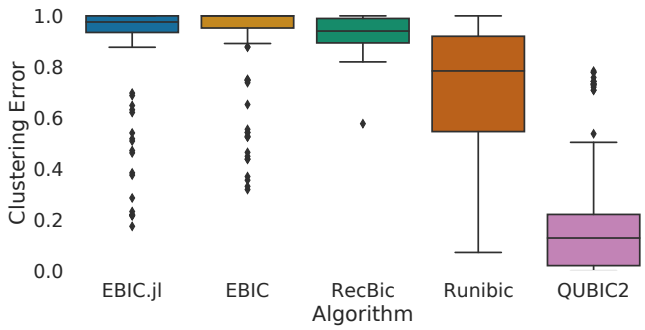

Figure 1: Clustering Error by an algorithm measured on Unibic benchmark.

It is worth noting that both EBIC and EBIC.jl have difficulties delivering satisfactory results in specific test cases, which is visible in Figure 2. The patterns of Type IV and VI, namely shift-scale and scale, are the most challenging for these algorithms, resulting in mean CE 0.77 and 0.70 for EBIC and 0.71 and 0.65 for EBIC.jl respectively. In the same test cases, RecBic discovers high-quality biclusters, scoring 0.95 for both patterns, but instead, its performance is lower in the Overlap and Type III (row-const pattern) tests, reaching 0.86 and 0.84, which are worse than for EBIC (0.89, 0.99), EBIC.jl (0.91, 0.99) and Runibic (0.90, 0.99).

Additionally, Runibic's average CE score is 0.74, and QUBIC2, apart from the Overlap test case, cannot find any accurate solution in the considered benchmark.

3.2 RecBic benchmark

The methods were subsequently tested on the second collection of the datasets. It needs to be noticed that this collection of datasets is provided by the authors of one of the methods included in the comparison (RecBic). Thus, some bias resulting in higher than expected performance of RecBic might be present in the following analysis.

As shown in Figure 3, RecBic achieves the best performance, standing out from the competing methods with a median equal to 0.94. The second best algorithm is EBIC.jl with a median quality of 0.74. The remaining methods visibly underperform, with medians 0.34 (EBIC), 0.07 (Runibic) and 0.01 (QUBIC2). It should be noted that the small performance difference on the previously evaluated benchmark between EBIC.jl and its ancestor (EBIC) no longer holds, confirming that the newly incorporated improvements brought desired outcomes.

For the smaller datasets with 1,000 rows, RecBic outperforms the rest of the algorithms in all tests (Fig. 4) except *Colincrease* and *1000colin* scenarios, where EBIC.jl performs better. EBIC.jl seems to be less affected by the increasing number of columns than RecBic. Large number of columns improves the performance of Runibic in both scenarios. EBIC.jl remains also the only method that can stand up to RecBic for the datasets with 20,000 columns, although its performance is lower.

Two *Overlap* scenarios need additional comment. The design of this scenario and placement of the biclusters seems to clearly put RecBic in favor. During the inspection of the heatmap of the datasets, we have unintentionally pointed out the same biclusters, as were found using EBIC and EBIC.jl, which turned out to be incorrect ones, both in number and location. We believe it is debatable which biclusters should be yielded in this scenario, but we also understand the rationale of the solutions found by RecBic, which in this scenario are closer to the ground truth.

Across the benchmark, Runibic obtains mediocre results for the majority of 1k-gene datasets at CE level equal 0.46 on average. Runibic runs into issues for 20k-gene datasets, as it is unable to find satisfactory solutions and crashes in two test cases of *Colincrease*, not outputting any results. QUBIC2 does not find any sensible solution in all of the RecBic benchmark datasets and achieves a mean score of CE of 0.01.

Afterwards analysing the overall results of the benchmarked we dwelved more thoroughly into specific test groups.

The results of *Six Types* are shown in Figure 5 broken down into distinct patterns. The test scenario is a much larger version of the six patterns from the Unibic benchmark. Type III scenario, namely row-constant pattern, turns out to be the easiest dataset for all the considered biclustering methods except for QUBIC2. For this scenario, EBIC.jl achieves the average CE of 0.97, which is slightly better than the other methods: RecBic 0.94, Runibic 0.93 and EBIC 0.84. For all the remaining patterns, RecBic turns out to discover the best-quality biclusters, achieving 0.94 on average on all scenarios. This score is higher than the second best method, EBIC.jl which average CE score is equal to 0.84. The remaining methods struggle to perform well on the benchmark, EBIC – 0.26, Runibic – 0.02 and QUBIC2 – 0.02. It can also be observed that the margin between EBIC.jl and RecBic on Type IV and VI patterns (shift-scale and scale) is much lower than in Unibic benchmark.

