

RaPiDS: An Algorithm for Rapid Expression Profile Database Search

Paul B. Horton Larisa Kiseleva
horton-p@aist.go.jp kiseleva-larisa@aist.go.jp

Wataru Fujibuchi
fujibuchi-wataru@aist.go.jp

Computational Biology Research Center, National Institute of Advanced Industrial Science and Technology, 2-42 Aomi, Koto-ku, Tokyo 135-0064, Japan

Abstract

In this paper we present a fast algorithm and implementation for computing the Spearman rank correlation (SRC) between a query expression profile and each expression profile in a database of profiles. The algorithm is linear in the size of the profile database with a very small constant factor. It is designed to efficiently handle multiple profile platforms and missing values. We show that our specialized algorithm and C++ implementation can achieve an approximately 100-fold speed-up over a reasonable baseline implementation using Perl hash tables.

RaPiDS is designed for general similarity search rather than classification – but in order to attempt to classify the usefulness of SRC as a similarity measure we investigate the usefulness of this program as a classifier for classifying normal human cell types based on gene expression. Specifically we use the k nearest neighbor classifier with a t statistic derived from SRC as the similarity measure for profile pairs. We estimate the accuracy using a jackknife test on the microarray data with manually checked cell type annotation. Preliminary results suggest the measure is useful (64% accuracy on 1,685 profiles *vs.* the majority class classifier's 17.5%) for profiles measured under similar conditions (same laboratory and chip platform); but requires improvement when comparing profiles from different experimental series.

Keywords: gene expression profile, database search, Spearman's rank correlation coefficient

1 Introduction

The rapidly expanding collection of microarray data publicly available from sites, such as NCBI's GEO [2], provides an immense opportunity for understanding the results of microarray experiments in the context of a large corpus of expression profiles (we use the term “profile” to mean many genes measured in one experiment).

The potential utility of database mining such large collections of expression data was understood from the early days of microarray technology [3]. Indeed Hunter *et al.* [7] suggested a relatively sophisticated similarity measure for comparing expression profiles, and considered database search as an application. Their paper provides an excellent discussion of the issues involved but, on tests with the limited data available at the time, their proposed measure did not outperform traditional measures of correlation – and computation time appears to be a serious issue when scaling their measure up to large datasets.

Much work has been done on both supervised classification (for example [4] and clustering (reviewed in [5] of expression profiles. But although somewhat related, work in these areas are not designed for fast, general search based only on similarity of expression.

The CellMontage server [6, 8] provides search for profiles similar to a query profile, from over 50,000+ stored expression profiles. This requires the definition of a suitable similarity function for

measuring the similarity of two profiles. The task is complicated by the fact that there are many different types of microarray platforms measuring distinct gene sets and employing various normalization techniques. Moreover, since the number of available profiles is growing rapidly, and one profile often contains many more than 10,000 genes, the similarity measure must allow for efficient computation. The Spearman rank correlation (SRC or ρ) is a simple, well studied similarity measure which avoids the difficulties of cross-platform numerical comparison. Here we present a practical and efficient algorithm (including constant factors) for rapid computation of SRC, which we have implemented as the RaPiDS (Rapid Profile Database Search) program. In addition, we use 1,685 expression profiles with manually curated cell type annotation to evaluate the effectiveness of SRC as a similarity measure for the task of classifying cell types from expression data.

2 Computational Problem

2.1 Definitions

We use the term *gene* to refer to an entity whose expression value is measured. For an Affymetrix type chip this will generally be a probe set. However the program presented here can be used on the individual probe level for comparison within a particular platform. We use the term *platform* to refer to a set of genes whose values can be measured in one experiment with a particular device. In the data used for this study the platforms are microarray chips of a particular model. We refer to the result of an experiment using a particular platform as a *profile* from that platform. A profile is a set of (gene, expression value) pairs containing some (in the case of missing values) or all of the genes its platform can measure.

