particleMDI: a Julia Package for the Integrative Cluster Analysis of Multiple Datasets

Nathan Cunningham, Jim E. Griffin, David L. Wild, and Anthony Lee [*]

Abstract We present particleMDI, a Julia package for performing integrative cluster analysis on multiple heterogeneous data sets, built within the framework of multiple data integration (MDI). particleMDI updates cluster allocations using a particle Gibbs approach which offers better mixing of the MCMC chain—but at greater computational cost—than the original MDI algorithm. We outline approaches for improving computational performance, finding the potential for greater than an order-of-magnitude improvement. We demonstrate the capability of particleMDI to uncovering the ground truth in simulated and real datasets. All files are available at `https://github.com/nathancunn/particleMDI.jl`

Key words: Bayesian inference, Cluster analysis, Computational statistics, Data integration, Particle Monte Carlo methods

Nathan Cunningham
Department of Statistics, University of Warwick, Coventry, CV4 7AL
e-mail: `n.cunningham@warwick.ac.uk`

Jim E. Griffin
University College London

David L. Wild
University of Warwick

Anthony Lee
University of Bristol

[*] Nathan Cunningham's research is funded by EPSRC grant 1654596 and supported by The Alan Turing Institute under grant TU/D/000013. David Wild acknowledges support from EPSRC grant EP/R014337/1.

1 Introduction

Cluster analysis is the task of inferring a latent group structure in data, such that observations within groups are, in some sense, 'closer' to one another than to observations in other groups. Standard methods, such as k-means, are not equipped for cases where the units of observation have data arising from multiple sources. Integrating multiple data sources into a composite analysis is a key challenge in the analysis of genomic data where multiple heterogeneous datasets can give different—but complementary—views of the same underlying process. In this context, one may perform cluster analysis to infer risk cohorts among groups of patients for whom we have multiple biological data sets recorded. We introduce particleMDI, a package developed in the statistical programming language Julia[2] for performing integrative cluster analysis in this context. While many such approaches exist[see e.g. 13, 17, 12, 7], particleMDI is built within the framework of multiple data integration (MDI)[11].

MDI facilitates integrative cluster analysis by allowing for the borrowing of information between datasets of potentially different types. Observations arise from a Dirichlet-multinomial allocation mixture model[9]—a finite approximation to the Dirichlet process mixture model. To infer dependence between the cluster structure of different datasets, Kirk et al. introduce a parameter, Φ, measuring the similarity between pairs of datasets at the level of the cluster allocations. The inferred value of, e.g., $\phi_{k,l}$ is used to inflate the probability of assigning observations in dataset k to the clusters they are assigned to in dataset l.

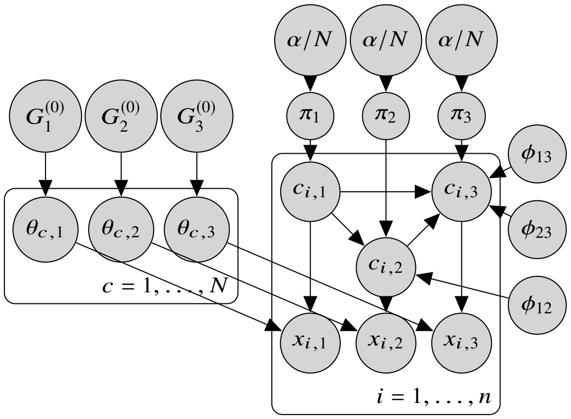

Fig. 1 A graphical model representation of MDI and particleMDI in a three dataset case ($K = 3$). $x_{i,k}$ denotes observation i in dataset k arising from cluster $c_{i,k}$ with parameters $\theta_{c_i,k}$, which are given a prior $G_k^{(0)}$. Cluster a in dataset k has prior allocation weight $\pi_{a,k}$ which is given a $Dirichlet(\alpha/N)$ prior. The $\phi_{i,j}$ values allow the allocations in data set i to inform those in data set j. (Figure recreated from [11])

Inference in MDI is performed via a Gibbs sampler, alternating between updating cluster allocations and hyperparameters; full details are available in [11]. As conjugate priors are used in MDI, the cluster parameters can be analytically integrated over and individual cluster allocations are updated while holding all other allocations fixed[14]. A result of this one-at-a-time approach is that once MDI infers an allocation which is ‘good enough’, in some sense, it can be difficult to consider alternatives unless they are similar to this current allocation.

