SNAP Biclustering

William Hannibal Chan

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

November 5, 2009
Falls Church, Virginia

Keywords:
Biclustering, Ant Colony Optimization, Microarray Analysis,
Collaborative Filtering, Single Nucleotide Polymorphisms

SNAP Biclustering

William Hannibal Chan

Abstract

This thesis presents a new ant-optimized biclustering technique known as SNAP biclustering, which runs faster and produces results of superior quality to previous techniques. Biclustering techniques have been designed to compensate for the weaknesses of classical clustering algorithms by allowing cluster overlap, and allowing vectors to be grouped for a subset of their defined features. These techniques have performed well in many problem domains, particularly DNA microarray analysis and collaborative filtering. A motivation for this work has been the biclustering technique known as bicACO, which was the first to use ant colony optimization. As bicACO is time intensive, much emphasis was placed on decreasing SNAP's runtime. The superior speed and biclustering results of SNAP are due to its improved initialization and solution construction procedures. In experimental studies involving the Yeast Cell Cycle DNA microarray dataset and the MovieLens collaborative filtering dataset, SNAP has run at least 22 times faster than bicACO while generating superior results. Thus, SNAP is an effective choice of technique for microarray analysis and collaborative filtering applications.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Advances in information technology have greatly reduced the cost of data acquisition, resulting in an exponential increase of available datasets. This rapid growth has increased the demand for low cost data analysis. Classical clustering techniques, such as K-means and hierarchical clustering, have become popular analysis tools [1] [2] [3]. However, these techniques have two drawbacks: they prohibit cluster overlap, and only perform clustering using all features. As a result, they have not performed well in DNA microarray analysis, a field of research that aims to learn more about biological processes [4] [5]. To address these drawbacks, this thesis develops an efficient and high-performing ant-optimized biclustering technique, named SNAP biclustering, which outperforms previous techniques.

Cheng and Church (CC) first introduced the use of *biclustering* algorithms in microarray analysis [6]. Biclustering is the task of finding large and highly similar *biclusters* from a dataset (matrix) [7] [8]. This problem will be described in detail in the next chapter.

The work of CC inspired development of improved techniques by other researchers. Yang et al. developed the FLOC algorithm [9], which surpasses CC in performance and runtime through its improved definition of bicluster size. Then, Coelho et al. developed the bicACO algorithm [10], the first biclustering technique to utilize ant colony optimization (ACO) [11]. Although bicACO produces even better biclustering results, its runtime is 24 times greater than CC [6] [11] [12]. BicACO's slow runtime serves as the primary motivation for this thesis. The objective is to develop a faster ant-optimized technique, while producing superior results.

Microarray analysis is the primary application domain of biclustering algorithms. SNAP will be evaluated using the Yeast microarray dataset [12] as was done in the cases of CC, FLOC, and bicACO. Recent research efforts have also extended biclustering techniques to other problem domains, particularly collaborative filtering (CF) [13]. CF datasets are extremely sparse, differentiating them from their microarray counterparts. In order to evaluate the potential effectiveness of SNAP in CF applications, SNAP will be applied to the MovieLens CF dataset (MovieLens) [14]. SNAP has produced superior results in both application domains. This algorithm runs at least 22 times faster than bicACO while producing better results for both datasets.

In this thesis, fundamental terms and concepts of the biclustering problem are introduced in Chapter 2. Chapter 3 outlines the details of the common application domains of biclustering techniques. Chapter 4 provides an overview of biclustering algorithms, with emphasis on CC and FLOC. Chapter 5 then outlines the bicACO algorithm, particularly how ant colony optimization was adapted to the biclustering problem. Chapter 6 describes the implementation details of SNAP, particularly the modified initialization and solution construction procedures. The experimental results, for Yeast and MovieLens, are presented in Chapters 7 and 8 respectively. The paper ends with some concluding remarks in Chapter 9.

2 Fundamental Concepts of Biclustering

Biclustering, also known as the *biclustering problem*, is the task of finding large and highly similar *biclusters* from a matrix. This term can also be used to refer to a *set of biclusters*.

2.1 What is a Bicluster?

Let an M-by-N matrix A represent a set of M vectors in N-dimensional space. Each entry a_{ij}, where $1 \leq i \leq M$ and $1 \leq j \leq N$, corresponds to the value of the j^{th} feature of the i^{th} vector.

A *bicluster B* is defined as some pair (I, J), where I is a subset of the rows of A, and J is a subset of the columns of A. B can also be defined as a set of *elements*, where an element refers to a distinct row or column of A. Elements 1 to M refer to rows 1 to M respectively. Elements $M + 1$ to $M + N$ refer to columns 1 to N respectively.

For example, assume that A is defined by the 9-by-6 matrix of (2-1), where 0 is arbitrarily selected to denote unspecified (null) entries. Two possible biclusters of A are $B_1 = (I_1, J_1) = (\{2,5\},\{1,4,6\}) = \{2,5,10,13,15\}$ and $B_2 = (I_2, J_2) = (\{3,4\},\{3,5\}) = \{3,4,12,14\}$.

$$A = \begin{bmatrix} 1 & 4 & 2 & 3 & 3 & 2 \\ \mathbf{9} & 1 & 5 & \mathbf{9} & 4 & \mathbf{9} \\ 1 & 0 & \mathbf{1} & 3 & \mathbf{2} & 7 \\ 3 & 6 & \mathbf{1} & 1 & \mathbf{2} & 1 \\ \mathbf{9} & 0 & 8 & \mathbf{9} & 6 & \mathbf{9} \\ 6 & 0 & 3 & 4 & 1 & 4 \\ 0 & 5 & 0 & 0 & 1 & 1 \\ 4 & 9 & 0 & 7 & 0 & 3 \\ 2 & 1 & 0 & 6 & 5 & 3 \end{bmatrix} \tag{2-1}$$

Since B contains a subset of A's rows and a subset of A's columns, it can be represented as a submatrix of A. The matrix entries, conserved by B's elements (rows and columns), are known as the entries of B. Thus, B_1 and B_2 can be expressed as:

$$B_1 = \begin{bmatrix} 9 & 9 & 9 \\ 9 & 9 & 9 \end{bmatrix} \tag{2-2}$$

$$B_2 = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} \tag{2-3}$$

B can also be represented as binary sequence of length $M + N$. The value of the i^{th} sequence entry (where $1 \leq i \leq M + N$) corresponds to the inclusion (or exclusion) of *element i* in B. A value of 1 indicates inclusion, while 0 indicates exclusion. Thus, $B_1 = 010\ 010\ 000\ 100\ 101$ and $B_2 = 001\ 100\ 000\ 001\ 010$.

Furthermore, B can be depicted as a bipartite graph. A can be viewed as a bipartite graph by treating it as an adjacency matrix. A graph is said to be bipartite if it exhibits two vertex sets and only allows edges to connect vertices of different sets [15]. With regards to the graph of *A*, one vertex set corresponds to rows, while the other corresponds to columns. Entry a_{ij} corresponds to the weight of the edge connecting row vertex *i* and column vertex *j*. The graphical representation of *A* is presented in Figure 2-1.

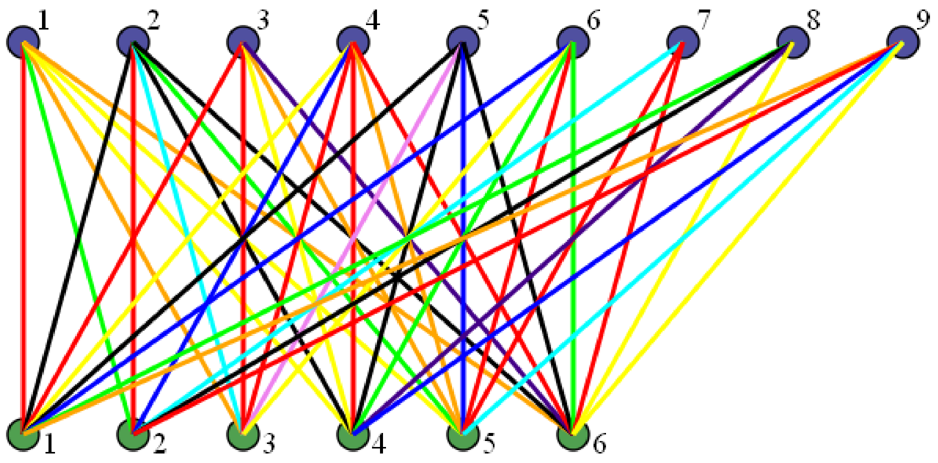

Figure 2-1: Graphical Representation of *A*

Here, the blue and green vertices correspond to row and column vertices respectively. Edge colors correspond to edge weights: (red, 1), (orange, 2), (yellow, 3), (green, 4), (cyan, 5), (blue, 6), (indigo, 7), (violet, 8), and (black, 9). Edges with 0 weights are not plotted because they correspond to null matrix entries.

Since *B* can be represented as a submatrix of *A*, *B* can also be represented as a bipartite subgraph. A bipartite subgraph is defined as a pair of subsets (*X*, *Y*), where *X* is a subset of the original graph's row vertex set, while *Y* is a subset of the original graph's column vertex set. The subgraph also retains edges that connect any pair of included row and column vertices. The graphical representations of B_1 and B_2 are presented in Figure 2-2 and Figure 2-3 respectively.

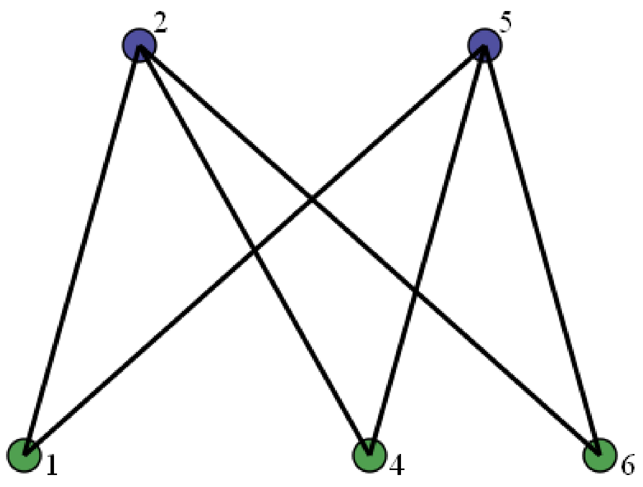

Figure 2-2 : Graphical Representation of B_1

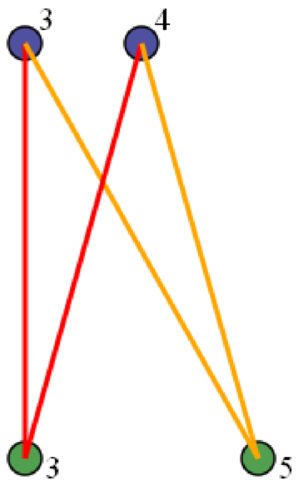

Figure 2-3 : Graphical Representation of B_2

2.2 Search Space of the Biclustering Problem

Search space (SS), with reference to the biclustering problem, is defined as the set of possible of biclusters that can be formed from the target matrix. If any row or column (element) can be included or excluded, the number of possible biclusters is 2^{M+N}. This value can be derived from the principles of binary sequence notation: 2^{M+N} unique sequences can be formed from M+N bits.

The majority of biclustering algorithms require biclusters to have at least 2 rows and at least 2 columns. Biclusters with 1 or fewer rows or columns are considered *trivial* [15]. Hence, the size of the search space of A can be expressed as:

$$|SS(A)| = 2^{M+N} - ((M+1)2^N + (N+1)2^M - (MN + M + N + 1)) \qquad \text{(2-4)}$$

The first term, 2^{M+N}, denotes the total number of trivial and nontrivial biclusters that can be formed by any M-by-N matrix. The second term, $(M+1)2^N$, represents the number of biclusters with 1 or 0 rows. The third term, $(N+1)2^M$, is the same, except it refers to column-deficient biclusters. The number of biclusters which are lacking in both rows and columns, is $MN + M + N + 1$. This number has been summed twice due to the addition of the second and third terms. The subtraction of forth term removes this redundancy. For matrices with large M + N, the first term becomes dominant, and the problem has NP complexity.