2.2 Spearman Rank Correlation

The Pearson correlation coefficient for equal length vectors of paired numbers $X = x_1, \dots, x_n$, $Y = y_1, \dots, y_n$; x_i paired with y_i is given by:

$$\rho = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}.$$

SRC is defined as the Pearson correlation coefficient computed on the *ranks* of the values instead of the values themselves. The calculation of SRC (ρ) between two vectors X and Y can be simplified to:

$$\rho = 1 - \frac{6 - \sum_{i=1}^n (\mathbf{R}(x_i) - \mathbf{R}(y_i))^2}{n^3 - n} \quad (1)$$

where, $\mathbf{R}(x_i)$ represents the rank of value x_i . When ties (equal expression values within the same vector) are present, this formula is only an approximation to the actual Spearman rank correlation coefficient – but in any case we broke ties arbitrarily in this study. The t statistic for SRC is given by:

$$t = \rho \sqrt{\frac{n-2}{1-\rho^2}}. \quad (2)$$

For comparisons between randomly ordered profiles, t is distributed as a student's t distribution with $n-2$ degrees of freedom. For $n \gg 30$ the distribution of t approaches a standard normal distribution. In this case, t has a monotonic relationship to the statistical significance in which the null model is the comparison of two randomly ordered profiles. The sufficient statistics for computing t are the size of the set of common genes and the sum of squared ranked differences within that set.

Table 1: Outline of profile similarity calculation.

1. Calculate the intersection of the two profiles.
2. For both profiles, calculate the relative rank (by expression value) of each gene in the intersection.
3. Calculate the SRC of those ranks.
4. Use the SRC derived t statistic as a measure of the similarity of the two profiles.

Table 2: A toy example of the computation used for approximate profile matching is shown. Single letters represent gene names in order of expression value. “# c ” denotes the number of genes in common between the query and each profile, and “ Σd^2 ” denotes the sum of squares of the relative rank differences.

query	B G F C A E D	# c	Σd^2
DB profile 1	B G F C A E D	7	0
DB profile 2	G B C F A D	6	4
DB profile 3	G E B F	4	10
DB profile 4	E A G C	4	18
DB profile 5	D C G B F A	6	44

2.3 Matching Computation

Conceptually our approach to comparing two heterogeneous profiles is equivalent to the procedure shown in Table 1. Due to multiple platforms and the presence of missing values, two profiles generally cover a different set of genes. Table 2 shows the value of the sufficient statistics for SRC for an example query profile and profile dataset.

3 Performance Requirements

3.1 Problem Size

Our intended application, as the similarity matching engine for microarray database search, has strict performance requirements. As of May 2006, NCBI’s GEO repository contains over 80,000 microarray samples [9], most of which have been indexed for searching with RaPiDS from the CellMontage web site [6, 8]. Thus in the near future a matching engine will need to search through $p = 10^6$ or more expression profiles. For human or mouse microarray data, most expression profiles contain more than 10,000 genes. With the advent of arrays with larger numbers of probes, designed to detect alternative splicing or non-protein-coding transcripts, many profiles may soon contain 100,000 or more “genes”.

3.2 Query Time Requirement

Let $g = 10^{4\sim 5}$ be the average number of genes in a profile. A matching algorithm must be able to match a query containing roughly g genes against a dataset of $p \geq 10^6$ profiles of equal magnitude size. Since our intended application is an interactive search engine, the search time should be no more than 100 seconds (ideally much less). Assuming constant factors greater than one, these numbers not only rule out any algorithm which is slower than $O(gp)$, but since $gp \approx 10^{11}$, and modern computer clock speeds give around 10^9 operations per second, these numbers also require that the constant factor of any algorithm which is linear in gp , to be not much greater than one.

3.3 Query Preprocessing

Note that the size of the query alone is on the order of 10^6 at most. Thus preprocessing, such as sorting, of the query in time $O(g \log g)$ is feasible.

3.4 Database Preprocessing Time

Assuming that the database is prepared in a batch manner, perhaps every week or so, the amount of time which can be used for database preprocessing is on the order of $10^{5\sim 6}$ seconds. This extra time allows for $O(pg \log g)$ algorithms, and somewhat looser constraints on constant factors.

4 Profile Comparison Algorithms

4.1 Naïve Algorithm

A natural implementation would implement each step of the procedure shown in Table 1 independently at query time. The intersection of the genes would be computed with an associative data structure, such as a hash table. The expression ranks (relative to the intersecting set) would be computed by sorting, and the mapping of gene names to rank would be stored in an associative data structure. The SRC calculation would then be performed by iterating through the intersecting genes using the associative data structure. This naïve algorithm is close to linear in gp . The two computations potentially requiring super-linear time are the combined time of associative table lookups and sorting by expression. However good hash table implementations offer close to constant time lookup performance in practice. Also, since the distribution of expression values possible for each platform can be precomputed, a bucket sort could be used for linear time sorting. Unfortunately, although for many applications the pseudo-constant factors associated with these calculations would also be acceptable, they are far higher than that associated with a simple array lookup, which can be implemented in a few machine instruction codes.