2 particleMDI

particleMDI extends the original MDI algorithm, replacing the one-at-a-time approach to clustering with a conditional particle filter, which has demonstrated good mixing properties even when the number of particles is relatively low[8]. This approach to cluster analysis [see 4, 5, 8] infers a latent cluster allocation, $c_{i,k}$, for an observation, $x_{i,k}$, given observations $x_{1:i,k}$ and allocations $c_{1:(i-1),k}$, using a weighted cloud of approximations, termed particles. The particle approximation of the Gibbs sampler[1] uses a conditional sequential Monte Carlo (SMC) update, which uses a single particle, sampled appropriately from the particle filter, to update the hyperparameters. The trajectory of this ‘reference particle’ is held fixed through a subsequent pass of the conditional SMC update, thus guiding other particles towards relevant regions of the state space.

We use the parameter Φ to share information across datasets by inflating the weights of particles in which allocations agree across datasets, as detailed in Alg. 1.

2.1 Improving Computational Performance

particleMDI is much more computationally costly than the original MDI algorithm. Fearnhead[5] discusses the inherent inefficiencies in particle Monte Carlo algorithms as applied to clustering algorithms: resampling and the discrete nature of the state-space mean it is likely some particles will be duplicates of others. Calculation of mutation weights—the weights for assigning an observation to each cluster—involves evaluating the posterior predictive of assigning an observation to each cluster. This step is wholly deterministic meaning identical particles will have identical mutation weights and, thus, there is no value in evaluating them more than once. To tackle this, we identify duplicated particles via the following ID assignments

$$ID_{i+1}^{(m)} = ID_i^{(m)} \times (M \times N) + c_{i+1}^{(m)}$$

where particle m assigns observation $i + 1$ to cluster $c_{i+1}^{(m)}$, M is the total number of particles, and N the maximum number of clusters.

It is also likely that where particles differ, they may share commonality, be it a shared subset of clusters, or even a shared partition up to a permutation of cluster labels. Again, this will involve redundant calculations evaluating posterior predictives multiple times for the same clusters. We adapt our algorithm so that each particle indexes into a global environment of clusters containing only a single copy of each unique cluster. We now need only evaluate posterior predictives once for

Algorithm 1 particleMDI

```
Inputs:
    $\boldsymbol{\pi}$, $\boldsymbol{\Phi}$, cluster allocations $c^*_{1:n,1:K}$, a random permutation over observation
    indices, $\sigma(\cdot)$, and thresholds $\alpha$ and $\rho$ to control resampling and the portion
    of data conditioned-on, respectively
Initialize:
    Set particle weights $\xi^{(1)}, \dots, \xi^{(M)} = 1$
    Set $\sigma(c)^{(1)}_{1:n,1:K} = \sigma(c)^*_{1:n,1:k}, \sigma(c)^{(2:M)}_{1:\lfloor n\rho \rfloor,1:K} = \sigma(c)^*_{1:\lfloor n\rho \rfloor,1:K}$
for $i = \lceil n\rho \rceil, \dots, n$ do                    ▹ (iterate over remaining observations)
    for $m = 1, \dots, M$ do                                   ▹ (iterate over particles)
        for $k = 1, \dots, K$ do                               ▹ (iterate over datasets)
            if $m \neq 1$ then                                 ▹ (particle 1 is the reference)
                Sample $\sigma(c)^{(m)}_{i,k}$                 ▹ (assign observation $\sigma(x)_{i,k}$ to a cluster)
                $q(\sigma(c)^{(m)}_{i,k} = a) \propto f(\sigma(x)_{i,k} | \sigma(c)^{(m)}_{i,k} = a) \times \pi_{a,k}$
            end if
        end for
        $\xi^{(m)} = \xi^{(m)} \times \prod_{k=1}^{K} \sum_{a=1}^{N} \pi_{a,k} f(\sigma(x)_{i,k} | \sigma(c)^{(m)}_{i,k} = a) \times \prod_{k=1}^{K-1} \prod_{l=k+1}^{K} (1 + \phi_{k,l} \mathbb{1}(\sigma(c)^{(m)}_{i,k} =$
$\sigma(c)^{(m)}_{i,l}))$                 ▹ (Update particle weights accounting for agreement across datasets)
    end for
    Calculate effective sample size $(ESS) = \frac{(\sum_{m=1}^{M} \xi^{(m)})^2}{\sum_{m=1}^{M} \xi^{(m)2}}$.
    if ESS $< \alpha M$ then
        resample particles according to $\frac{\xi^{(m)}}{\sum_{m=1}^{M} \xi^{(m)}}$ and reset particle weights $\xi^{(1)}, \dots, \xi^{(M)} = 1$
    end if
end for
Select a final cluster allocation according to $\frac{\xi^{(m)}}{\sum_{m=1}^{M} \xi^{(m)}}$ and use to update $\boldsymbol{\pi}$, $\boldsymbol{\Phi}$ and use as $c^*_{1:n,1:K}$
and return to start.
```