Many publications impose their own constraints on biclusters, reducing the size of the search space. Some algorithms strive to find biclusters like B_1 [16], in which all entries are identical; other works attempt to find biclusters like B_2 [17] with uniform-value columns. Nevertheless, the process of finding these biclusters often involves an extensive coverage of the target matrix's original search space.

2.3 Time Complexity of the Biclustering Problem

Researchers have formally shown that this problem is NP-hard [6] [15]. The proofs are founded on two principles: the graphical representation of a bicluster and the hardness proof of the balanced biclique subgraph problem [18].

3 Applications of Biclustering

Most biclustering algorithms have been developed to tackle the microarray analysis problem. Recent research has focused on extending the usefulness of these algorithms to different problem domains, such as collaborative filtering.

3.1 DNA Microarray Analysis

DNA Microarray technology applications have been used to great advantage in understanding serious afflictions like cancer [19] by enabling better surveillance of gene behavior and their interactions in biological systems (processes).

These surveillance capabilities have been established through transcriptome monitoring [4]. The transcriptome is the collection of the genome's expressed genes. The genome is the collection of all the genes of the host organism. Whenever a gene is expressed (activated), its effects become more prominent on the host organism. Whenever it is repressed (de-activated), its effects become less noticeable and potentially absent within the organism. The level of expression of a gene is proportional to the quantity of its transcripts (copies) within the transcriptome [20] [21]. Disease-causing biological systems involve the repression of beneficial genes and the expression of their harmful counterparts [19].

Within microarray experiments, the levels of different gene transcripts are measured in the transcriptome under varying conditions. These conditions often refer to different organs of an organism, and sometimes, under different environmental conditions, such as extreme heat [4]. Hence, microarray datasets are large matrices, depicting the expression levels of genes over a set of measured conditions. Each row corresponds to a distinct gene while each column refers to a distinct condition.

In earlier publications, scientists attempted to map biological systems using classical clustering techniques like K-means or hierarchical clustering. The purpose of these efforts was to discover new biological process, especially those that contribute to disease [19]. Genes were treated as vectors while conditions were treated as features. Clusters were formed using the Guilt-By-Association heuristic. Genes, exhibiting a high level of similarity as estimated by the dot product or Euclidian distance measures, were probably interacting to form a biological process [5] [22].

These efforts have yielded limited success because biological systems have two properties that make them inappropriate for traditional clustering:

(i) Biological processes may share gene members.
(ii) Gene interactions (in these systems) mostly occur for a limited set of conditions.

Consequently, biclustering methods are more successful because they allow inter-cluster overlap and they perform clustering for a subset of the vectors' features [6] [15] [17] [23] [24] [25] [26]. These techniques also allow the subset of features to vary for each cluster of vectors.

3.2 Collaborative Filtering

Collaborative filtering (CF) has been successfully used in dealing with the large datasets now available as a result of the internet age in the areas of consumer (user) choice. These datasets often consist of millions of consumers with an even greater number of potential choices. Effectively organizing this data avalanche to improve consumer choice is a non-trivial problem. Whether a consumer is searching for products such as movies, clothing, or even potential companions, the set of potential choices is influenced by the consumer's item preferences and the choices of other consumers with similar interests. The accuracy of a recommendation is positively related with a successful transaction. Thus, collaborative filtering can be defined as the process of using these sources of information to provide effective product recommendations [27] [28].

There are two major types of CF systems: memory-based and model-based algorithms. Memory-based systems follow the *K* nearest-neighbor paradigm, where for each user, *K* neighbors (other users) with similar item preferences are selected. A user's item preferences correspond to the items that have received high ratings from him/her. The neighbors' preferred items are then recommended to the user of interest. Alternatively, model-based algorithms attempt to classify users by their item preferences. The preferred items of other users in the same class are suggested to the active user. Memory-based applications yield high levels of accuracy, but low levels of scalability. Model-based applications are more scalable, but are not as fast or accurate [29] [30].

Nevertheless, the common deficiency of both types of methods lies in their computation of similarity. Similarities are based on all, not a subset, of the measurable dimensions. These techniques may not be able to provide useful recommendations to users with eclectic interests. People may often share interests on a subset of items, but their opinions may vary significantly regarding others. For example, it is relatively easy to find a large group of moviegoers with matching levels of interest in science fiction movies. It is less likely that those same people will share the same levels of interest in history, romance, and comedy films [13] [28].

Previous publications have demonstrated the power of biclustering algorithms within CF applications. The BIC-aiNet algorithm, outlined in [13], outperformed a variety of memory-based and model-based algorithms such as Probabilistic Memory-based Collaborative Filtering (PMCF) [31], Pearson Correlation (memory-based) [32], Naïve Bayes (model-based) [33], and a multi-layer perception network (model-based) [34]. BIC-aiNet even bested the Nearest-Biclusters technique, a biclustering algorithm tailored for CF applications [28].

3.3 Significance and Properties of Evaluation Datasets

The Yeast Cell Cycle dataset, also known as Yeast, details the expression levels of 2884 of the organism's genes (rows) for 17 experimental conditions (columns) [12]. The matrix has an occupancy ratio of 99.97% as only 15 entries are unspecified. The values of specified entries lie between 0 and 600 while -1 denotes a missing entry. The dataset, under the manual analysis of Cho et al., elucidated some of the biological systems responsible for cell division in the yeast organism and in the human body. Consequently, Yeast has been used to evaluate the performance of bicACO and its predecessors, CC and FLOC.

The MovieLens collaborative filtering dataset, also known as MovieLens, contains 80,000 user-submitted ratings from 943 different users (rows) regarding 1682 different movies (columns) [14]. Ratings range from 1 (worst) to 5 (best). The matrix has an occupancy ratio of 6.3% MovieLens has been used to evaluate many CF algorithms, including BIC-aiNet and Nearest-Biclusters [13] [28].

As evidenced by Yeast and MovieLens, microarray and collaborative filtering datasets differ greatly with regards to sparseness. Within microarray experiments, the intent is to capture as much data as possible. With respect to consumer choice and information available on the World Wide Web, it is likely that a consumer will not be able to express opinions on the full range of available choices. Therefore, these datasets will tend to be very sparse.

4 Overview of Biclustering Algorithms

The main source of variation of biclustering algorithms lies in the type of biclusters that they strive to find. The first type of biclustering algorithm attempts to find biclusters in which all entries are of *constant value* [16]. Subsequent methods relax this requirement, only striving to find biclusters with identical entries on the rows or columns [17] [35]. SNAP, and its predecessors CC, FLOC, and bicACO, search for *coherent value biclusters* (CVB). In general, bicluster similarity determines how well a bicluster fits the requirements of a particular type [15].

4.1 What is a Coherent Value Bicluster?

Each entry of an ideal CVB can be expressed as $b_{ij} = C + r_i + c_j$. C is an arbitrary constant, r_i is the adjustment for row i, and c_j is the adjustment for column j.

For example, assume B_{CVB} is an ideal CVB, using $r_1 = 1,\ r_2 = -2,\ r_3 = 3,\ r_4 = 8,$ $c_1 = 5,\ c_2 = 1,\ c_3 = 4,\ c_4 = 2,$ and $C = 1$.

$$B_{CVB} = \begin{bmatrix} 8 & 4 & 7 & 5 \\ 5 & 1 & 4 & 2 \\ 10 & 6 & 9 & 7 \\ 15 & 11 & 14 & 12 \end{bmatrix} \qquad (4\text{-}1)$$

Constant row biclusters (CRB) only allow for row adjustments. Constant column biclusters (CCB) only allow for column adjustments. Each entry of an ideal CRB is $b_{ij} = C + r_i$. Each entry of an ideal CCB is $b_{ij} = C + c_i$. Constant value biclusters (CBs) are more restricted. All entries must be of identical value. Thus, $b_{ij} = C$ for ideal CBs.

B_{CRB} is an ideal CRB in (4-2), where $r_1 = 1,\ r_2 = -2,\ r_3 = 3,\ r_4 = 8,$ and $C = 1$. B_{CB} is an ideal CB in (4-3), where $C = 1$.

$$B_{CRB} = \begin{bmatrix} 2 & 2 & 2 & 2 \\ -1 & -1 & -1 & -1 \\ 4 & 4 & 4 & 4 \\ 9 & 9 & 9 & 9 \end{bmatrix} \tag{4-2}$$

$$B_{CB} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{4-3}$$

The coherent value bicluster type specification is superior as it can also describe ideal CRBs, CCBs, and CBs. This flexibility is beneficial for microarray analysis. Consequently, CVB-finding algorithms, particularly CC and FLOC, have become quite popular in microarray research [36].

4.2 Cheng and Church (CC)

Cheng and Church devised the mean square residue (MSR) dissimilarity (noise) measure. MSR measures the degree to which a bicluster deviates from the requirements of a CVB [6]. The individual noise (residue) value of some entry b_{ij} of bicluster *B*, where *B* = (*I*,*J*), can be expressed using:

$$r_{ij} = b_{ij} - \mu_{iJ} - \mu_{Ij} + \mu_B \tag{4-4}$$

μ_{iJ} represents the mean value of the bicluster entries residing on row *i*. This is also known as the row-wise (or element-wise) average of *i*. μ_{Ij} represents the column-wise (or element-wise) average of *j*. Lastly, μ_B represents the bicluster-wise average of *B*.

$$\mu_{iJ} = \frac{1}{|J|}\sum_{j \in J} b_{ij} \qquad \mu_{Ij} = \frac{1}{|I|}\sum_{i \in I} b_{ij} \tag{4-5}$$

The MSR of *i*, *j*, and *B*, can be expressed as follows:

$$\text{MSR}(i, B) = \frac{1}{|J|}\sum_{j \in J} r_{ij}^2 \tag{4-6}$$

$$\text{MSR}(j, B) = \frac{1}{|I|}\sum_{i \in I} r_{ij}^2 \tag{4-7}$$

$$\text{MSR}(B) = \sum_{i \in I}\sum_{j \in J} r_{ij}^2 \tag{4-8}$$

The formula for MSR can be used to derive the ideal entry equation for perfect CVBs. Please note that $r_{ij} = 0$ for ideal CVBs because they exhibit zero noise.

$$b_{ij} = r_{ij} + \mu_{iJ} + \mu_{Ij} - \mu_B \tag{4-9}$$

$$b_{ij} = -\mu_B + \mu_{iJ} + \mu_{Ij} \tag{4-10}$$

$$C = -\mu_B,\ \ r_i = \mu_{iJ},\ \ c_j = \ \mu_{Ij} \qquad (4\text{-}11)$$

$$b_{ij} = C + r_i + \ c_j \qquad (4\text{-}12)$$

4.2.1 Algorithm Description

The objective of the Cheng and Church algorithm is to find *K* biclusters of large size and small MSR. Each MSR value must not exceed a threshold *R*. A bicluster which meets this requirement is known as an *R*-bicluster. *K* and *R* are user-defined parameters. The size of a bicluster is defined as the product of its row and column set sizes. All unspecified values are replaced with random values.

The Cheng and Church algorithm initializes each of the *K* biclusters to the original matrix. CC then utilizes a *greedy search* approach to remove and add elements in order to refine the biclusters.

The greedy search approach consists of three parts:

(i) *Multiple Element Deletion* (MEDel)*:* Member rows are removed if their MSR is greater than α times the bicluster's MSR, where $\alpha \geq 1$. The MSR is then recomputed. Thereafter, member columns are deleted in the same fashion. This entire process is repeated until the bicluster's MSR falls below *R*, or if no more rows or columns can be deleted.

(ii) *Single Element Deletion* (SEDel): The bicluster's element that has the largest MSR is removed. The MSR of the bicluster is then recomputed. This process continues until the bicluster's MSR falls below the *R* threshold.

(iii) *Element Addition* (EAdd): the MSR of non-member columns are returned into the bicluster if their potential MSR is less than the bicluster's MSR. The MSR is updated once again. Non-member rows are also added in the same fashion. This process is repeated until no more elements can be appended. Additionally, non-member rows are added if adding their negated values decreases the bicluster's MSR.