Subsequently, we break down the outcomes of the *Colincrease* test group in its 1k-gene variant (see Figure 6). EBIC.jl, EBIC and RecBic discover almost perfectly narrow and medium-wide biclusters reaching nearly 1.0 CE in most cases, only EBIC's accuracy slight decreases for some datasets in the 30-column test. However, in the wide-biclusters scenario, RecBic is incapable of finding an acceptable solution achieving merely 0.33 CE. On the contrary, EBIC.jl maintains high accuracy across different widths of the biclusters, scoring CE of 0.98 in all test cases on average. Runibic is utterly ineffective in discovering narrow biclusters, but its performance improves with the increasing number of columns reaching 1.0 CE in the broad-biclusters test case. It confirms that this method was purposefully designed for such scenarios. QUBIC2 again does not come with any acceptable solution.

Finally, we analyze how different noise levels incorporated in genes' data interfere with the performance of the biclustering methods. The results can be observed in Figure 7. The first test case

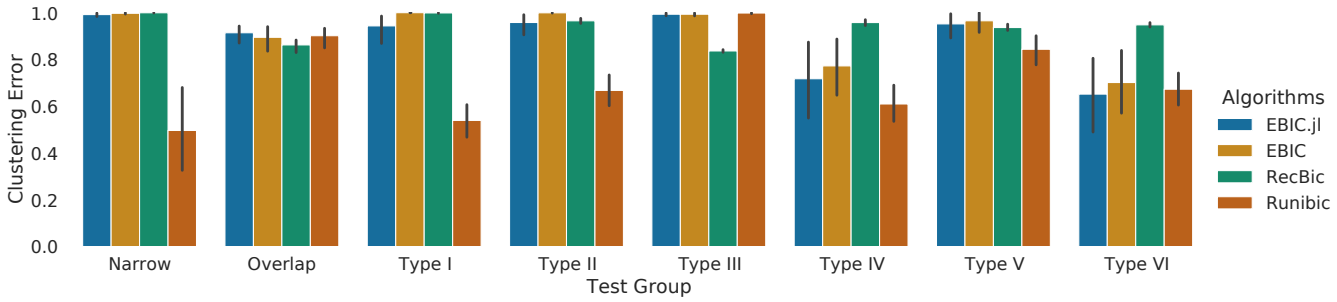

Figure 2: Clustering Error by different test groups for Unibic benchmark. The higher score, the better.

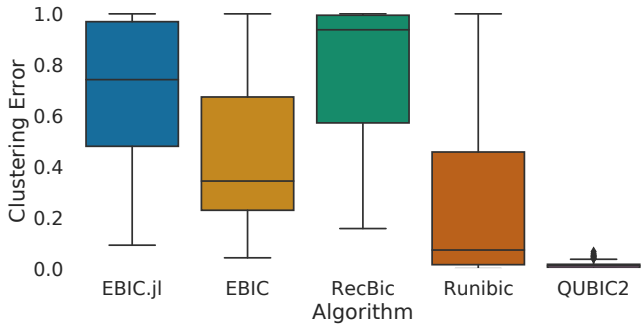

Figure 3: Clustering Error by an algorithm measured on RecBic benchmark. The higher, the better.

involves a sanity check with no noise included, the best three algorithms (EBIC.jl, EBIC and RecBic) achieve the same CE equal to 0.97. Then, together with increasing noise, their efficiency almost identically deteriorates, reaching CE 0.48 for RecBic, and 0.4 for EBIC.jl and EBIC in the scenario of the highest tested noise. It demonstrates that RecBic is somewhat more resistant when dealing with poor-quality data. Runibic seems to react similarly to the algorithms mentioned above, but its performance is much lower than the leading methods (0.3 CE on average). Lastly, we can not evaluate how noise in data influences QUBIC2 as the quality of all its results is close to 0.01 CE on average.