each unique cluster and then combine these at the level of the particle to form the mutation weights.

A separate layer of inefficiency arises in the sequential nature of SMC methods. Evaluation of the posterior predictive of observation i conditional on the cluster allocations of observations $1 : (i - 1)$ is uninformative for very small values of i. To address this, we augment the particle Gibbs sampler such that we only update a predetermined number of cluster labels, holding $\lfloor n\rho \rfloor$ labels fixed for $0 < \rho < 1$. As the observations are exchangeable, we permute the observation indices according to a uniform permutation function σ and hold the first $\lfloor n\rho \rfloor$ cluster labels fixed from a previous pass of the conditional particle filter. The idea of updating blocks of sequential observations in the particle Gibbs sampler has previously been discussed[1] and a similar idea has been explored in the context of cluster analysis[3]. The permutation function, $\sigma(\cdot)$, is updated at every Gibbs iteration, ordering observation such that $\sigma(c)_{i,k}$ is the allocation for observation $\sigma(x)_{i,k}$ and $\sigma(x)_{i,l}$ corresponds to the same observational unit in a different data set. Therefore, where the standard particle Gibbs algorithm samples alternately from $p(\theta | x_{1:n}, c_{1:n})$ and $p_\theta(c_{1:n} | x_{1:n})$, our approach samples from $p(\theta | x_{1:n}, c_{1:n})$ and $p_\theta(\sigma(c)_{\lceil n\rho \rceil:n} | \sigma(x)_{1:n}, \sigma(c)_{1:\lfloor n\rho \rfloor})$. ($\theta$ here refers to the hyperparameters of the model, not the cluster parameters indicated in Fig. 1—as we use conjugate priors, the cluster parameters can be integrated out.) As well as giving the algorithm a 'warm start', this also avoids introducing a dependency between the inferred allocations and the order data are observed. Other approaches, such as that in [8], resolve this issue by instead updating all previous allocations during the resampling step. In a worst-case scenario—where resampling is performed at every step—this would increase the complexity of the algorithm from $O(n)$ to $O(n^2)$, assuming the mutation weights can be computed in constant time. The choice of ρ warrants careful consideration as it imposes a trade-off between computation time and the mixing of the algorithm. However, where computation time is not a concern, lower values of ρ are not strictly to be preferred; setting ρ too low can result in too few conditioned-on observations to overcome the dependency in the observation order. We explore the impact of this in Sec. 4.

3 Using the `particleMDI` package

The `pmdi()` function provides the primary functionality of the particleMDI algorithm. It takes the following inputs:

- `dataFiles` a vector of matrices containing the data to be clustered
- `dataTypes` a vector of types. For convenience, Gaussian and categorical data types are included and can be specified as `particleMDI.GaussianCluster` and `particleMDI.CategoricalCluster` respectively. However, this can easily be extended to any other data type for which a posterior predictive can be specified, as detailed in Sec. 3.1
- `N` the maximum number of clusters to be inferred in each dataset
- `particles` an integer indicating the number of particles to use in the analysis
- ρ a value in $(0,1)$ indicating the proportion of the data whose allocations are held fixed at each iteration of the Gibbs sampler, as outlined in Sec. 2.1.
- `iter` an integer specifying the number of iterations of the Gibbs sampler
- `outputFile` a string specifying the path of a .csv file in which to store the output