Assuming *B* = (*I*,*J*), the potential MSR of some excluded row *i* and some excluded column *j* can be expressed as:

$$\mathrm{MSR}(i, B) = \frac{1}{|J|} \sum_{g \in J} r_{ig}^2 \qquad (4\text{-}13)$$

$$\mathrm{MSR}(j, B) = \frac{1}{|I|} \sum_{h \in I} r_{hj}^2 \qquad (4\text{-}14)$$

Within their research, Cheng and Church prove that the principles of MEDel, SEDel, and EAdd contribute to a reduction of the bicluster's MSR. A tradeoff of time and accuracy is involved in the use of MEDel and SEDel. The former is more efficient because the expensive MSR computation, $O(MN)$, is performed after multiple element deletions. On the other hand, SEDel reduces the size loss inherent in the MEDel. Deletion is immediately halted after the bicluster's MSR falls below R. To balance the benefits of both deletion schemes, a valid value for α must be selected such that it minimizes runtime and MSR while maximizing bicluster size.

The Cheng & Church algorithm is summarized in Figure 4-1.

```
Input:      A, K, R, α
Output:     K biclusters of A
1. Replace unspecified values in A with random values
2. Biclustering = {}
3. For i from 1 to K
       a. Initialize Bicluster i to A
       b. Perform MEDel
       c. Perform SEDel
       d. Perform EAdd
       e. Report row and column membership of Bicluster i
       f. Mask bicluster i entries on A
```

Figure 4-1: Cheng and Church Biclustering

4.2.2 Justification for Parameter Settings

Cheng and Church set K to 100, R to 300 and α to 1.2. No justification was provided for the values of the first and last parameters. Nevertheless, the choice of R was based on the results of a random bicluster generation simulation, which was also run by Cheng and Church. The simulation results are presented in Table 4-1.

Table 4-1: MSR Distribution of Randomly Generated Yeast Biclusters

Rows (Genes)	Columns (Conditions)	Lowest MSR	Highest MSR	Peak MSR	Tail
3	6	10	6870	390	15.50%
3	17	30	6600	480	6.83%
10	6	110	4060	800	0.064%
10	17	24	3470	870	0.002%
30	6	410	2460	960	$< 10^{-6}$ %
30	17	480	2310	1040	$< 10^{-6}$ %
100	6	630	1720	1020	$< 10^{-6}$ %
100	17	700	1630	1080	$< 10^{-6}$ %

For each pair of dimensions listed in Table 4-1, one million biclusters were randomly generated. The third and forth columns corresponded to the lowest and highest MSRs found in the population. The peak value corresponded to the most common MSR value. The last column indicates the percentage of the population whose MSR was less than 300. Based on these trends, 300-biclusters, exhibiting more than 1700 entries, rarely occur by chance. Their high statistical significance makes them worthy of further inspection by biologists.

4.2.3 Time Complexity of CC

This algorithm exhibits a time complexity of $O(K(MN)(M+N))$. Each deletion or addition requires $O(MN)$ time due to MSR bicluster computation. In the worst case, $O(M+N)$ deletions and insertions could be performed to acquire the necessary bicluster.

4.3 FLOC Biclustering

4.3.1 Problem with CC Biclustering

Yang et al. show that the practice of inserting random values into the dataset (in CC) interferes with the biclustering process [9]. This action reduces the size of future biclusters.

4.3.2 Re-defining Bicluster Size and MSR

As a remedy, Yang et al. redefined the concept of bicluster size as *bicluster volume*. Bicluster volume measures the number of specified entries.

The equations for computing MSR have been modified to account for missing values. Assume row *i* and column *j* are elements of some bicluster *B*, where $B = (I,J)$. J' is a subset of J, representing the specified columns of row *i*. I' is a subset of *I*, containing the specified rows for column *j*. The element-wise averages of *i* and *j* are expressed in (4-15) and (4-16) respectively. The volume values of *i* and *j* are expressed in (4-17) and (4-18) respectively. As seen in (4-19), the residue of entry b_{ij} exhibits zero residue if it is unspecified. The MSR equations for *i*, *j*, and *B* have been re-defined in (4-20), (4-21), and (4-22) respectively.

$$\mu_{iJ} = \frac{1}{|J'|}\sum_{j\in J'} b_{ij} \tag{4-15}$$

$$\mu_{Ij} = \frac{1}{|I'|}\sum_{i\in I'} b_{ij} \tag{4-16}$$

$$\text{Volume}(i,B) = |J'| \tag{4-17}$$

$$\text{Volume}(j,B) = |I'| \tag{4-18}$$

$$r_{ij} = \begin{cases} 0, & \text{if } b_{ij} \text{ is null} \\ b_{ij} - \mu_{iJ} - \mu_{Ij} + \mu_B, & \text{otherwise} \end{cases} \tag{4-19}$$

$$\text{MSR}(i,B) = \frac{1}{|J'|}\sum_{j\in J'} r_{ij}^2 \tag{4-20}$$

$$\text{MSR}(j,B) = \frac{1}{|I'|}\sum_{i\in I'} r_{ij}^2 \tag{4-21}$$

$$\mathrm{MSR}(B) = \sum_{i\in I}\sum_{j\in J} r_{ij}^2 \tag{4-22}$$

Similar to the Cheng and Church algorithm (CC), the *FL*exible *O*verlapping *C*lustering (FLOC) algorithm strives to find *K* biclusters of maximal size and minimal MSR.

4.3.3 FLOC: Phase One

The FLOC algorithm is a two-stage process: first, bicluster initialization and second, action execution. Initially, each of the *K* biclusters is randomly assigned $M \times p$ rows and $N \times p$ columns, where $p > 1$. The random assignment is changed until all of the following conditions are met:

(i) The occupancy ratio (OR) of each row, and column, and the overall bicluster, must match or exceed that of the original matrix. This ratio measures the proportion of specified entries versus total entries within a row, column, bicluster, or matrix. Definitions are presented in Table 4-2.

Table 4-2: Variations of the Occupancy Ratio Equation

Type	OR Equation
Row *i*	$\mathrm{OR}(i, B) = \frac{\mathrm{Volume}(i)}{\lvert J\rvert}$
Column *j*	$\mathrm{OR}(j, B) = \frac{\mathrm{Volume}(j)}{\lvert I\rvert}$
Bicluster *B*	$\mathrm{OR}(B) = \frac{\mathrm{Volume}(B)}{\lvert I\rvert\lvert J\rvert}$
Matrix *A*	$\mathrm{OR}(A) = \frac{\mathrm{Volume(A)}}{MN}$

(ii) Each bicluster must also exhibit an MSR that is less than or equal to *R*.

(iii) Any other user-specified constraints, such as the maximum overlap between biclusters.

Within their experiments, Yang et al. only enforced constraints (i) and (ii). Their results suggested that only t_0 random row and column initializations are required to generate *K* acceptable biclusters, where t_0 is usually much smaller than $M + N$.

4.3.4 FLOC: Phase Two

After initializing all *K* biclusters, *t* iterations of the Phase Two process are executed until the biclustering of the current iteration is not an improvement over the biclustering from the previous iteration. Similar to t_0, *t* is smaller than $M + N$ [9].

During every phase two iteration, a list of possible actions are formulated and executed for each element in the original matrix. An action of element *i*, where $1 \leq i \leq M + N$, is defined removing or including an element from some bicluster *k*, where $1 \leq k \leq K$. Under the binary sequence representation, an action can be interpreted as toggling the value of entry *i* of bicluster sequence *k*. Although there are *K* possible actions for each element, only the most beneficial action can be executed for each element.

The gain (quality) of action involving element *i*, and bicluster *k*, can be expressed using:

$$\mathrm{GAIN}(i,k) = \frac{\mathrm{MSR}(i,k) - \mathrm{MSR}(i,k')}{\frac{R^2}{\mathrm{MSR}(i,k)}} + \frac{\mathrm{Volume}(i,k') - \mathrm{Volume}(i,k)}{\mathrm{Volume}(i,k)} \qquad (4\text{-}23)$$

In the equation, $\mathrm{MSR}(i,k)$ corresponds to the MSR of bicluster *k* before toggling the value of entry *i*. $\mathrm{Volume}(i,k)$ corresponds to the volume of bicluster *k* before toggling the membership of entry *i*. $\mathrm{MSR}(i,k')$ corresponds to the MSR of bicluster *k* after toggling the membership of entry *i*. $\mathrm{Volume}(i,k')$ corresponds to the volume of bicluster *k* after toggling the membership of entry *i*. *R* corresponds to a user-defined MSR threshold.

After generating the list of the best actions for each element, the actions are executed sequentially. After each action is executed, the average MSR and volume of the biclusters are measured. An action can be *blocked* (reversed) if any of the resulting biclusters violate the occupancy ratio, MSR, or user-defined bicluster requirements.

If the action is not blocked, and the average volume of the biclusters exceeds the best biclustering of the previous Phase Two iteration, this biclustering will be marked as an *improved* solution. The improved biclustering with the lowest MSR is then selected as the best biclustering of the current Phase Two iteration. The phase two process is repeated until no more improved solutions are found.

Subsequent actions become more prone to blockage. The highest-gain actions should be placed first. Nevertheless, Yang et al. felt that strictly ordering actions on gain would bind biclustering solutions to local optima. Thus, randomness also played a factor in list ordering.

Assume the LOA is initially ordered from the lowest to highest element index. Two elements, *i* and *j*, where the position of *i* is ahead of *j*, are switched if a random float (4-byte float) variable exceeds $g_{\text{swap}} = 0.5 + \frac{g_i - g_j}{2(G_H - G_L)}$. G_H and G_L refer to the highest and lowest gain of the list. This weighted random re-arrangement is performed $2(M + N)$ times on the list.

The FLOC algorithm, in its entirety, has been outlined below for the reader's convenience in Figure 4-2.

```
Input:      p, R, A, K, and other user-defined constraints
Output:     K biclusters of A

1. Initialize
   a. K biclusters of A which met MSR, volume, and any user-defined
      constraints.
   b. best_biclustering_volume = 0
   c. best_biclustering = initialized biclusters of step (1a)
   d. curr_best_biclustering_MSR = $+\infty$
   e. Set curr_best_biclustering to empty biclusters
2. While ( improved != true )
   a. Compute action for each element e, $1 \leq e \leq M + N$
   b. Put actions on action list
   c. Perform Weighted Random Re-arrangement on action list
   d. For each action f, from beginning to end of action list,
      1) Block f if resulting bicluster violates constraints
      2) Otherwise, if the average volume of biclustering f is greater than
         best_biclustering_volume
         a) curr_best_biclustering = biclustering f, if average MSR of
            biclustering f exceeds curr_best_biclustering_MSR
         b) ensure improved is true
   e. if (improved = true)
      1) best_biclustering = curr_best_biclustering
3. return best_biclustering
```

Figure 4-2: Summary of the FLOC Biclustering Algorithm

4.3.5 Time Complexity

The time complexity of the first phase is $O(t_0K(M + N))$.

For every insertion or deletion, each element-wise (row or column-wise) average is re-computed if the action introduced or removed entries. The time cost of updating any element-wise averages is at worst $O(M+N)$. The cost of updating MSR is still $O(MN)$ as the residue values for every element has to be re-computed. The bicluster-wise average will always be affected by any bicluster modifications. As there are $K(M + N)$ gain computations to perform, each iteration of Phase Two is $O(K (M + N)(MN))$.

Thus, the time complexity of FLOC biclustering is $O(tK(M + N)(MN))$.

4.3.6 Comparison with CC

At first glance, it appears that the Cheng and Church algorithm has a lower time cost than its predecessor. However, FLOC MSR computations are more efficient. Unspecified values are ignored during these operations. The experimental results also suggest that FLOC has a better time complexity and bicluster mining performance than its predecessor. Table 4-3 displays the relevant experimental statistics of both algorithms reported in [9].