3.3 Running times

It should be stressed that the capability of finding high-quality solutions is essential, but a method applied in real-life scenarios is also requested to provide results in reasonable amount of time. The choice of the most suitable method might impose a compromise by trading overall accuracy for notably shorter execution time. For that reason, we compared the execution times of all five considered biclustering methods to check how fast they are in various scenarios. For this purpose, we again used two benchmarks, Unibic and RecBic.

Unibic/Runibic was historically deemed the leading algorithms in the field of biclustering. Similarly, QUBIC2 was claimed to outperform multiple other methods (including EBIC and runibic) on diverse collection of datasets [46]. However, as shown in the previous Subsection, on two large collections of datasets, they were found to be incapable of detecting high-grade solutions in the majority of test scenarios. In order to better analyze differences in performance of the current leading methods (RecBic, EBIC and EBIC.jl) we have focused on those methods.

The execution times obtained on the Unibic benchmark are presented in Figures 8 and 9. Its datasets are relatively small, ranging from 150 to 1000 rows and a few tens of columns. Among the more precise methods, RecBic operates the quickest in all test cases, lasting only 81 seconds on average. Our algorithm usually tackles most of the problems somewhat longer, apart from a few cases. The test cases of Type III, namely row-const pattern, take surprisingly long for EBIC.jl (11 minutes on average), pushing its average execution time up to 145 seconds. On the other hand, EBIC, even though being a predecessor for our algorithm, achieves worse performance (5.5 minutes on average) and operates visibly slower in different test cases than EBIC.jl (e.g. Type IV – 8.3 minutes and Type I – 7 minutes on average). Runibic and QUBIC2 are both very fast algorithms, their average time is 3.6 and 0.02 seconds respectively.

The second comparison is made on RecBic benchmark, which includes two sizes of datasets (1k rows and 20k rows). The execution time of all runs is presented in Figures 10 and 11, showing the running times across all of the problems.

Among the most precise biclustering algorithms, EBIC.jl discovers biclusters in the shortest time on average (248s), which is considerably shorter than two other algorithms, EBIC (1148s) and RecBic (1294s). The most time-consuming test case for the RecBic algorithm is *1000Colin*, with 2k columns because the algorithm, even though operating 2.5 hours on average, scores only 0.3 CE, whereas our algorithm's average time for this scenario is 13 minutes, and it achieves 0.59 CE. This comparison suggests that our method is much more scalable to larger datasets than RecBic, and greatly lower running times are somewhat compromised by slightly lower accuracy. Additionally, the less precise algorithms again run much faster then these mention above. Their average times in the benchmark are: 1 minute for Runibic and 21 seconds for QUBIC2.

4 CONCLUSIONS

This paper introduces *EBIC.jl*, an open-source implementation of GPU-based evolutionary biclustering algorithm EBIC in Julia. To our best knowledge, this is the very first available and working biclustering method in this emerging programming language for

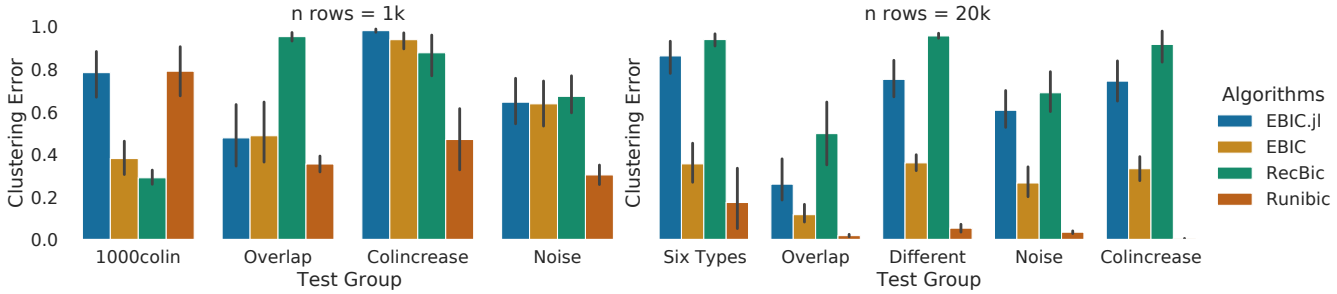

Figure 4: Clustering Error for different methods by different test groups on RecBic benchmark. To the left: test cases with 1,000 rows, to the right: test cases with 20,000 rows. The higher score, the better.