4.2 RaPiDS

4.2.1 Serial Numbering of Genes

One key idea of RaPiDS is the use of serial numbers, which allow ordinary arrays to replace the associative data structures used by the naïve algorithm. For each platform, a canonical ordering (we use alphabetical ordering) of the genes which can be measured by that platform is defined. Dataset genes are converted to serial numbers before preparing the precomputed arrays described below. An associative data structure (our current implementation uses a sorted list) is used to convert the query gene names to serial numbers at query time.

4.2.2 Preprocessing of the Dataset Profiles

For each platform, the dataset profiles belonging to that platform are sorted by expression value and stored in an array of arrays (denoted `profiles` in the pseudo-code given below. For profile i , `profiles[i][j]` holds the serial number of the gene with rank j in the profile (generally different than the rank relative to the *intersection* with the query, which is what is ultimately needed). This is the major storage requirement for data indexed for use with RaPiDS, one (currently 64 bit) integer for each gene in each profile. This representation is compact and convenient. When expanding the database, new profiles can be added to the end of the binary file holding `profiles` without recomputing the existing profiles.

4.2.3 Computation of the Intersection between Query and Platform

The intersection between the query and platform is calculated as shown in the pseudo-code of Table 4. This calculation uses a conventional associative table and assumes the query has been sorted in order of expression value. However, this computation is not time critical because it is only called once per platform. This computation produces two arrays: `aCommonIdsQPt`, a boolean array for looking up if

a gene id is present in the query, and `eCommonIdsQPt`, an array for processing gene ids (contained in the current platform) in their order of query expression value.

4.2.4 Computation of the Relative Rank of Dataset Profile

`aCommonIdsQPt` and the fact that `profiles` is sorted by expression order is used to efficiently compute the rank relative to the intersection with the query profile of the dataset profile genes, as shown in “Loop D” in Table 5. This loop computes an array `relativeRankD`, such that `relativeRankD[k]` holds the intersection rank of dataset profile gene with id k , or 0 if k is not in the intersection.

4.2.5 Computation of the Relative Rank of Query Profile

Similar to “Loop D” for the dataset profile, “Loop Q” in Table 5, uses `relativeRankD` and the fact that `eCommonIdsQPt` is sorted by expression value to efficiently compute the intersection rank of the query profiles genes. Note that if the platform were known to never produce profiles with missing values, it would not be necessary to check for `relativeRankD` equaling 0. The sum of the squared rank differences, `d2`, in Table 5, is also computed in “Loop Q”. Here we compare and contrast RaPiDS versus the naïve algorithm:

Task	Naïve Algorithm	RaPiDS
calculate the intersection of the two profiles.	Use associative data structure. <i>time</i> : pseudo-linear with moderate constant factor	Sort query by canonical order once per platform, otherwise only simple array ops. <i>time</i> : linear with small constant factor ($\# \text{ profiles} \gg \# \text{ platforms}$)
for both profiles, calculate the relative rank of each gene in the intersection.	Sort the intersecting genes by expression value, store name to rank mapping in hash table. <i>time</i> : $O(g \log g)$ or pseudo-linear at best (bucket sort & hash table) moderate to high constant factor	Iterate through two g sized loops using only simple array ops. <i>time</i> : linear with small constant factor
calculate the sufficient statistics for the SRC of those ranks.	Compute by looking up gene ranks with hash table. <i>time</i> : pseudo-linear with moderate constant factor	Computation is integrated with the previous task. <i>time</i> : linear with small constant factor

The computation of the SRC of the common genes of two profiles by a Naïve algorithm and RaPiDS is compared step by step. The time row indicates the computation performed at query time and does not include preprocessing time.