Table 4-3: CC vs. FLOC Biclustering

Algorithm	Avg. Bicluster MSR	Avg. Bicluster Volume	Time Complexity
CC	204.29	1576.98	12 mins
FLOC	187.54	1825.78	6.7 mins

5 The bicACO Algorithm

BicACO is the first technique to use principles of ant colony optimization (ACO) to solve the biclustering problem. ACO is a form of optimization modeled after ant behavior. In order to understand bicACO, one must first understand the principles of natural ant behavior (NAB) and how NAB principles are generally incorporated into ACO algorithms.

5.1 Mechanics of Natural Ant Behavior

Ants of real-world colonies are simplistic creatures which randomly trace paths from their home colony to food sources in physical environments. In the search process, ants deposit pheromone on locations while traveling through them in order to mark the overall trail for future travel. Thus, a trail can be described as a set of sequentially traveled locations. The pheromone level of each location evaporates (decreases) over time. In order to compensate for evaporation, a location must receive frequent pheromone deposits through frequent travel. Although ant movements are random, they are more likely to travel to high-pheromone locations.

Shorter trails have shorter journey times. Ants are able to deposit pheromone on their locations in faster time. Thus, their locations' pheromone levels tend to be higher than those of longer trails. Since shorter paths have more high-pheromone locations, ants travel more frequently on them. Positive feedback occurs as travel frequency increases with deposit frequency, which in turn increases with higher pheromone levels. Thus, ants travel more frequently on shorter paths in the long run. The concept of pheromone-induced positive feedback is the key feature of all ACO algorithms [37].

5.2 How NAB Principles are Generally Incorporated in ACO Algorithms

5.2.1 Overview

In most ACO algorithms, the target problem must be specified as a combinational optimization (CO) problem. Using the notation of [38], any CO problem can be described as a set of *components* $P = \{c_1, \ldots, c_f\}$, where c_i (for all $1 \leq i \leq f$) is a distinct component. These algorithms search for K solutions, where each solution S_k (for all $1 \leq k \leq K$) is a subset of P. In most cases, K is a user-specified parameter. The objective of ACO is to approximate the highest-quality acceptable solution (HQAS) in at least one of these K solutions. The quality measure is specified by the target problem.

Solution S_k can be represented using binary f-dimensional coordinates, where a 1 for the i^{th} coordinate indicates the inclusion of component c_i; a 0 indicates exclusion. The search space of any CO problem is defined as the set of all possible subsets that can be generated from P. Hence, $SS(P) = \{S_1, \ldots, S_{2^n}\}$, where S_x is a unique subset of P (solution) and $1 \leq x \leq 2^f$.

Most ACO algorithms, including bicACO, exhibit a tri-phase structure. These phases include: initialization (INIT), solution construction (SC), and iterative improvement (I2) [37]. The phases are executed sequentially for T iterations. The connection between ACO and NAB will be established in each phase.

5.2.2 Initialization

During this phase, S_1 to S_K are initialized to some solution S_0. The settings of S_0 depend on the target problem and/or the researcher's intuition. The search space of the target problem serves as the analog of a natural ant's physical environment. Any solution in the search space corresponds to some location in the environment. Binary f-dimensional coordinates are used to describe solutions in the same manner that Cartesian coordinates are used to describe physical locations.

5.2.3 Solution Construction

Here, each ant Q_k is assigned the task of randomly adding or removing components from S_k until it becomes an acceptable solution. Since S_0 is analogous to the home colony location, the process of randomly modifying the solution is akin to ant travel. These modifications change the coordinates of the solution. An acceptable solution is analogous to the location of a food source because it is Q_k's desired destination. The ACO analog of a trail is defined as the set of solutions that are randomly "visited" by Q_k until S_k becomes acceptable.

The probability of selecting a component for addition is usually proportional to two factors: its pheromone level and its information heuristic measure. Component removal probabilities are inversely proportional to these values.

Unlike NAB, pheromone levels are not assigned to solutions (locations), but to components. This variation is due to the nature of ACO's objective. The purpose of all ACO algorithms is to find the highest-quality acceptable solution (the location of the best source of food), not the best (shortest) trail. Nevertheless, ACO uses pheromone-induced positive feedback to improve solution quality.

Due to the I2 phase (which is described in Section 5.2.3), the components of superior solutions receive higher pheromone deposits than those of inferior ones. These *beneficial* components exhibit higher pheromone levels. Since ants are more likely to include high-pheromone components in their solutions, more high-quality solutions will be constructed in subsequent iterations. Constructed solutions should eventually converge to the HQAS.

The information heuristic measure can be viewed as a supplement to pheromone levels. It is a problem-specific metric that evaluates the usefulness of adding a component to a solution. Thus, this measure will increase the likelihood of including appropriate components, enhancing pheromone-induced positive feedback.

5.2.4 Iterative Improvement

After the ants have constructed their solutions in the SC phase, the pheromone level of each component c_i is updated for the next iteration $t + 1$, where $1 \leq t \leq T$. This phase is designed to model the fluctuations of pheromone in NAB.

Although ACO algorithms update pheromone differently, most update equations account for the following factors: the evaporation rate of pheromone (which is a constant value), the pheromone level of c_i at iteration t, and the sum of the pheromone deposits given to c_i at iteration t. Ant Q_k deposits pheromone to c_i if it is included in S_k. This pheromone deposit is proportional to the quality of solution S_k. Thus, the components of better solutions are given high levels of pheromone, which in turn precipitates positive feedback.

Most ACO algorithms do not model the return journey (to the home colony) in natural ant behavior. The ants are re-positioned back to their colony during INIT of iteration t + 1. The tri-phase structure for ACO algorithms has been outlined in Figure 5-1.

```
Input:      $P$, $K$, $T$, $S_0$
Output:     $K$ solutions
1. For $t$ = [1, $T$]
   1. Initialization
      a. For each solution $S_k$
         i. $S_k = S_0$
   2. Solution Construction
      a. For each ant $Q_k$
         i. Randomly Remove + Add Elements Until $S_k$
            is acceptable
   3. Iterative Improvement
      a. Update all pheromone entries, accounting for
         pheromone deposits and evaporation
2. Return $K$ solutions from iteration $T$
```

Figure 5-1: Tri-Phase Structure of ACO Algorithms

5.3 Description of bicACO

ACO can be applied to the biclustering problem as it can be mathematically expressed as a CO problem: $P_{BIC} = \{1, \ldots, M + N\}$, where i (for all $1 \leq i \leq M + N$) refers to element i. Thus, the biclustering problem's components correspond to the original matrix's elements (rows or columns). $SS(P_{BIC}) = \{B_{P1}, \ldots, B_{P2^{M+N}}\}$. Each component subset B_{Px}, where $1 \leq x \leq 2^{M+N}$, denotes a unique bicluster. The binary sequence representation of B_{Px} can be derived by converting x – 1 into an unsigned binary number. For example, assume that M = 3 and N = 3.

Therefore, $B_{P1} = 000001$ and $B_{P64} = 111111$. Any bicluster can be represented using (M+N)-dimensional coordinates by corresponding its ith sequence entry to the ith coordinate.

BicACO is similar to many ACO algorithms as it utilizes the same tri-phase structure. Nevertheless, the algorithm has five distinguishing features:

(i) Higher levels of pheromone increase a component's likelihood of being excluded from a solution (bicluster).
(ii) The information heuristic measure is proportional to the probability of excluding an element (component) from a bicluster (solution).
(iii) Multiple pheromone tables are used.
(iv) All pheromone values must exceed 0 and cannot exceed 1.
(v) All pheromone entries of all tables must be initialized to 0.5 even before the execution of the INIT phase.

Feature (i) was an arbitrary choice made by Coelho et al. Feature (ii) was designed to measure the usefulness by excluding a component from a solution. Feature (iii) was used because the biclustering problem requires multiple optimal solutions, unlike many other CO problems. Features (iv) and (v) were taken from Blum and Dorigo's Hypercube Framework, a generic and robust ACO algorithm [39].

5.3.1 bicACO: Initialization

During initialization, each bicluster B_k (where $1 \leq k \leq K$) is initialized to bicluster B_A (the location of the home colony). B_A contains all the rows and columns of the original matrix. The choice of B_A's value was influenced by the CC algorithm.

5.3.2 bicACO: Solution Construction

Each ant Q_k is assigned the task of randomly removing elements (tracing a path through the physical environment) until B_k's MSR falls below a threshold R or until B_k's volume falls below a threshold V_m.

During any execution of SC, at most M+N element deletions are performed. Similar to CC's SEDel procedure, elements are deleted one at a time. The probability that some element i will be selected for deletion can be expressed as:

$$p_{iB_k} = \begin{cases} \dfrac{\tau_{iB_k}(t)\ \eta_{iB_k}}{\sum_{j \in B_k} \tau_{jB_k}(t)\, \eta_{jB_k}}, i \in B_k \\ 0,\ \text{otherwise} \end{cases} \tag{5-1}$$

$\tau_{iB_k}(t)$ denotes the pheromone entry value of element i of B_k's table at iteration t. BicACO also uses positive feedback to build better versions of B_k.

Beneficial elements of B_k tend to exhibit lower pheromone values than their detrimental counterparts. Since ants have a tendency to retain low-pheromone elements, Q_k will construct superior versions of B_k in later iterations, allowing the bicluster to converge to its optimal form.

η_{iB_k} denotes the information heuristic measure for element i with regards to B_k. The information heuristic measure originates from CC and corresponds to the MSR of element i relative to the other elements of B_k. This measure increases the probability of removing high-noise elements. η_{iB_k} can be expressed as:

$$\eta_{iB_k} = \frac{\mathrm{MSR}(i, B_k)}{\sum_{j \,\square B_k} MSR(j, B_k)} \tag{5-2}$$

After every deletion, the MSR and volume values of B_k are updated. If B_k's MSR drops below R or its volume drops below V_m, B_k has finally become acceptable. An acceptable bicluster is analogous to the location of a food source.

5.3.3 bicACO: Iterative Improvement

After solution construction, the pheromone entries of every table are updated for the next iteration to account for the effects of evaporation and pheromone deposits. For bicluster B_k's table, the only entries that are updated, correspond to the member elements of B_k. Assuming element i is included in B_k, its pheromone entry for iteration $t + 1$ can be expressed as:

$$\tau_{iB_k}(t+1) = \tau_{iB_k}(t) + e_v\left(\Delta\tau_{B_k}(t) - \tau_{iB_k}(t)\right) \tag{5-3}$$

e_v denotes the evaporation rate of all pheromone. $\Delta\tau_{B_k}(t)$ denotes the deposit for B_k's elements on iteration t. The latter variable can be expressed as:

$$\Delta\tau_{B_k}(t) = \frac{F(B_k)}{\sum_{1 \le l \le K} F(B_l)} \tag{5-4}$$

$F(B_k)$ measures the quality of bicluster B_k. The denominator of (5-4) represents the sum of the quality measures corresponding to all the acceptable biclusters formed on iteration t. Thus, B_k's pheromone deposit is defined by the ratio of its quality to the overall sum. Deposits of 0.5 or greater indicate that B_k is far superior to its counterparts.

The bicluster quality function, $F(B_k)$, is a signal-to-noise ratio. Volume can be viewed as the power level of the signal while MSR can be power level of the noise. Thus, $F(B_k)$ can be described as:

$$F(B_k) = \frac{\mathrm{Volume}(B_k)}{\mathrm{MSR}\ (B_k)} \tag{5-5}$$

Coelho et al. state that (5-3) precipitates positive feedback and increases exploration of the search space [10]. (5-3) is also based on the update rule of the Hypercube Framework [39]. A detailed outline of bicACO is presented in Figure 5-2.

```
Inputs:     K, T, R, V_m, e_v, A
Output:     K biclusters of iteration T
  1. Set all pheromone entries of all tables to 0.5
  2. For t = [1, T ]
       a. Initialization
            i. Initialize each bicluster B_k to A
       b. Solution Construction
            i. For each ant Q_k, where 1 ≤ k ≤ K,
                 1) Randomly remove some element that is included within
                    Bicluster B_k
                      a) The probability that some member element i will
                         be selected for deletion can be computed using
                         Equation (5-3).
       c. Iterative Improvement
            i. For each bicluster B_k, where 1 ≤ k ≤ K,
                 1) For each member element i of B_k
                      a) Compute $\tau_{iB_k}(t+1)$
  3. Return K solutions from iteration T
```
Figure 5-2: Outline of the bicACO algorithm

5.3.4 Time Complexity of bicACO

Each iteration of the solution construction phase is slower than the entire CC algorithm: only single element deletions are performed. Since the SC phase is performed for T iterations, bicACO's time complexity is O($TK(MN)(M+N)$). Thus, bicACO is much slower than its predecessors, CC and FLOC.