`pmdi()` outputs a .csv file, where each row contains the mass parameters, the phi values, and the allocations $c_{1:n,1:K}$. A user can assess this output file via some plotting functions built in the Julia library Gadfly[10]. In order to visualise the cluster allocations from multiple iterations of the Gibbs sampler, as well as across multiple datasets, `generate_psm()` and `consensus_map()`, can be used to visualise the posterior similarity matrices[13, 16] as heatmaps. That is, for each of K datasets, an $n \times n$ heatmap is constructed where element (i, j) reflects the frequency that observations i and j are assigned to the same cluster, as seen in Fig. 2. `plot_phimatrix()`, `plot_phichain()`, and `plot_nclust()` can each be useful tools for monitoring convergence of the Gibbs sampler, returning a heatmap of mean Φ values, a line graph of inferred Φ values, and the number of clusters inferred respectively.

3.1 Extending particleMDI for user-defined data types

One of the strengths of the original MDI method is its ability to cluster a variety of different data types within a single analysis. While we provide the functionality for Gaussian and categorical data types, we take advantage of Julia's multiple dispatch capabilities to allow users to extend `particleMDI` to perform cluster analysis on other data types. As Julia code is just-in-time compiled, these user-specified data types do not suffer any penalty in terms of computation time. We illustrate this capability with a trivial example of assigning observations to clusters based on their sign.

We first create a cluster struct, a structure containing a single cluster and sufficient statistics for calculating the posterior predictive. In this case, we just need indicators of whether any observations belong to the cluster, as well as their sign.

```
mutable struct SignCluster
  n::Int64    # No. of observations in cluster
  isneg::Bool # Positive/negative flag
  SignCluster(dataFile) = new(0, false)
end
```

We then define `calc_logprob`, a function which returns the log posterior predictive of an observation, `obs`, given the observations assigned to cluster `cl`. It is important to specify `cl` as being of type `SignCluster`.

```
function particleMDI.calc_logprob(cl::SignCluster, obs)
    if cl.n == 0
        return log(0.5)
    else
        return ((obs[1] <= 0) == cl.isneg) ? 0 : - 10
    end
end
```

Finally, the function `cluster_add!` updates a cluster, `cl`, when an observation, `obs`, is added to it.

```
function particleMDI.cluster_add!(cl::SignCluster, obs)
    cl.n += 1
    cl.isneg = (obs[1] < 0)
end
```

We can now cluster univariate data into positive and negative clusters by passing `SignCluster` as a data type in `pmdi()`.

4 Application

We demonstrate `particleMDI` on three simulated Gaussian datasets, with cluster means $\mu_{\cdot,1} = [-0.5, 0, 0.75]$, $\mu_{\cdot,2} = [0, 0.75, -0.5]$, $\mu_{\cdot,3} = [0.75, -0.5, 0]$, where $\mu_{i,j}$ indicates the mean of observations belonging to cluster i in dataset j. We choose balanced clusters for clarity of illustrating results; analysis on other data suggests this does not unduly impact the results. All observations are drawn independently with standard deviation $\sigma = 1$ meaning we expect significant overlap across all clusters. Each data set has 150 observations with 16 dimensions, with 25% being noise. The analyses are run for 1000 iterations, with $M = 32$, and $\rho = 0.25$. The results, in Fig. 2, show that, by considering all three datasets simultaneously, particleMDI is able to recover the true underlying structure of the data.

Figure 3 shows the empirical effect on computation time as a function of observations, dimensions, and the number of particles used. Where the relevant parameters are not altered, particleMDI is run for $n = 1000$, $M = 32$, clustering two Gaussian and one categorical dataset with $n = 150$ observations with 16 dimensions. In all cases 25% of dimensions are drawn as random noise. All analyses were performed in Julia 0.6.4 on a Windows laptop with a 2.80GHz Intel Core i7-7700HQ CPU and 32.0 GB RAM. We contrast the computation times between two implementations of the algorithm: one which benefits from the performance improvements obtained by exploiting the redundancy of the particle filter as outlined in Sec. 2.1; and one without these improvements. As we are only avoiding performing redundant calculations, these improvements do not come at the cost of any decrease in accuracy. Figure 3 shows we can improve computation time by more than an order of magnitude.