5 RaPiDS Speed Result

Before designing the RaPiDS algorithm, we implemented a few Perl programs as prototypes. The fastest amongst them implements a slightly more sophisticated algorithm than the naïve algorithm described here – in that it avoids sorting at query time with preprocessing; but still uses Perl hash tables instead of the specialized algorithm RaPiDS uses to compute the gene set intersections. Table 3 shows a comparison of the speed of RaPiDS versus our Perl prototype and also the effect of replacing the serial number combined with simple array lookups of RaPiDS with a standard implementation of a general associative data structure in C++ (the C++ standard template library `map` class). Note that although the RaPiDS running time is simply linear in the number of database profiles, it displays a more complicated dependence on the query size.

Table 3: Times for different implementations of SRC based profile matching programs are shown. `std::map` uses the C++ standard template library map associative data structure. Perl implements the naïve algorithm described in the text. Times are given in seconds. The database searched contained 3,983 profiles with an average of 14,657 genes per profile. This experiment was performed on a 2.8 GHz Pentium 4 machine with 1GB of memory.

query size	RaPiDS	std::map	Perl	query size	RaPiDS	std::map	Perl
2	2.0	2.4	37	1,024	2.3	9.0	55
8	2.1	3.1	37	2,048	2.5	12	74
16	2.1	3.7	37	4,096	2.9	19	137
128	2.1	5.0	39	8,192	3.4	46	192
512	2.2	7.1	47	14,725	3.7	115	348

6 Classification Problem

We present a preliminary evaluation of the effectiveness of the t statistic of SRC as a similarity measure.

6.1 Dataset

We prepared a dataset of 1,685 GEO derived profiles with cell type information by manually checking the cited literature. All profiles were taken from normal, human cells and contained an average of 10,306 genes per profile. Each profile was labeled as one of 78 cell types and one of 29 cell “super types”. The most common types are shown in Table 6. Note that in some cases, such as “kidney” the assigned super type is the same as the cell type. We based this manual classification on traditional cytology.

6.2 Classification Accuracy Estimation

We measured the accuracy of the k nearest neighbor classifier with the SRC t statistic for the profile similarity measure, using a jackknife test on the dataset. For scoring, we gave the classifier one point for matching the cell type perfectly, otherwise half a point for getting the super type correct, and no points if the super type was misclassified. This produced an accuracy score of 63.9% with $k = 1$. In this experiment the average SRC and the number of common genes between the query and its nearest neighbor was 0.756 and 6,854 respectively. For comparison the majority class classifier would pick “leukocyte” in each case for an accuracy of just 17.5%.

6.2.1 Matching Across Experiments:

Due to platform dependent biases, it is expected that inter-platform comparison should be harder than intra-platform comparison. Also it has been observed that microarray results are often sensitive to the laboratory in which microarray experiments are performed. To investigate the magnitude of these effects, we performed another experiment in which the nearest neighbors were required to be from a different experimental series (GDS in GEO terminology) than the query profile. This experiment was performed on the 1,562 profiles whose cell types occurred in two or more GDS’s. An accuracy of 25.9% with $k = 5$ was attained. In this case the average SRC and the number of common genes between the query and its nearest neighbors was 0.407 and 3,674 respectively. After removing frequency information from the same GDS as the query, the majority class classifier yields an accuracy of 11.7%.

6.3 Comparison to Other Profile Search Programs

Although GEST [7] uses a more sophisticated similarity measure for matching profiles and is not optimized for speed, it would be desirable to compare the speed and accuracy of the two methods. The authors of GEST were kind enough to provide us with their source code – but unfortunately the code was written to read profiles in a particular format from a commercial database application. Modification of the GEST code to use it outside that environment should be possible but has not yet been done.

7 Discussion

7.1 RaPiDS Algorithm

We have presented RaPiDS, an efficient algorithm and implementation for calculating the SRC of a query expression profile to each expression profile in a database. We have shown RaPiDS is fast enough to be used on a reasonable sized database. One potential drawback of RaPiDS is that it scans through all of the dataset and therefore has no hope of being sub-linear in the database size. Clearly for queries involving a small number of genes an associative data structure could be used to compute SRC in sub-linear time relative to the dataset size. A more serious challenge would be to construct a sub-linear time search algorithm analogous to BLAST [1] for sequence similarity search. This may prove to be a daunting task, as BLAST and related algorithms exploit the fact that similar sequences contain small *exact* matches, whereas a gene in two similar expression profiles cannot be expected to have an identical expression value or even rank. On a more positive note, RaPiDS can handle the currently available data on a single commodity PC. A trivial parallelization done by dividing the dataset amongst several machines, each running RaPiDS on their data, and then merging the top hits, should allow RaPiDS to scale up by a factor of 10~100 relatively easily.