5.3.5 Experimental Performance of bicACO

Setting K = 100, T = 10, R = 180, V_m = 100, e_v = 0.2, and A to Yeast, bicACO produced biclusterings far superior to CC and FLOC. R (the bicluster MSR threshold) was set to 180 instead of 300 [10]. The value of T was set to minimize the runtime cost of the algorithm, while providing enough iterations for ant-optimized growth [11]. V_m was set to prevent the algorithm from returning empty biclusters. The evaporation rate was set to a low value, 0.2, in order to stabilize the changes to pheromone levels.

Table 5-1: Performance of various MSR biclustering algorithms on Yeast

Algorithm	**Avg. Bicluster MSR**	**Avg. Bicluster Volume**	**Runtime (relative to CC)**
CC	204.29*	1576.98*	1.0
FLOC	187.54*	1825.78*	**0.56****
bicACO	**176.15 ± 1.37****	**2725.61 ± 105.53****	24***

* - results were taken from [9]
** - results were taken from [10]
*** - runtime of bicACO was taken from [11]

Although bicACO results were superior to its predecessors, its runtime was far slower. Based on the results of Yang et al., FLOC required around half (0.56) of CC's runtime to produce its final results. According to [11], the runtime of bicACO was 24 times greater than CC.

6 SNAP Biclustering

SNAP biclustering is the direct descendant of bicACO because it uses the same tri-phase structure. SNAP also retains most of the distinguishing features of its predecessor. Nevertheless, the details of each phase are different.

INIT has been optimized for speed. FLOC's initialization methodology is used in the place of CC. The SC phase has been designed to increase coverage of the search space by using a more flexible bicluster modification strategy. This phase also reduces algorithm runtime by omitting the computation of the information heuristic measure. Pheromone values in SNAP correspond to higher element addition probabilities. Thus, the update equation of the I2 phase has been re-formulated to reflect this change.

6.1.1 Initialization: FLOC Phase One

The slow performance of bicACO is partially due to the nature of its initialization procedure. Like CC, bicACO assigns all the rows and columns of the original matrix to each bicluster. Thus, elements must be removed until the corresponding bicluster attains an acceptable MSR. Since bicACO only removes one element prior to re-computing MSR, much time is wasted calculating the noise-levels of excessively large biclusters (those with volumes close in value to the original matrix).

To avoid this, SNAP follows the methodology of FLOC's first phase. Each bicluster is initialized to small and different row and column subsets. However, more parameters are used to improve initialization. All of the parameters have been listed in Table 6-1.

Table 6-1: SNAP's Initialization Parameters

i_r	Initial number of rows
i_c	Initial number of columns
i_R	Maximum initial MSR
i_d	Maximum occupancy ratio
i_o	Maximum allowable overlap

Thus, each initialized bicluster B_k is randomly assigned i_r rows and i_c columns, where $1 \leq k \leq K$, $2 \leq i_r \leq M$ and $2 \leq i_c \leq N$. B_k should exhibit an MSR no greater than i_R, where $0 < i_R \leq R$ and R is a user-specified MSR threshold. It must have an occupancy ratio no smaller than i_d and should exhibit no more than i_o percentage of *overlap* with bicluster B_l , where $0\% \leq i_o \leq 100\%$ and $1 \leq l \leq k - 1$. Overlap is defined as the percentage of specified entries common between a pair of biclusters. New row and column subsets are repetitively generated until B_k meets all of the constraints imposed by the parameters. This process can get stuck if parameter values are too restrictive.

One should note that the initialized biclusters are only generated in the first iteration. In subsequent iterations, each bicluster is reassigned its initial form from iteration 1. In terms of natural ant behavior, this initialization procedure is equivalent to the use of multiple ant colonies. Each ant begins from a different location in the search space.

6.1.2 Solution Construction: Bicluster Single Nucleotide Polymorphisms

Within this phase, ant modifications (movements) are far less restricted than their bicACO counterparts. Elements can be repetitively included and/or excluded from a bicluster on the same iteration of SC. This increased modification flexibility promotes greater coverage of the search space.

During each execution of the SC, each ant Q_k attempts to perform G modifications to bicluster B_k. Within each attempted modification, bicluster B_k is viewed as a binary sequence. Sequence entry i is selected for modification, where i a randomly chosen number and $1 \leq i \leq M+N$. Another random number f is generated, where $0.0 \leq f \leq 1.0$.

The probability of adding element i increases with higher pheromone levels. A modification is executed if sequence entry i is 0 and $f \leq \tau_{iB_k}(t)$. The probability of removing element i increases with lower pheromone. Execution also proceeds if entry i is 1 and $f > \tau_{iB_k}(t)$. Nonetheless, a modification can be *reversed* if the resulting bicluster violates one of the following constraints:

i) B_k's MSR must not exceed R
ii) The volume of B_k must not fall below V_m
iii) B_k must have two or more rows
iv) B_k must have two or more columns

A reversal does not count as a separate alteration. G modifications can be *executed* and reversed by Q_k in an iteration of SC. B_k's finalized form can be viewed as the location of a food source in natural ant behavior.

The modification strategy of the SC algorithm was inspired by Single Nucleotide Polymorphisms. During cell reproduction, a cell's genome sequence is copied. As the duplication process is faulty, random entries are given incorrect values. These entries are known as SNPs. Hence, each bicluster modification can be viewed as the analog of an SNP because a randomly selected entry is being modified. Within humans, SNPs affect susceptibility to certain diseases and bodily responses to different chemicals. Thus, these sequence changes can be beneficial or harmful [40] [41]. Bicluster modifications are similar because they can either improve or reduce bicluster quality.

BicACO's information heuristic measure (IHM) is not used to influence element removal probabilities. This measure requires one to compute the MSRs of every element included in a bicluster. The equations for row and column-wise MSRs are expressed as (4-18) and (4-19) respectively. This IHM is omitted from the algorithm because it is computationally expensive for large biclusters.

An IHM for element addition can be derived from CC's EAdd procedure. (BicACO's IHM was derived from CC's SEDel.) This heuristic measures every excluded element's potential MSR. The equations for a row and column's potential MSR are expressed as (4-13) and (4-14) respectively. However, SNAP does not use this measure because it is expensive for small biclusters.

Although omitting the information heuristic measure may reduce solution quality as discussed in Chapter 5.2.2, improving runtime was the first priority. Furthermore, quality loss can be rectified by fine-tuning parameter values, particularly those of *G*. Chapters 7 and 8 discuss the effects of different parameters on SNAP's performance.

The pheromone values of beneficial elements tend to exceed those of detrimental elements. Since ants are inclined to include high pheromone elements, the quality of constructed biclusters improves in later iterations.

6.1.3 Iterative Improvement

Within this phase, pheromone values are proportional to element addition probabilities. In bicACO, pheromone levels are interpreted in the opposite fashion: higher levels correspond to higher removal probabilities.

BicACOs update equation is designed to increase pheromone levels whenever $\tau_{iB_k}(t) < \Delta\tau_{B_k}(t)$. Coelho et al. state that this mechanism prevents bicACO from getting stuck at local optima [42].The update rule of SNAP also incorporates this feature by decreasing levels under the same condition. The pheromone entry value, of element *i* on bicluster B_k, can be expressed as:

$$\tau_{iB_k}(t+1) = \tau_{iB_k}(t) + e_v\big(\tau_{iB_k}(t) - \Delta\tau_{B_k}(t)\big) \tag{6-1}$$

In order to verify that the formulation of (6-1) is appropriate, SNAP's performance using (6-1) is compared with that of (6-2) in Chapter 7. (6-2) does not have the local optima mechanism as it increases pheromone whenever $\tau_{iB_k}(t) < \Delta\tau_{B_k}(t)$. Experimental results suggest that (6-1) is critical for precipitating positive feedback: bicluster quality improves for later iterations. The results also suggest that (6-2) cause a progressive decline in bicluster quality.

$$\tau_{iB_k}(t+1) = \tau_{iB_k}(t) + e_v\left(\Delta\tau_{B_k}(t) - \tau_{iB_k}(t)\right) \tag{6-2}$$

The pseudocode for the SNAP algorithm is presented in Figure 6-1.

```
Input:      $e_v$, $G$, $T$, $K$, $R$, $V_m$, $A$, $i_r$, $i_c$, $i_o$, $i_d$, $i_R$
Output:     $K$ biclusters

1. For $t$ = [1, $T$]
    a. Initialization:
        i. If $t$ = 1, for each bicluster $B_k$,
            1) Randomly assign a pair of rows and columns
               that meet initialization parameter constraints
            2) This pair will be denoted as the initial form of
               $B_k$
        ii. Else, assign $B_k$ to its initial form (which was generated
            in the first iteration)
    b. Solution Construction:
        i. For each bicluster $B_k$ and for $g$ = [1, $G$]
            1) $i$ = random number between [1, $M+N$]
            2) $f$ = random number in range of [0, 1]
            3) if ( $i \notin B_k$ AND $f \leq \tau_{iB_k}(t)$)
                   Add $i$ to $B_k$
            4) else if ($i \in B_k$ AND $f > \tau_{iB_k}(t)$ )
                   Remove $i$ from $B_k$
            5) Undo modification if any one of the following
               conditions is met:
                   a) MSR($B_k$) > R
                   b) Vol($B_k$) < $V_m$
                   c) $B_k$ has 1 or fewer rows
                   d) $B_k$ has 1 or fewer columns
    c. Iterative_Improvement ( )
        i. Update pheromone entries according to
           update rule (6-1).
2. Return $K$ biclusters formed on iteration $T$
```

Figure 6-1: Outline of the SNAP Biclustering Algorithm

7 Performance of SNAP for the Yeast Dataset

7.1 Description of Parameter Settings and Performance Statistics

SNAP was run on Yeast using a wide variety of parameter sets (PSs). The algorithm's performance was compared for these PSs in order to determine if its update rule promoted positive feedback. These comparisons were also used to identify a configuration of parameters that produces better results. The values of each PS have been listed in Table 7-1.

Table 7-1: Parameter settings for different executions of SNAP

Parameter Set	Parameter Values
-1	K = 100, e = 0.2, g = 1000, T = 10, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, Iterative Improvement – Equation(6-1)
0	K = 100, e = 0.2, g = 1000, T = 10, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, **No Iterative Improvement**
1	K = 100, e = 0.2, g = 1000, T = 10, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, **Iterative Improvement - Equation (6-2)**
2	K = 100, **e = 0.4**, g = 1000, T = 10, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, Iterative Improvement - Equation (6-1)
3	K = 100, **e = 1.0**, g = 1000, T = 10, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, Iterative Improvement - Equation
4	K = 100, **e = 1.0**, g = **2901**, T = 10, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, Iterative Improvement - Equation (6-1)
5	K = 100, e = 0.2, g = 1000, T = 10, R = 300, V_m = 5, i_r = 3, **i_c = 17**, i_o = 30%, i_d = 0.9, i_R = 90, Iterative Improvement - Equation (6-1)
6	K = 100, e = 0.2, g = 1000, T = 10, R = 300, V_m = 5, i_r = **2**, i_c = **3**, **i_o = 10%**, i_d = 0.9, **i_R = 15**, Iterative Improvement - Equation (6-1)
7	K = 100, **e = 1.0**, g = 1000, T = 10, R = 300, V_m = 5, i_r = **2**, i_c = **3**, **i_o = 10%**, i_d = 0.9, i_R = 15, Iterative Improvement - Equation (6-1)
8	K = 100**, e = 1.0**, **g = 2901**, T = 10, R = 300, V_m = 5, i_r = **2**, i_c = **3**, **i_o = 10%**, i_d = 0.9, i_R = 15, Iterative Improvement - Equation (6-1)
9	K = 100, **e = 1.0**, g = **2901**, T = 10, R = **180**, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, Iterative Improvement - Equation (6-1)

SNAP was run 5 times for each PS. The following metrics were utilized to evaluate the results (biclusters) of each run: average MSR, average initial volume, average final volume, average overlap, coverage, and runtime. These metrics have been defined in Table 7-2.