In order to assess the impact of ρ, we examine cluster accuracy from analysis on Fisher's iris dataset[6] for varying levels of ρ. We assess cluster accuracy by means of the adjusted Rand index [15]—a measure of agreement between two partitions adjusted for agreement by chance, a value of 1 indicating perfect agreement, and 0 indicating agreement no better than chance. As expected, very large ρ values lead to slow mixing of the Gibbs sampler, leading to many iterations before the algorithm converges. Interestingly though, very small values of ρ appear to be more problematic, with values of $\rho = 0.05$ and $\rho = 0.1$ struggling to get close to the ground truth. As discussed in Sec. 2.1 when ρ is very small, the conditional particle filter has little information on which to base allocations for observations it encounters at the beginning, inducing a strong dependence on the order of the observations.

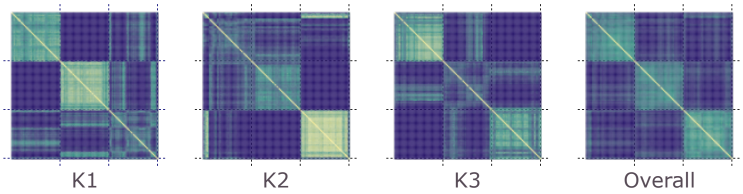

Fig. 2 Heatmap representation of the posterior similarity matrices as output from `generate_psm()` and `consensus_map` for three Gaussian datasets (K1, K2, K3) with different degrees of overlap in clusters across data sets. The brightness of point (i, j) in each reflects the empirical probability that observations i and j are clustered together in each dataset, with these values averaged across datasets to give the value in 'overall'.

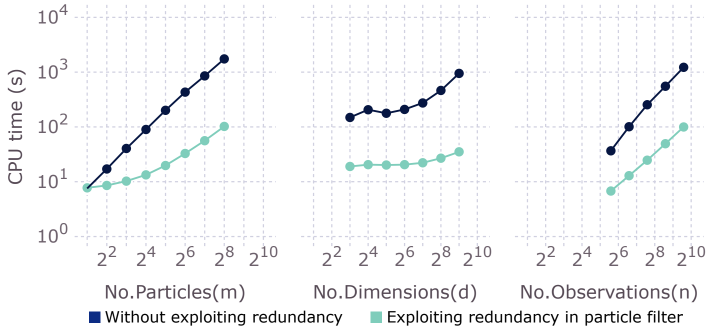

Fig. 3 Computation time as a function of the number of particles, the number of dimensions, and the number of observations respectively. The results show that reducing inefficient calculations can contribute to greater than an order of magnitude improvement in computation time.

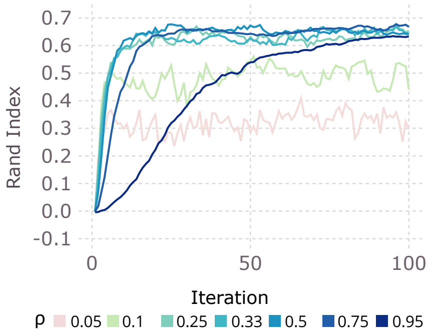

Fig. 4 The effect on cluster accuracy as a function of ρ from analyses on Fisher's iris dataset. Analyses were performed 10× and adjusted Rand index values per iteration were averaged across runs. The results suggest extreme values can negatively influence the output while there is little observable difference between thresholds in the range $0.25 - 0.5$.

5 Discussion

In this paper, we have presented particleMDI, a Julia package implementing a particle Monte Carlo approach to the integrative cluster analysis of multiple data sets. We have demonstrated the capability of the package to uncover the ground truth cluster structure in a group of synthetic datasets of different data types. In Sec. 3.1 we showed how this package can perform cluster analysis on any data type for which a posterior predictive distribution can be specified. We outlined methods for improving computational performance of our algorithm in Sec. 2.1 and demonstrated that these approaches can achieve performance improvements of an order of magnitude or more in terms of computation time. While the context of our work is in integrative cluster analyses, these approaches are also applicable to the single-data context of cluster analysis using particle Monte Carlo methods.