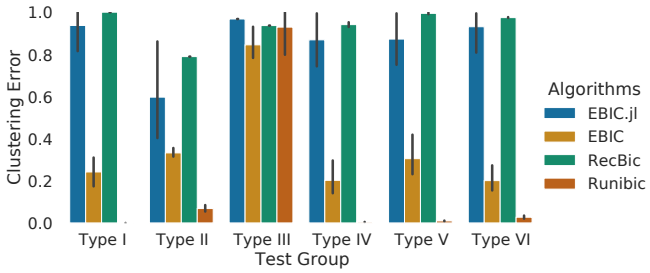

Figure 5: Clustering Error of the algorithm for six biclustering patterns on RecBic benchmark. Notice high increase in performance on this benchmark of EBIC.jl vs original EBIC. The higher score, the better.

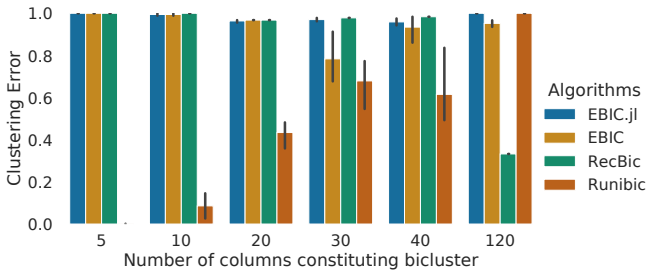

Figure 6: Clustering Error of biclustering with increasing number of columns on RecBic benchmark (*Colincrease* scenario). Notice that the performance of RecBic visibly drops with 120 columns, whereas EBIC.jl and Runibic maintain high accuracy.

data science. The proposed method was benchmarked on two extensive collections of datasets: introduced by Wange et al. [44] with 119 datasets grouped in eight different sets of problems, and Liu et al. [21] with 156 datasets and nine problems, and achieved the accuracy competitive to the state-of-the-art methods.

Our second contribution is providing a modern, optimized and enhanced implementation of one of the leading biclustering methods in the field, which not only works faster than the original

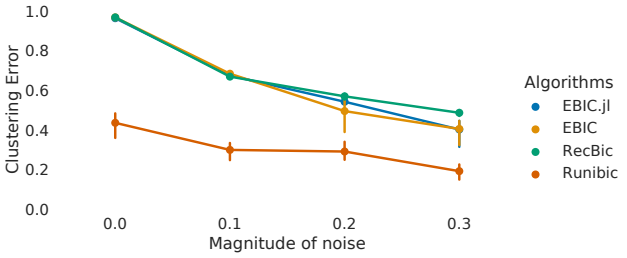

Figure 7: Clustering Error of biclustering methods for noise scenario on RecBic benchmark.

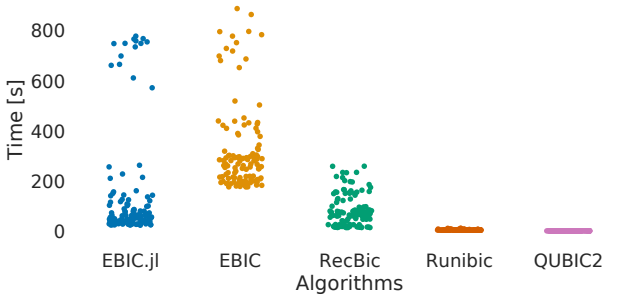

Figure 8: Running times by algorithm on Unibic benchmark. Each dot represents a single dataset.

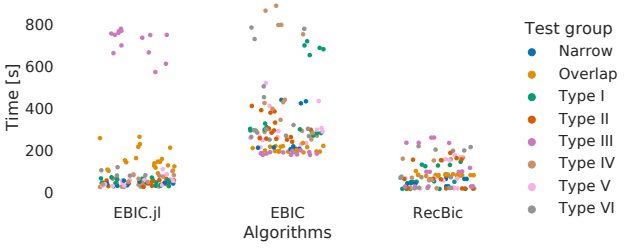

Figure 9: Running times by algorithm on Unibic benchmark for three top biclustering methods with the test group division. Each dot represents a single dataset.