7.2 SRC as a Profile Comparison Similarity Measure

We tested the utility of using of SRC as a similarity measure for cell classification. The preliminary results reported here indicate that SRC has good predictive power for profiles obtained with the same platform under similar experimental conditions. The results obtained for comparison across experiment series were less impressive, but the prediction by SRC was still much more effective than the majority class classifier. We view this part of this study as a baseline for developing more effective similarity measures. Measures which may use a modified version of RaPiDS if they are computationally related to SRC, or may use RaPiDS as an initial filter, if they prove too computationally intensive to compute against the entire dataset.

7.3 Conclusion

We have described in detail the algorithm used by the first practical program for rapidly searching large expression profile databases for similar expression profiles. Moreover we have shown it to be somewhat effective as measured by a reasonable classification task on real data.

Table 4: Pseudocode showing the computation of genes common to query and platform. The notation is based on C++ or Java. // marks comments, == tests equality, = denotes assignment, ++i represents $i = i + 1$, string[] declares an array of strings, eGeneNamesQ.size() is the number of elements in array eGeneNamesQ.

```
// computeQueryPlatformIntersection
//
// computes: eCommonIdsQPt, aCommonIdsQPt.
// This function is only called once per platform.
// The id of a gene is its serial number for the current platform
// Array names starting in "e" involve ordering by expression
// Array names starting in "a" involve alphabetical ordering
//
// input: eGeneNamesQ, stringToGeneSerialNo
//
// eGeneNamesQ: holds the query gene names in expression order
//
// stringToGeneSerialNo( geneName ): function returning the
// serial number of the string geneName -- or NOT_FOUND if
// the gene named geneName is not measured by the current
// platform. Currently implemented with a sorted list.
//
// output: eCommonIdsQPt, aCommonIdsQPt
//
// eCommonIdsQPt: an array holding the gene ids common to query
// and platform in the expression order of the query.
// Its size equals the number of common ids.
// aCommonIdsQPt: a boolean array holding TRUE for indexes
// representing gene ids common to query and platform.
// Its size equals the number of platform ids.
//
computeQueryPlatformIntersection( string[] eGeneNamesQ,
                                stringToIntTable stringToGeneSerialNo ){
    eCommonIdsQPt.clear(); // initialize to empty list
    aCommonIdsQPt.zero(); // initialize all elements to FALSE.

    // process in order of expression value.
    for( int i = 0; i < eGeneNamesQ.size(); ++i ){
        int curId = stringToGeneSerialNo( eGeneNamesQ[i] );
        if( curId == NOT_FOUND ) continue;
        push eCommonIdsQPt, curId; // push onto end of eCommonIdsQPt.
        aCommonIdsQPt[ curId ] = TRUE;
    }
}
```

Table 5: Pseudocode showing the computation of rank correlation for the query compared with each profile in the database. The notation is based on C or Java. (Table 4 for more details).

```

// computeRankCorrelationsAux computes: match results
// array names including "Q" hold query profile information.
// array names including "D" hold dataset profile information.
// also see the pseudo-code of computeQueryPlatformIntersection
//
// Input: profiles, aCommonIdsQPt
// profiles: is an array of profiles. Each profile is an array of
//   gene platform serial numbers representing a profile in the
//   dataset. The serial numbers in the profile appear in order
//   of their expression values. Note that a profile does not
//   necessarily contain all of the genes in the platform.
// aCommonIdsQPt: see computeQueryPlatformIntersection.
//
// Output: matchResults:
//   an array of resultStruct. Each element holds
//   (i, numCommonGenes, d2) where;
//   i: a number identifying the dataset profile
//   numCommonGenes: the number of common genes
//   between the query and that profile.
//   d2: the sum of squared rank differences.
//
computeRankCorrelationsAux( int[][] profiles,
                           boolean[] aCommonIdsQPt ){
  resultStruct[] matchResults.clear(); // make empty list.
  int[] idToRelativeRankD;

  for( int i = 0; i < profiles.size(); ++i ){
    // for each profile...
    // initialize all values of idToRelativeRankD to zero.
    idToRelativeRankD.zero();
    int relativeRankD = 1; // ranks count from 1.
    // ** Loop D in main text **
    for( unsigned int j = 0; j < profiles[i].size(); ++j ){
      if( aCommonIdsQP[ profiles[i][j] ] == TRUE ){
        idToRelativeRankD[ profiles[i][j] ] = relativeRankD;
        ++relativeRankD;
      }
    } // at end of for loop: for dataset profile gene, id k:
    // relativeRankD[k] == 0 --> gene not in query.
    // == r, r > 0, --> gene has rank r in intersection.
    double d2 = 0.0; // to hold sum of squared rank differences
    unsigned int numCommonQueryAndProfile = 0;
    unsigned int relativeRankQ = 0;
    // ** Loop Q in main text **
    for( unsigned int j = 0; j < eCommonIdsQPt.size(); ++j ){
      relativeRankD = idToRelativeRankD[ eCommonIdsQPt[j] ];
      if( relativeRankD > 0 ){
        // relativeRankD == 0 --> gene eCommonIdsQPt[j]
        //   not found in dataset profile[i]
        ++numCommonQueryAndProfile;
        ++relativeRankQ; // equals one on first use.
        int diff = relativeRankQ - relativeRankD;
        d2 = d2 + (diff * diff);
      }
    }
    // add the result for the ith profile.
    push matchResults, (i, numCommonQueryAndProfile, d2);
  } // goto next dataset profile.
  return matchResults;
}