All files relevant to this package are available on Github. (`https://github.com/nathancunn/particleMDI.jl`)

References

[1] Andrieu, C., Doucet, A., Holenstein, R.: Particle Markov chain Monte Carlo methods. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **72**(3), 269–342 (2010)

[2] Bezanson, J., Edelman, A., Karpinski, S., Shah, V.: Julia: A fresh approach to numerical computing. SIAM Review **59**(1), 65–98 (2017). DOI 10.1137/141000671. URL `https://doi.org/10.1137/141000671`

[3] Bouchard-Côté, A., Doucet, A., Roth, A.: Particle Gibbs split-merge sampling for Bayesian inference in mixture models. Journal of Machine Learning Research **18**(28), 1–39 (2017)

[4] Chopin, N.: A sequential particle filter method for static models. Biometrika **89**(3), 539–552 (2002)

[5] Fearnhead, P.: Particle filters for mixture models with an unknown number of components. Statistics and Computing **14**(1), 11–21 (2004)

[6] Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annals of Eugenics **7**(2), 179–188 (1936)

[7] Gabasova, E., Reid, J., Wernisch, L.: Clusternomics: Integrative context-dependent clustering for heterogeneous datasets. PLoS Computational Biology **13**(10), e1005781 (2017)

[8] Griffin, J.: Sequential Monte Carlo methods for mixtures with normalized random measures with independent increments priors. Statistics and Computing pp. 1–15 (2014)

[9] Ishwaran, H., Zarepour, M.: Exact and approximate sum representations for the Dirichlet process. Canadian Journal of Statistics **30**(2), 269–283 (2002)

[10] Jones, D.C., Arthur, B., Nagy, T., Gowda, S., Godisemo, Holy, T., Noack, A., Sengupta, A., Darakananda, D., Mattriks, Leblanc, S., Dunning, I., Fischer, K., Chudzicki, D., Yu, Y., Breloff, T., Kleinschmidt, D., Mellnik, A., john verzani, inkyu, Innes, M.J., Huchette, J., Bauman, M., Buzby, K., Hyatt, K., Forsyth, J., Borje, G., Saba, E., Coalson, C., Pelenitsyn, A.: Giovineitalia/gadfly.jl: v0.7.0 (2018). DOI 10.5281/zenodo.1284282. URL `https://doi.org/10.5281/zenodo.1284282`

[11] Kirk, P., Griffin, J.E., Savage, R.S., Ghahramani, Z., Wild, D.L.: Bayesian correlated clustering to integrate multiple datasets. Bioinformatics **28**(24), 3290–3297 (2012)

[12] McParland, D., Gormley, I.C., McCormick, T.H., Clark, S.J., Kabudula, C.W., Collinson, M.A.: Clustering South African households based on their asset status using latent variable models. The Annals of Applied Statistics **8**(2), 747 (2014)

[13] Monti, S., Tamayo, P., Mesirov, J., Golub, T.: Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data. Machine learning **52**(1-2), 91–118 (2003)

[14] Neal, R.M.: Markov chain sampling methods for Dirichlet process mixture models. Journal of computational and graphical statistics **9**(2), 249–265 (2000)

[15] Rand, W.M.: Objective criteria for the evaluation of clustering methods. Journal of the American Statistical Association **66**(336), 846–850 (1971). DOI 10.1080/01621459.1971.10482356. URL `https://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356`

[16] Rasmussen, C., de la Cruz, B., Ghahramani, Z., Wild, D.: Modeling and visualizing uncertainty in gene expression clusters using Dirichlet process mixtures. IEEE/ACM transactions on computational biology and bioinformatics **6**(4), 615–628 (2009)

[17] Shen, R., Olshen, A.B., Ladanyi, M.: Integrative clustering of multiple genomic data types using a joint latent variable model with application to breast and lung cancer subtype analysis. Bioinformatics **25**(22), 2906–2912 (2009)