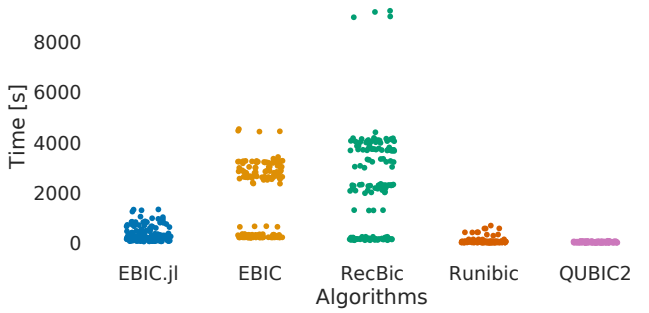

Figure 10: Running times of algorithms on RecBic benchmark. Each dot represents a single dataset.

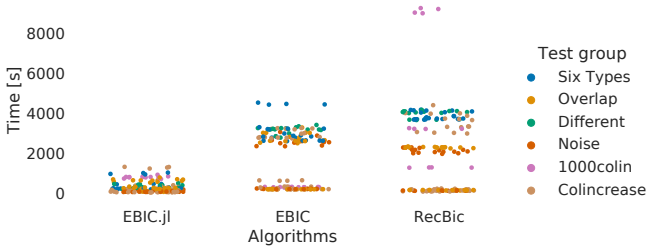

Figure 11: Running times of algorithms on RecBic benchmark for three top biclustering methods with the test group division. Each dot represents a single dataset.

method, but also outperforms it across multiple scenarios – visible especially in the second benchmark. There are several reasons, why EBIC.jl in Julia works faster than the original EBIC implemented in C++. First, the amount of communication sent between GPU and CPU was reduced by changing organization of CUDA kernel. Thus, EBIC.jl sends to CPU a single value for each of the columns in the group of threads. Reorganization also allowed us to use loop unrolling, which is considered more efficient. Second, the tabu hits allowance was decreased, what resulted in faster convergence. Third, we improved the generation of initial population, which was not efficiently managed in the original EBIC. Some other issues were also fixed, e.g. the genetic mutations did not work as expected and there was no chance for one for the deletion to happen.

This study has certain limitations. First, we have decided not to evaluate the methods on gene expression datasets. The reason for that is that the final results very frequently depend on how the datasets were preprocessed. As none of the workflows has been established in biclustering, our study might unnecessarily bring high bias and become detached from the previous reports. Secondly, statistical analyses comparing performance of the methods were excluded from the paper. The main reason for that is that such analyses are not commonly applied in biclustering benchmarking studies (e.g. Eren et al. [9], Padilha et al. [36]). Instead, in line with the previous studies, a detailed report highlighting performance of the methods on solving different biclustering problems is provided. Thirdly, the comparison of the runtimes between EBIC and EBIC.jl might be biased, as some optimizations (loop unroll, atomics etc.) have not been behchmarked for the vanilla EBIC.

Overall, we believe EBIC.jl can be considered a worthy expansion, or even a successor of EBIC. The new implementation has around half less lines of code than the original one and marks a milestone for prototyping, proving that an application written in a high-level programming language can successfully compete in performance with those written in considered the most efficient language, C++. Simultaneously ultimately excelling them in terms of software engineering, debugging, maintenance and potential extensibility.

Future work on the method will focus on expanding the implementation to multiple GPUs and parallelization of CPU parts of the code. The method will also be tested on multiple public datasets. Lastly, we plan to work on incorporating new techniques that will accelerate the algorithm's converging time.

The new implementation will also serve as a base for development of new biclustering methods. With already fast and parallel GPU-supported evaluation of the results, the emphasis can now be placed on generation of new candidates which can potentially lead to achieving better final solution in faster time.

ALGORITHM AVAILABILITY

EBIC.jl is open source: https://github.com/EpistasisLab/EBIC.jl

ACKNOWLEDGMENTS

This research was supported in part by PLGrid Infrastructure and by NIH grants LM010098 and LM012601.

AUTHORS' CONTRIBUTION

PO conceived the study. PR implemented the algorithm and performed analyses. PR and PO analyzed the results. PR and PO wrote the draft of the manuscript with help of AB, JW and JHM. AB and JW consulted the project. All the authors edited and reviewed the manuscript. JHM supervised the project.