Table 7-2: Description of Performance Statistics

Average MSR	Average MSR represents the average MSR levels of all the biclusters returned by the run. This metric is not reported with units [6] [9] [10]. Regardless of the parameter set, it was observed that MSR values rarely fluctuate. Although this metric will be presented, it will not be used for performance comparisons.
Average Initial/Final Volume	Average initial volume refers to the mean bicluster volume for the first iteration of each run. The average final volume refers to the mean for the last iteration (the returned results). Similar to MSR, this quantity is unit free. The disparity between these metrics is used to determine the effectiveness of ant optimization. Average final volume is also used as the primary measure of SNAP's performance, particularly during comparisons with its predecessors.
Coverage	Coverage refers to the percentage of specified entries (of the original matrix) that is occupied by the results of each run. This criterion should be maximized in order to avoid redundancy.
Average Overlap	Average overlap is the percentage of common specified entries between every unique pair of biclusters from the results of each run. This value should be minimized in order to avoid the production of redundant results.
Runtime	Runtime refers to the time elapsed from the beginning of the run's execution to its termination. All values are reported in seconds.

Tables 7-3 to 7-7 report each evaluation statistic as the mean of all five runs for each PS. The accompanying standard deviation values indicate the square root of the variance among the runs.

7.2 Evaluating the Effectiveness of the Pheromone Update Rule

In order to verify that SNAP's update rule was formulated correctly, the performance of parameter set 1 was compared to PS0 and PS-1. PS0 omitted the iterative improvement step, and PS-1 used (6-2) instead of (6-1) as the pheromone update rule.

The results of PS1 showed a gradual improvement in biclustering volume. As illustrated in Table 7-3, PS1's average initial and average final volumes were 1188.23 and 1685.42, respectively. The results of PS0 showed no improvement while PS-1's results indicated progressive decline. The average initial and final volumes of PS0 were 1160.48 and 1162.66 respectively. PS-1's corresponding values were 1192.45 and 906.29.

Table 7-3: Performance of SNAP for parameter sets 1, 0 and -1 for the Yeast dataset

Parameter Set	1	0	-1
Avg. MSR	298.96 ± 0.09	298.01 ± 0.51	296.68 ± 1.19
Avg. Initial Volume	1188.23 ± 24.32	1160.48 ± 25.99	1192.45 ± 17.21
Avg. Final Volume	1685.42 ± 40.34	1162.66 ±18.25	906.29 ± 14.35
Coverage (%)	79.89 ± 0.12	75.73 ± 0.22	72.17 ± 0.60
Avg. Overlap (%)	5.78 ± 0.20	3.88 ± 0.00	2.91 ± 0.00
Runtime (s)	92.49 ± 1.73	76.02 ± 1.16	64.59 ± 0.42

Figure 7-1 elucidates the variations of volume growth in greater detail. Here, the average bicluster volumes are plotted for each iteration and parameter set. Similar to Tables 7-3 – 7-7, the reported average bicluster volume corresponds to the mean of all five runs for each PS.

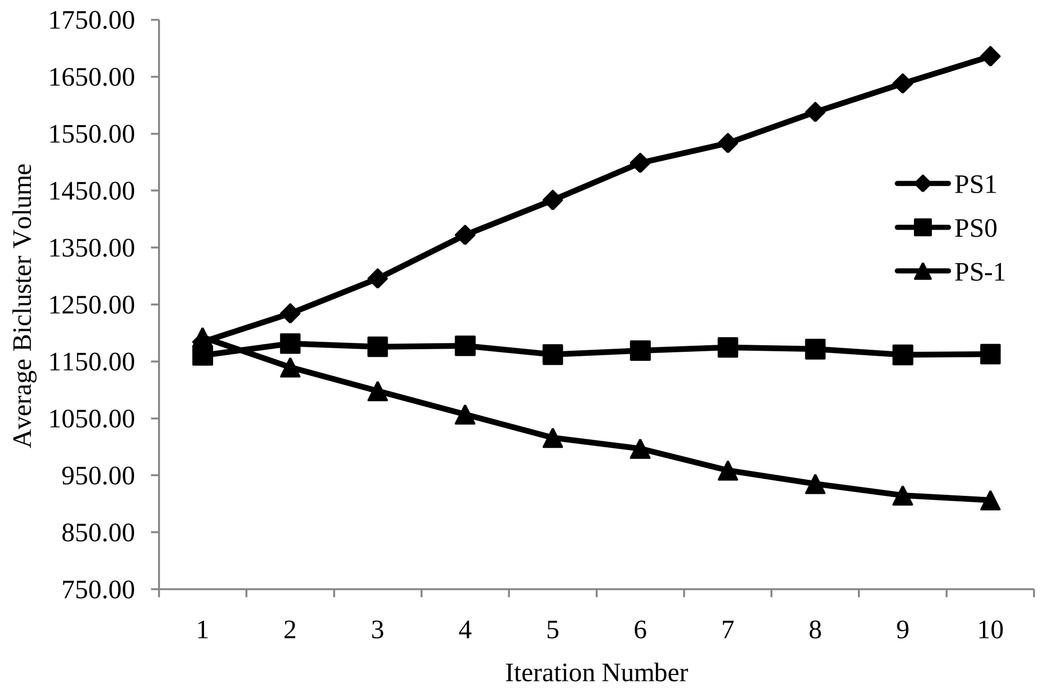

Figure 7-1: Performance comparison for SNAP between parameter sets 1, 0, and -1

A Welch's two-sample t-test indicated that the bicluster volumes of PS1 were statistically significant from PS0 and PS-1. There was less than a 0.001% likelihood that the average bicluster volumes of PS0 and PS1 were equal. This low p-value was also present between the volume averages of PS1 and PS-1, and between those of PS0 and PS-1.

The values of the other metrics (average coverage, overlap, and runtime) were not as drastically different between the aforementioned PSs. These values have also been listed in Table 7-3. The results of PS1 exhibited the highest coverage of the matrix: 79.89% in comparison to 75.73% of PS0 and 72.71% of PS-1. Nevertheless, PS1 had the worst overlap and runtime values: 5.78% and 92.49s respectively. PS0 had an average overlap of 3.88% and an average runtime of 76.02s. The corresponding values for PS-1 were: 2.91% and 64.59s.

The slower runtimes of PS1 were due to increased volume of its biclusters: the MSR computation increases with higher volume. Large biclusters tend to have greater overlap values. PS1's values were acceptable because they were better than bicACO's. The biclusters returned by SNAP's predecessor exhibited 62.47% coverage and 18.40% overlap. Thus, the results of Table 7-3 and Figure 7-1 show that (6-1) promoted positive feedback, the absence of an update rule promoted no feedback, and (6-2) promoted negative feedback.

7.3 Optimizing Parameter Settings

This analysis determined better settings for each of the following parameters: initialization parameters, evaporation rate (e_v), number of attempted modifications per iteration (G), and the number of iterations (T).

7.3.1 Initialization Parameters

In order to evaluate the effect of initialization on SNAP, the initialization parameters were varied between PS1, PS5, and PS6. Much care was taken when selecting parameter values because it was increasingly difficult to randomly generate low-noise large biclusters. Thus, the initial MSR threshold (i_R) was increased with higher values of initial volume. However, i_R was always set below R, which in turn was set to 300 (except in PS9).

(i) Within PS1, each bicluster was initially given 3 rows and 6 columns. Since the occupancy ratio threshold was equal for all parameter settings, 0.9, the minimum volume was 17. The initial MSR threshold was set to 30. Each pair of biclusters was prohibited from exhibiting more than 30% overlap.

(ii) Within PS5, biclusters were initialized to larger sizes: 3 rows and 17 columns. The minimum initial volume threshold was 46. Consequently, the initial MSR threshold was increased to 90. However, the initial overlap threshold was maintained at 30%.

(iii) Within PS6, biclusters were initialized to smaller sizes: 2 rows and 3 columns. The MSR and overlap thresholds were set to 15 and 10% respectively.

Initializing biclusters to larger volumes increased average initial volume and average final volume. Nevertheless, this action decreased coverage, and increased average overlap and runtime. As listed in Table 7-4, the average initial and final volume pairs of the parameter sets were: (1188.23, 1685.42) for PS1, (1417.59, 1919.44) for PS5, and (835.56, 1222.66) for PS6. The coverage, average overlap, and runtime triplets were: (79.89%, 5.78%, 92.49s) for PS1, (44.79%, 15.03%, 228.88s) for PS5, and (86.61%, 3.07%, 60.73s) for PS6.

The biclusters of PS5 exhibited unusually low coverage, and a relatively high level of average overlap, indicating the presence of high redundancy. Thus, the initialization settings of PS5 were not incorporated in other parameter sets.

Table 7-4: Performance of SNAP for parameter sets 1,5, and 6 with the Yeast dataset

Parameter Set	1	5	6
Avg. MSR	298.96 ± 0.09	298.67 ± 0.16	298.16 ± 0.13
Avg. Initial Volume	1188.23 ± 24.32	1417.59 ± 15.27	835.56 ± 17.98
Avg. Final Volume	1685.42 ± 40.34	1919.44 ± 28.61	1222.66 ± 14.97
Coverage (%)	79.89 ± 0.12	44.79 ± 0.38	86.61 ± 0.27
Avg. Overlap (%)	5.78 ± 0.20	15.03 ± 0.00	3.07 ± 0.00
Runtime (s)	92.49 ± 1.73	228.88 ± 24.49	60.73 ± 0.74

7.3.2 Evaporation Rate (e_v)

Based on the results of PS1, PS2, and PS3, overall volume growth increased with higher evaporation rates. In PS1, the evaporation rate was set to 0.2. In PS2, this rate was set to 0.4. The evaporation rate was then set to its maximum value 1.0 in PS3.

As listed in Table 7-5, all PSs exhibited similar average initial volumes: 1188.23 (PS1), 1183.92 (PS2), and 1181.03 (PS3). However, average final volumes were significantly different: 1685.42 (PS1), 1978.05 (PS2) and 2204.86 (PS3). The increase in volume was also correlated with increases in coverage, average overlap, and runtime: (79.89%, 5.78%, 92.49s) for PS1, (81.75%, 6.81%, 107.22s) for PS2, and (83.18%, 7.52%, 121.94s) for PS3.

The benefits of higher evaporation rates outweighed their costs. The volume increase of PS2 and PS3 was 67% and 87% respectively. The PS1 results only exhibited a 42% increase. The coverage rates of PS2 and PS3 were 2.3% and 4.1% greater than PS1. Even though the average overlap rates were higher for PS2 and PS3, their values were still much lower than bicACO (18.40%). The runtime of PS3, 121.94s, was far smaller than bicACO's runtime, 11853.00s. Thus, the evaporation rate was set to 1.0 in the next section of the analysis. Using this parameter setting, pheromone update rule (6-1) can be reduced to:

$$\tau_{iB_k}(t) = 2\tau_{iB_k}(t) - \Delta\tau_{iB_k}(t) \qquad (7\text{-}1)$$

Table 7-5: Performance of SNAP for parameter sets 1,2, and 3 for the Yeast dataset

Parameter Set	1	2	3
Avg. MSR	298.96 ± 0.09	299.00 ± 0.09	299.11 ± 0.08
Avg. Initial Volume	1188.23 ± 24.32	1183.92 ± 25.85	1181.03 ± 20.95
Avg. Final Volume	1685.42 ± 40.34	1978.05 ± 40.82	2204.86 ± 25.71
Coverage (%)	79.89 ± 0.12	81.75 ± 0.46	83.18 ± 0.41
Avg. Overlap (%)	5.78 ± 0.20	6.81 ± 0.00	7.52 ± 0.00
Runtime (s)	92.49 ± 1.73	107.22 ± 2.52	121.94 ± 4.28

7.3.3 Number of Attempted Modifications (G)

Based on the performance comparisons of PS3 and PS4, along with PS7 and PS8, algorithm performance increased with higher values of G. The performance of these PSs is presented in Tables 7-6 and 7-7.