```

Table 6: The most frequent cell types and super types in the dataset are shown. In the third column “same” indicates that the super type and cell type are the same for that cell type. Abbreviations: (“reproductive”, “reprod.”), (“immune system related”, “imm. sys. rel”).

freq	type	super type	freq	type	super type
296	leukocyte	immune system	22	bladder	same
249	skeletal muscle	muscle	21	retina	same
162	kidney epithelial	kidney	20	spinal cord	nerve
118	liver	same	19	breast	same
51	lung	respiratory	18	thymus	imm. sys. rel.
47	kidney	same	16	thyroid	same
37	cardiac muscle	muscle	16	stomach	digestive tract
33	frontal cortex	brain	15	uterus	female reprod.
32	fibroblast(lung)	fibroblast	15	spleen	same
31	fibroblast(foreskin)	fibroblast	14	ovary	female reprod.
30	bronchi epithelial	same	14	breast epithelial cell	same
27	hypothalamus	brain	14	bone marrow	imm. sys. rel.
24	astrocyte	nerve	13	smooth muscle	muscle
23	fibroblast	same	11	cervix	female reprod.
22	prostate	male reprod.	10	salivary gland	same
22	pancreas	same	10	cerebellum	brain
22	brain	same	10	adrenal gland	same

References

- [1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J., Basic local alignment search tool, *J. Mol. Biol.*, 215(3):403–410, 1990.
- [2] Barrett, T., Suzek, T.O., Troup, D.B., Wilhite, S.E., Ngau, W.C., Ledoux, P., Rudnev, D., Lash, A.E., Fujibuchi, W., and Edgar, R., NCBI GEO: Mining millions of expression profiles-database and tools, *Nucleic Acids Res.*, 33(Database issue):D562–D566, 2005.
- [3] Basset, D.E. Jr., Eisen, M.B., and Boguski, M.S., Gene expression informatics – it’s all in your mine, *Nat. Genet.*, 21(Suppl. 1):51–55, 1999.
- [4] Davis, C.A., Gerick, F., Hintermair, V., Friedel, C.C., Fundel, K., Küffner, R., and Zimmer, R., Reliable gene signatures for microarray classification: Assessment of stability and performance, *Bioinformatics*, 22(19):2356–2363, 2006.
- [5] D’haeseleer, P., How does gene expression clustering work?, *Nat. Biotechnol.*, 23(12):1499–1501, 2005.
- [6] Fujibuchi, W., Kiseleva, L., Taniguchi, T., Harada, H., and Horton, P., Cellmontage: Searching a large microarray database for similar expression profiles, *in preparation*.
- [7] Hunter, L., Taylor, R.C., Leach, S.M., and Simon, R., GEST: A gene expression search tool based on a novel Bayesian similarity metric, *Bioinformatics*, 17(Suppl. 1):S115–S122, 2001.
- [8] <http://cellmontage.cbrc.jp/>
- [9] <http://www.ncbi.nlm.nih.gov/projects/geo/>