REFERENCES

[1] Harith Al-Sahaf, Ying Bi, Qi Chen, Andrew Lensen, Yi Mei, Yanan Sun, Binh Tran, Bing Xue, and Mengjie Zhang. 2019. A survey on evolutionary machine learning. *Journal of the Royal Society of New Zealand* 49, 2 (2019), 205–228. https://doi.org/10.1080/03036758.2019.1609052

[2] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. 2003. Discovering local structure in gene expression data: the order-preserving submatrix problem. *J. Comput. Biol.* 10, 3-4 (2003), 373–384.

[3] Sven Bergmann, Jan Ihmels, and Naama Barkai. 2003. Iterative signature algorithm for the analysis of large-scale gene expression data. *Physical review E* 67, 3 (2003), 031902.

[4] Tim Besard, Valentin Churavy, Alan Edelman, and Bjorn De Sutter. 2019. Rapid software prototyping for heterogeneous and distributed platforms. *Advances in Engineering Software* 132 (2019), 29–46.

[5] Tim Besard, Christophe Foket, and Bjorn De Sutter. 2018. Effective Extensible Programming: Unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed Systems* abs/1712.03112 (2018). https://doi.org/10.1109/TPDS.2018.2872064

[6] Anthony D. Blaom, Franz Kiraly, Thibaut Lienart, Yiannis Simillides, Diego Arenas, and Sebastian J. Vollmer. 2020. MLJ: A Julia package for composable machine learning. *Journal of Open Source Software* 5, 55 (2020), 2704. https://doi.org/10.21105/joss.02704

[7] Yizong Cheng and George M. Church. 2000. Biclustering of Expression Data. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, 93–103. http://dl.acm.org/citation.cfm?id=645635.660833

[8] Phuong Dao, Recep Colak, Raheleh Salari, Flavia Moser, Elai Davicioni, Alexander Schönhuth, and Martin Ester. 2010. Inferring cancer subnetwork markers using density-constrained biclustering. *Bioinformatics* 26, 18 (2010), i625–i631.

[9] Kemal Eren, Mehmet Deveci, Onur Küçüktunç, and Ümit V Çatalyürek. 2013. A comparative analysis of biclustering algorithms for gene expression data. *Briefings in bioinformatics* 14, 3 (2013), 279–292.

[10] David E. Goldberg and John H. Holland. 1988. Genetic Algorithms and Machine Learning. *Mach. Learn.* 3 (1988), 95–99.

[11] Rave Harpaz, Hector Perez, Herbert S Chase, Raul Rabadan, George Hripcsak, and Carol Friedman. 2011. Biclustering of adverse drug events in the FDA's spontaneous reporting system. *Clinical Pharmacology & Therapeutics* 89, 2 (2011), 243–250.

[12] Samuel Hindmarsh, Peter Andreae, and Mengjie Zhang. 2012. Genetic Programming for Improving Image Descriptors Generated Using the Scale-Invariant Feature Transform. In *Proceedings of the 27th Conference on Image and Vision Computing New Zealand* (Dunedin, New Zealand) *(IVCNZ '12)*. Association for Computing Machinery, New York, NY, USA, 85–90. https://doi.org/10.1145/2425836.2425855

[13] S. Hochreiter, U. Bodenhofer, M. Heusel, A. Mayr, A. Mitterecker, A. Kasim, T. Khamiakova, S. Van Sanden, D. Lin, W. Talloen, et al. 2010. FABIA: factor analysis for bicluster acquisition. *Bioinformatics* 26, 12 (2010), 1520–1527.

[14] Danilo Horta and Ricardo JGB Campello. 2014. Similarity measures for comparing biclusterings. *IEEE/ACM transactions on computational biology and bioinformatics* 11, 5 (2014), 942–954.

[15] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. 2018. Fashionable Modelling with Flux. *CoRR* abs/1811.01457 (2018). arXiv:1811.01457

[16] Paul Jaccard. 1912. The distribution of the flora in the alpine zone. 1. *New phytologist* 11, 2 (1912), 37–50.

[17] K. Krawiec. 2002. Genetic Programming-based Construction of Features for Machine Learning and Knowledge Discovery Tasks. *Genetic Programming and Evolvable Machines* 3 (2002), 329–343.