Within PS3, the initialization parameters were set to those of PS1, evaporation rate was set to 1.0, and G was set to 1000. The settings of PS4 were identical to PS3 except for G. Here, this parameter was set to M+N (which is 2901 in Yeast). PS7 utilized the same settings as PS3 except it also used the initialization parameters of PS6. PS8 was identical to PS7 except G's value was also set to 2901.

Table 7-6: Performance of SNAP for parameter sets 4 and 5 for the Yeast dataset

Parameter Set	3	4
Avg. MSR	299.11 ± 0.08	299.75 ± 0.03
Avg. Initial Volume	1181.03 ± 20.95	2248.14 ± 64.22
Avg. Final Volume	2204.86 ± 25.71	5387.91 ± 39.17
Coverage (%)	83.18 ± 0.41	90.04 ± 0.25
Avg. Overlap (%)	7.52 ± 0.00	18.31 ± 0.00
Runtime (s)	121.94 ± 4.28	979.38 ± 32.26

Table 7-7: Performance of SNAP for Parameter sets 7 and 8 for the Yeast dataset

Parameter Set	7	8
Avg. MSR	298.65 ± 0.08	299.48 ± 0.25
Avg. Initial Volume	837.00 ± 15.49	1700.52 ± 12.40
Avg. Final Volume	1610.87 ± 35.23	3930.04 ± 55.14
Coverage (%)	91.17 ± 0.46	97.28 ± 0.00
Avg. Overlap (%)	4.03 ± 0.00	10.07 ± 0.46
Runtime (s)	86.40 ± 1.67	656.52 ± 5.79

The average initial volume and average final volume statistics of PS4 were at least 100% greater than PS3. PS4's average coverage was 8.2% greater than PS3. Although PS4's overlap rate (18.31%) and runtime (979.38s) were much higher than those of PS3 (7.52% and 121.94s), PS4's overlap rate was approximately equal bicACO's (18.40%). PS4's runtime was also acceptable as it was 90% lower than bicACO. PS8's results also dominated PS7's in the same fashion. The performance differences between PS4 and PS8 were also consistent with the findings of Chapter 7.3.1. Initializing biclusters to smaller volumes and MSRs resulted in lower average initial and final volumes, but also resulted in higher coverage, lower average overlap, and lower runtime values.

7.3.4 Number of Iterations (*T*)

The value of *T* remained at 10 for PS-1 to PS9s. This setting originated from bicACO. In order to determine if this value was appropriate, SNAP's convergence behavior was examined. Here, behavior refers to bicluster quality, mainly the average bicluster volume per iteration. As seen in previous Chapters 7.3.2 and 7.3.3, average bicluster volumes were positively correlated with coverage and negatively correlated with average overlap and runtime.

In order to analyze convergence, *T* was set to 100, much higher than original setting. Average bicluster volumes were measured for parameter sets 10 to 15. Among these PSs, e_v and/or *G* values were varied. Table 7-8 presents the settings of each PS. The average bicluster volume per iteration of each PS is presented in Figure 7-2.

Table 7-8: Parameter settings for different executions of SNAP

Parameter Set	Parameter Values
10	K = 100, e_v = **0.2**, g = **1000**, ***T* = 100**, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, Iterative Improvement – Equation (6-1)
11	K = 100, e_v = **0.4**, g = **1000**, ***T* = 100**, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, Iterative Improvement – Equation (6-1)
12	K = 100, e_v = **1.0**, g = **1000**, ***T* = 100**, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, Iterative Improvement – Equation (6-1)
13	K = 100, e_v = **0.2**, g = **2901**, ***T* = 100**, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, Iterative Improvement – Equation (6-1)
14	K = 100, e_v = **0.4**, g = **2901**, ***T* = 100**, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, i_R = 30, Iterative Improvement – Equation (6-1)
15	K = 100, e_v = **1.0,** g = **2901**, ***T* = 100**, R = 300, V_m = 5, i_r = 3, i_c = 6, i_o = 30%, i_d = 0.9, ***i*** $\boldsymbol{i_R}$ **= 30**, Iterative Improvement – Equation (6-1)

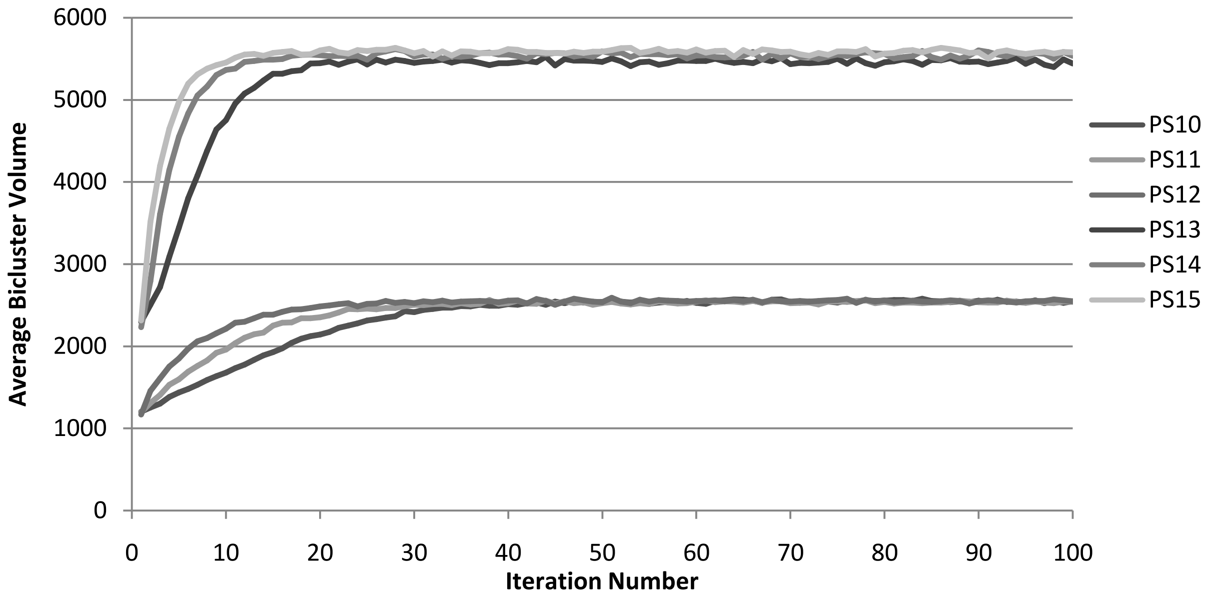

Figure 7-2: Convergence behavior of SNAP under varying e_v and G

Higher values of G and evaporation rate resulted in earlier convergence. When G was set to 1000, SNAP required at least 30 iterations to converge using an evaporation rate of 0.2. Convergence occurred in 20 iterations using evaporation rates of 0.4 and 1.0. When G was increased to 2901, convergence occurred in 20 iterations despite a low evaporation rate (0.2). Using evaporation rates of 0.4 and 1.0, SNAP converged in approximately 10 iterations. Additionally, the convergence value dramatically increased with higher values of G. Thus, the default setting of T (T = 10) was appropriate for higher values of G and e_v.

7.4 Performance Comparison with Predecessors

Although the results of PS4 and PS8 yielded impressive average final volumes, their MSR values (around 300) were inferior to those of bicACO (172.26). Thus, the value of R was set to 180 for PS9. The initialization parameters of PS1, high evaporation rate (1.0), and high value of *G* (2901) were incorporated into PS9. The table below compared the performance of SNAP against its predecessors, CC, FLOC, and bicACO.

Table 7-9: SNAP vs. Predecessors

Algorithm	SNAP: PS9	bicACO	FLOC	CC
Avg. MSR	179.80 ± 0.02	**172.26 ± 0.99**	187.54*	204.29*
Avg. Final Volume	**3040.57 ± 75.37**	2720.35 ± 59.88	1825.78*	1576.98*
Coverage (%)	**69.73 ± 0.75**	62.47 ± 0.01	N/A	N/A
Avg. Overlap (%)	**14.72 ± 0.08**	18.40 ± 0.00	N/A	N/A
Runtime (s)	547.39 ± 14.84	11853.00 ± 235.28	**275.73 ****	493.88 ******

* - these results were taken from [9]
** - runtime estimations of CC and FLOC came from [11]

The bicACO results were executed (for five times) using the same computer architecture as all of the SNAP runs. The FLOC and CC results were taken from [9]. The runtimes of [9] could not be directly reported due to the difference in computer architectures of experimentation. Thus, their runtimes were re-weighted using the bicACO to CC runtime difference (which in turn was reported in [11]).

SNAP clearly outperformed its predecessors. Although its MSR was slightly greater than bicACO (4.4%), SNAP's average final volume was 12% greater. These differences in volume were statistically significant. A Welch two-sample t-test indicated these averages have less than a 0.001% likelihood of being equal. Furthermore, SNAP exhibited a significantly greater coverage rate (7.6%) than bicACO. The average overlap of SNAP was 4% lower than bicACO. SNAP exhibited runtimes 22 times faster than bicACO, making the algorithm comparable in speed to CC and FLOC.

7.5 Overlap Analysis of SNAP & bicACO

The biclusters, generated by SNAP using PS9, were superior to bicACO as indicated by Table 7-9. This suggested that the results of SNAP were more worthy of further examination by biologists.

Nevertheless, the results of bicACO could still be important if they exhibited low overlap with those of SNAP. A low overlap value suggests that bicACO's biclusters corresponded to biological processes that were not mapped out by SNAP's results. This section determined if bicACO's biclusters were still relevant for analysis by determining if they were "different" from SNAP.

All the biclusters, generated by SNAP using PS9 regardless of run, were grouped in one large (500-member) population. This process was also repeated for bicACO's biclusters. For each member of SNAP's population, the overlap rate was measured for each member of bicACO's population. The mean overlap, between any pair of SNAP and bicACO biclusters, was 15.53% ± 9.81%. As a baseline measure, the overlap rates were also computed for each distinct pair of biclusters from SNAP's populous, resulting in a mean of 14.76% ± 7.76%. Based on these figures, bicACO's biclusters were highly similar to SNAP's. Thus, there was a low likelihood that bicACO's results could be used to find unique biological processes.

8 Performance of SNAP for MovieLens Dataset

SNAP was also applied to MovieLens in order to determine the algorithm's potential for collaborative filtering applications. Due to the sparseness of the dataset, the occupancy ratio of a bicluster must be kept in check. According to [8], a bicluster's occupancy ratio should be kept high in CF applications.

Thus, this algorithm enforced another constraint during the SC phase for this dataset: each bicluster must exhibit an occupancy ratio (OR) greater than or equal to 0.9. Using the findings of Chapter 7.3, e_v was set to 1.0, G was set to 2625 (the number of combined rows and columns in MovieLens), and T was set to 10.

Since MovieLens was sparse, the biclusters were initialized to extremely small volumes due to the difficulty of randomly generating high occupancy biclusters. Thus, all initial biclusters were assigned 2 rows and 2 columns. They also exhibited MSRs no greater than 0.3. No pair of biclusters were allowed to exhibit more than 20% overlap. Similar to [8], R was set to 1.5. As shown in Table 8-1, these aforementioned parameter settings were incorporated in PS15 and PS16.