[18] William La Cava, Lee Spector, and Kourosh Danai. 2016. Epsilon-Lexicase Selection for Regression *(GECCO '16)*. Association for Computing Machinery, New York, NY, USA, 741–748. https://doi.org/10.1145/2908812.2908898

[19] Laura Lazzeroni and Art Owen. 2002. PLAID MODELS FOR GENE EXPRESSION DATA. *Statistica Sinica* 12, 1 (2002), 61–86. http://www.jstor.org/stable/24307036

[20] G. Li, Q. Ma, H. Tang, A. H. Paterson, and Y. Xu. 2009. QUBIC: a qualitative biclustering algorithm for analyses of gene expression data. *Nucleic acids research* 37, 15 (2009), e101–e101.

[21] Xiangyu Liu, Di Li, Juntao Liu, Zhengchang Su, and Guojun Li. 2020. RecBic: a fast and accurate algorithm recognizing trend-preserving biclusters. *Bioinformatics* 36, 20 (07 2020), 5054–5060. https://doi.org/10.1093/bioinformatics/btaa630

[22] Yiyi Liu, Quanquan Gu, Jack P Hou, Jiawei Han, and Jian Ma. 2014. A network-assisted co-clustering algorithm to discover cancer subtypes based on gene expression. *BMC bioinformatics* 15, 1 (2014), 1–11.

[23] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. 2019. NSGA-Net: Neural Architecture Search Using Multi-Objective Genetic Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Prague, Czech Republic) *(GECCO '19)*. Association for Computing Machinery, New York, NY, USA, 419–427. https://doi.org/10.1145/3321707.3321729

[24] S. C. Madeira and A. L. Oliveira. 2004. Biclustering algorithms for biological data analysis: a survey. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on* 1, 1 (2004), 24–45.

[25] Jonathan Malmaud and Lyndon White. 2018. TensorFlow.jl: An idiomatic Julia front end for TensorFlow. *Journal of Open Source Software* 3, 31 (2018), 1002.

[26] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. 2019. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 293–312.

[27] James Munkres. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics* 5, 1 (1957), 32–38.

[28] Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. 2016. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I.* Springer International Publishing, Chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, 123–137. https://doi.org/10.1007/978-3-319-31204-0_9

[29] Patryk Orzechowski and Krzysztof Boryczko. 2015. Effective biclustering on GPU-capabilities and constraints. *Prz Elektrotechniczn* 1 (2015), 133–6.

[30] Patryk Orzechowski, Krzysztof Boryczko, and Jason H Moore. 2019. Scalable biclustering — the future of big data exploration? *GigaScience* 8, 7 (06 2019). https://doi.org/10.1093/gigascience/giz078 giz078.

[31] Patryk Orzechowski, William La Cava, and Jason H. Moore. 2018. Where Are We Now? A Large Benchmark Study of Recent Symbolic Regression Methods. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Kyoto, Japan) *(GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 1183–1190. https://doi.org/10.1145/3205455.3205539

[32] Patryk Orzechowski and Jason H Moore. 2019. EBIC: an open source software for high-dimensional and big data analyses. *Bioinformatics* 35, 17 (01 2019), 3181–3183. https://doi.org/10.1093/bioinformatics/btz027 arXiv:https://academic.oup.com/bioinformatics/article-pdf/35/17/3181/29591797/btz027.pdf

[33] Patryk Orzechowski and Jason H. Moore. 2019. Mining a Massive RNA-Seq Dataset with Biclustering: Are Evolutionary Approaches Ready for Big Data?. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Prague, Czech Republic) *(GECCO '19)*. Association for Computing Machinery, New York, NY, USA, 304–305. https://doi.org/10.1145/3319619.3321916

[34] Patryk Orzechowski, Artur Pańszczyk, Xiuzhen Huang, and Jason H Moore. 2018. runibic: a Bioconductor package for parallel row-based biclustering of gene expression data. *Bioinformatics* 34, 24 (2018), 4302–4304. https://doi.org/10.1093/bioinformatics/bty512

[35] Patryk Orzechowski, Moshe Sipper, and Xiuzhen Huang. 2018. EBIC: an evolutionary-based parallel biclustering algorithm for pattern discovery. *Bioinformatics* 34, 21 (05 2018), 3719–3726. https://doi.org/10.1093/bioinformatics/bty401

[36] Victor A Padilha and Ricardo JGB Campello. 2017. A systematic comparative evaluation of biclustering techniques. *BMC bioinformatics* 18, 1 (2017), 55.