Table 8-1: Parameter Settings for different executions of SNAP

Parameter Set	Parameter Values
15	K = 100, e = 1.0 g = 2625, T = 10, R = 1.5, V_m = 5, i_r = 2, i_c = 2, i_o = 0.2, i_d = 1.00, i_R = 0.3 Iterative Improvement – Equation (6-1) **Fixed Occupancy Ratio Constraint**
16	K = 100, e = 1.0 g = 2625, T = 10, R = 1.5, V_m = 5, i_r = 2, i_c = 2, i_o = 0.2, i_d = 1.00, i_R = 0.3 Iterative Improvement – Equation (6-1) **Incrementing Occupancy Ratio Constraint**

Although the values of PS15 were identical to PS16, PS15 utilized a fixed occupancy ratio constraint while PS16 allowed the constraint value to vary per iteration. Since returned biclusters were required to have an OR of 0.9, PS15 forced the biclusters to adhere to this constraint for all iterations. It was suspected PS15's rigid constraint would limit exploration of the search space. Within PS16, all biclusters were required to have an occupancy ratio of at least 0.7 on the first iteration. This value is increased by 0.05 for successive iterations until the constraint value reached 0.9.

The termination constraints of bicACO were modified to account for MovieLens' sparseness. The algorithm halted single element deletion if one of the following conditions had been met:

(i) the MSR of the bicluster fell below *R* and the bicluster's OR was 0.9 or greater
(ii) the bicluster's volume fell below V_m

The results of the MovieLens experiments were presented in Table 8-2, SNAP performed better for PS16. Although PS16's average MSR is 9.8% greater than PS15, PS16's average final volume is 89% greater. PS16 also exhibited far greater average coverage (+7.14%) and a lower overlap rate (-1.24%) than PS15. Although PS16's runtime is 532% greater than PS15, it is only 3.7% of bicACO's.

Regardless of the occupancy ratio constraint, SNAP outperformed bicACO in every measure except for MSR and overlap. However, bicACO's superior average MSR and average overlap statistics are irrelevant because the biclusters exhibited an extremely low occupancy ratio, 0.0032. Thus, SNAP excelled in the analysis of CF datasets. The performance values of bicACO, PS15, and PS16 were presented in Table 8-2.

Table 8-2: bicACO vs. SNAP for the MovieLens dataset

Algorithm	bicACO	SNAP: PS15	SNAP: PS16
Avg. MSR	**0.11 ± 0.00**	0.41 ± 0.02	0.45 ± 0.01
Avg. Final Volume	96.0 ± 0.41	158.55 ± 6.63	**300.41 ± 14.40**
Avg. Coverage (%)	8.18 ± 0.11	15.68 ± 0.45	**22.82 ± 0.00**
Avg. Overlap (%)	**0.02 ± 0.00**	1.27 ± 0.18	0.03 ± 0.00
Avg. Occupancy (%)	0.32 ± 0.00	90.2 ± 0.04	**90.2 ± 0.00**
Avg. Runtime	16403 ± 1831	**87.3 ± 0.00**	552.19 ± 27.69

9 Conclusion

BicACO was the first biclustering technique to utilize ant colony optimization. Coelho et al.'s technique outperformed CC and FLOC but exhibited sluggish runtimes. To compensate for this drawback, this thesis introduced the SNAP algorithm, a faster and higher performing alternative. Within the analysis of Yeast, SNAP ran 22 times faster than bicACO while returning superior results. Furthermore, SNAP exhibited greater versatility as it was able to process MovieLens, a highly sparse collaborative filtering dataset.

The results of this thesis have multiple implications to biclustering research. Firstly, they have demonstrated that ant optimization can be utilized to solve the biclustering problem without incurring a heavy time cost. Subsequent research efforts should focus on the development of newer ant-optimized techniques by employing principles of different generic ACO algorithms (besides the Hypercube Framework). Secondly, SNAP's biclusters (using PS9) should be subject to biological analysis due to their statistical superiority and high overlap with bicACO's results. SNAP should be used for analysis of different microarray datasets. Lastly, SNAP's ability to handle sparse matrices is an indication of great potential for CF applications. Future studies should integrate this algorithm into recommender systems, evaluating their effectiveness in e-commerce and online dating sites.

Works Cited

[1] Han, J. and Kamber, M., *Data Mining: Concepts and Techniques.* San Francisco: Information Retrieval, 2006.

[2] Berkhin, P., "Survey Of Clustering Data Mining Techniques." [Online] 2002. http://www.ee.ucr.edu/~barth/EE242/clustering_survey.pdf.

[3] Xu, R. and Wunsch II, D., "Survey of Clustering Algorithms." *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645-678, 2005.

[4] Piatetsky-Shapiro, G. and Tamayo, P., "Microarray Data Mining: Facing the Challenges." *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 2, pp. 2-5. 2003.

[5] Gaasterland, T and Bekiranov, S., "Making the Most of Microarray Data." *Nature Genetics*, vol. 24, pp. 204–206, 2000.

[6] Cheng, Y and Church, George M., "Biclustering of expression data." *International Conference on Intelligent Systems for Molecular Biology*, 2000.

[7] Coelho, G.P., Franca, F.O. and Von Zuben, F.J., "A Multi-Objective Multipopulation Approach for Biclustering." *Proceedings of the 7th International Conference on Artificial Immune Systems*, 2008.

[8] Coelho, G.P., França, F.O. and Von Zuben, F.J., "Multi-Objective Biclustering: When Non-dominated Solutions are not Enough." *Journal of Mathematical Modeling and Algorithms*, vol. 8, no. 2, pp. 175-202, 2009.

[9] Yang, J., Wang, H., Wang, W. and Yu, P., "Enhanced Biclustering of Expression Data." *Proceedings of the 3rd IEEE Symposium on BioInformatics and BioEngineering*, 2003.

[10] Coelho, G.P., Franca, F.O. and Von Zuben, F.J., "bicACO: An Ant Colony Inspired Biclustering Algorithm." *Proceedings of the 6th international conference on Ant Colony Optimization and Swarm Intelligence*, 2008.

[11] Franca, F.O., "Private communications."

[12] Cho, R.J., Campbell, M.J., Winzeler, E.A., Steinmetz, L., Conway, A., Wodicka, L., Wolfsberg, T.G., Gabrielian, A.E., Landsman, D., Lockhart, D.J. and Davis, R.W., "A Genome-Wide Transcriptional Analysis of the Mitotic Cell Cycle." *Cell*, Vol. 2, pp. 65-73, 1998.

[13] Castro, P.A.D., Franca, F.O., Ferreira, H.M. and Von Zuben, F.J., "Evaluating the Performance of a Biclustering Algorithm Applied to Collaborative Filtering-A Comparative Analysis." *7th International Conference on Hybrid Intelligent Systems*, 2007.

[14] Herlocker, J. L., Konstan, J. A., Borchers, A. and Ridel, J., "An Algorithmic Framework for Performing Collaborative Filtering." *Proceedings of the 22nd Annual international ACM SIGIR Conference on Research and Development in information Retrieval*, 1999.

[15] Madeira, S.C. and Oliveira, A.L., "Biclustering Algorithms for Biological Data Analysis: A Survey." *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 1, no. 1, pp. 24-45, 2004.

[16] Hartigan, J. A., "Direct Clustering of Data Matrix." *Journal of the American Statistical Association*, vol. 67, no. 337, pp. 123-129, 1972.

[17] Getz, G., Levine, E. and Domany, E., "Coupled Two-Way Clustering Analysis of Gene Microarray Data." *PNAS*, Vol. 97, pp. 12079-12084, 2000.

[18] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New Jersey: W.H. Freeman and Company, 1979.

[19] Golub, T.R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D. and Lander, E. S., "Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring." *Science*, vol. 286, no. 5439, pp. 531-537, 1999.

[20] Brown, T.A. *Genomes 3.* New York: Garland Science Publishing, 2007.

[21] Spellman, P. T., Sherlock, G., Zhang, M. Q., Iyer, V. R., Anders, K., Eisen, M. B., Brown, P. O., Botstein, D. and Futcher, B., "Comprehensive Identification of Cell Cycle-Regulated Genes of the Yeast Saccharomyces Cerevisiae by Microarray Hybridization." *Molecular Biology of the Cell*, vol. 9, pp. 3273–3297, 1998.

[22] Eisen, M. B., Spellman, P. T., Brown, P. O. and Botstein, D., "Cluster Analysis and Display of Genome-Wide Expression Patterns." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 95, pp. 14863–14868, 1998.

[23] Bittner, M., Meltzer, P. and Trent, J., "Data Analysis and Integration: of Steps and Arrows." *Nature Genetics*, vol. 22, no. 3, pp. 213-215, 1999.

[24] Ihmels, J., Friedlander, G., Bergmann, S., Sarig, O., Ziv, Y. and Barkai, N., "Revealing Modular Organization in the Yeast Transcriptional Network." *Nature Genetics*, vol. 31, pp. 370-377, 2002.

[25] Murali, T. M. and Kasif, S., "Extracting Conserved Gene Expression Motifs from Gene Expression Data." *Pacific Symposium on Biocomputing*, 2003.

[26] Ben-Dor, A., Chor, B., Karp, R. and Yakhini, Z., "Discovering Local Structure in Gene Expression Data: the Order-Preserving Submatrix Problem." *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, vol. 10, pp. 373-384, 2003.

[27] Goldberg, D., Nichols, D., Oki, B. M. and Terry, D., "Using collaborative filtering to weave an information Tapestry." *Communications of the ACM*, vol. 35, 1992.

[28] Symeonidis, P., Nanopoulos, A., Papadopoulos, A. and Manolopoulos, Y., "Nearest-Biclusters Collaborative Filtering with Constant Values." *Information Retrieval*, vol. 11, no. 1, pp. 51-75, 2007.

[29] Xue, G. R., Lin, C., Yang, Q., Xi, W., Zeng, H. J., Yu, Y. and Chen, Z., "Scalable Collaborative Filtering Using Cluster-based Smoothing." *In Proceedings of the ACM SIGIR Conference*, 2005.

[30] Goldberg, K., Roeder, T., Gupta, D. and Perkins, C., "Eigentaste: A Constant Time Collaborative Filtering Algorithm." *Information Retrieval*, vol. 4, no. 2, pp. 133-151, 2001.

[31] Yu, K., Schwaighofer, A., Tresp, V., Xu, X. and Kriegel, H. P., "Probabilistic Memory-Based Collaborative Filtering." *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 1, pp. 56-69, 2004.

[32] Brozovsky, L and Petrıcek, V., "Recommender System for Online Dating Service." *Proceedings of Conference Znalosti*, 2007.

[33] Miyahara, K and Pazzani, M. J., "Collaborative Filtering with the Simple Bayesian Classifier." *Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*, 2000.

[34] Kim, M. W., Kim, E. J. and Ryu, J. W., "Collaborative Filtering for Recommendation Using Neural Networks." *Lecture Notes in Computer Science*, vol. 3483, pp. 127-136, 2005.

[35] Tang, C., Zhang, L., Zhang, A. and Ramanathan, M., "Interrelated Two-Way Clustering: An Unsupervised Approach for Gene Expression Data Analysis." *Proceedings 2nd Annual IEEE International Symposium on Bioinformatics and Bioengineering*, 2001.

[36] Aguilar-Ruiz, J. S., "Shifting and Scaling Patterns from Gene Expression Data." *Bioinformatics*, vol. 20, pp. 3840–3845, 2005.

[37] Maniezzo, V., Gambardella, L. M. and de Luigi, F., "Ant Colony Optimization." [book auth.] Onwubolu, G. C. and Babu, B.V. *New Optimization Techniques in Engineering.* New York: Springer, pp. 101-120, 2004.

[38] Maniezzo, V., "Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem." *INFORMS Journal of Computing*, vol. 11, no. 4, 1995.

[39] Blum, C. and Dorigo, M., "The Hyper-Cube Framework for Ant Colony Optimization." *IEEE Transactions on Systems, Man, and Cybernetics— Part B*, vol. 34, no. 2, pp. 1161-1172, 2004.

[40] Brooks, A. J., "The Essence of SNPs." *Gene*, vol. 234, no. 2, pp. 177-186, 1999.

[41] Mitra, A. K., Singh, N., Garg, V. K., Chaturvedi, R., Sharma, M. and Rath, S. K., "Statistically Significant Association of the Single Nucleotide Polymorphism (SNP) rs13181 (ERCC2) with Predisposition to Squamous Cell Carcinomas of the Head and Neck (SCCHN) and Breast Cancer in the North Indian Population." *Journal of Experimental & Clinical Cancer Research*, vol. 28, no. 1, pp. 104, 2009.