[37] Anne Patrikainen and Marina Meila. 2006. Comparing subspace clusterings. *IEEE Transactions on Knowledge and Data Engineering* 18, 7 (2006), 902–916.

[38] A. Prelić, S. Bleuler, P. Zimmermann, A. Wille, P. Bühlmann, W. Gruissem, L. Hennig, L. Thiele, and E. Zitzler. 2006. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics* 22, 9 (2006), 1122–1129.

[39] David J Reiss, Nitin S Baliga, and Richard Bonneau. 2006. Integrated biclustering of heterogeneous genome-wide datasets for the inference of global regulatory networks. *BMC bioinformatics* 7, 1 (2006), 1–22.

[40] Ankur Sinha, Pekka Malo, and Timo Kuosmanen. 2015. A Multiobjective Exploratory Procedure for Regression Model Selection. *Journal of Computational and Graphical Statistics* 24, 1 (2015), 154–182. https://doi.org/10.1080/10618600.2014.899236

[41] Y. Sun, G. G. Yen, and Z. Yi. 2019. Evolving Unsupervised Deep Neural Networks for Learning Meaningful Representations. *IEEE Transactions on Evolutionary Computation* 23, 1 (2019), 89–103. https://doi.org/10.1109/TEVC.2018.2808689

[42] Alain B Tchagang, Ahmed H Tewfik, Melissa S DeRycke, Keith M Skubitz, and Amy PN Skubitz. 2008. Early detection of ovarian cancer using group biomarkers. *Molecular cancer therapeutics* 7, 1 (2008), 27–37.

[43] Yi Kan Wang, Edmund J Crampin, et al. 2013. Biclustering reveals breast cancer tumour subgroups with common clinical features and improves prediction of disease recurrence. *BMC genomics* 14, 1 (2013), 1–15.

[44] Zhenjia Wang, Guojun Li, Robert W Robinson, and Xiuzhen Huang. 2016. UniBic: Sequential row-based biclustering algorithm for analysis of gene expression data. *Scientific reports* 6, 1 (2016), 1–10.

[45] Juan Xie, Anjun Ma, Anne Fennell, Qin Ma, and Jing Zhao. 2018. It is time to apply biclustering: a comprehensive review of biclustering applications in biological and biomedical data. *Briefings in Bioinformatics* 20, 4 (02 2018), 1450–1465. https://doi.org/10.1093/bib/bby014 arXiv:https://academic.oup.com/bib/article-pdf/20/4/1450/31614291/bby014.pdf

[46] Juan Xie, Anjun Ma, Yu Zhang, Bingqiang Liu, Sha Cao, Cankun Wang, Jennifer Xu, Chi Zhang, and Qin Ma. 2019. QUBIC2: a novel and robust biclustering algorithm for analyses and interpretation of large-scale RNA-Seq data. *Bioinformatics* 36, 4 (09 2019), 1143–1149. https://doi.org/10.1093/bioinformatics/btz692

[47] Liying Yang, Yunyan Shen, Xiguo Yuan, Junying Zhang, and Jianhua Wei. 2017. Analysis of breast cancer subtypes by AP-ISA biclustering. *BMC bioinformatics* 18, 1 (2017), 1–13.

[48] Huimin Zhao. 2007. A multi-objective genetic programming approach to developing Pareto optimal decision trees. *Decision Support Systems* 43, 3 (2007), 809 – 826. https://doi.org/10.1016/j.dss.2006.12.011 Integrated Decision Support.

[49] Xiuqing Zhou and Jinde Wang. 2005. A genetic method of LAD estimation for models with censored data. *Computational Statistics & Data Analysis* 48, 3 (2005), 451 – 466. https://doi.org/10.1016/j.csda.2004.03.002

[50] Z. Zhu, Y. Ong, and J. M. Zurada. 2010. Identification of Full and Partial Class Relevant Genes. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 7, 2 (2010), 263–277. https://doi.org/10.1109/TCBB.2008.105

[51] Eckart Zitzler and Kalyanmoy Deb. 2007. Evolutionary multiobjective optimization. In *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007, Companion Material*, Dirk Thierens (Ed.). ACM, 3792–3809. https://doi.org/10.1145/1274000.1274133