Swarm: robust and fast clustering method for amplicon-based studies

Frédéric Mahé[1,2,3], Torbjørn Rognes[4,5], Christopher Quince[6], Colomban de Vargas[1,2], and Micah Dunthorn[3]

[1]**CNRS, UMR 7144, EPEP – Évolution des Protistes et des Écosystèmes Pélagiques, Station Biologique de Roscoff, 29680 Roscoff, France.**
[2]**Sorbonne Universités, UPMC Univ Paris 06, UMR7144 Station Biologique de Roscoff, Roscoff, France.**
[3]**Department of Ecology, University of Kaiserslautern, Kaiserslautern, Germany.**
[4]**Department of Microbiology, Oslo University Hospital, Rikshospitalet, Oslo, Norway.**
[5]**Department of Informatics, University of Oslo, Oslo, Norway.**
[6]**School of Engineering, University of Glasgow, Glasgow, UK.**

ABSTRACT

Popular *de novo* amplicon clustering methods suffer from two fundamental flaws: arbitrary global clustering thresholds, and input-order dependency induced by centroid selection. Swarm was developed to address these issues by first clustering nearly identical amplicons iteratively using a local threshold, and then by using clusters' internal structure and amplicon abundances to refine its results. This fast, scalable, and input-order independent approach reduces the influence of clustering parameters and produces robust operational taxonomic units, improving the amount of meaningful biological information that can be extracted from amplicon-based studies.

Keywords: environmental diversity, barcoding, molecular operational taxonomic units

INTRODUCTION

High-throughput sequencing technologies can generate millions of amplicons (or barcode sequences) in a single run, and are thus today our best approach to deeply assess the environmental or clinical diversity of complex microbial assemblages of archaea, bacteria, and eukaryotes. The millions, and soon billions, of raw reads produced in molecular ecology and metabarcoding projects need to be clustered into molecular operational taxonomic units (OTUs) before being used for diversity estimates or other statistical analyses.

Because of the increasing sizes of today's amplicon datasets, fast and greedy *de novo* clustering heuristics are the preferred and the only practical approach to produce OTUs (Edgar, 2010; Ghodsi et al., 2011; Fu et al., 2012). Shared steps in these current algorithms are: an amplicon is drawn out of the amplicon pool and becomes the center of a new OTU (centroid selection), this centroid is then compared to all other amplicons remaining in the pool. Amplicons for which the distance is within a global clustering threshold, t, to the centroid are moved from the pool to the OTU. The OTU is then closed. These steps are repeated as long as amplicons remain in the pool (Fig. 1a).

These greedy clustering methods suffer from two fundamental problems. First, they use an arbitrary fixed global clustering threshold. As lineages evolve at variable rates, no single cut-off value can accommodate the entire tree of life. A single global clustering threshold will inevitably be too relaxed for slow-evolving lineages and too stringent for rapidly evolving ones (Stackebrandt and Goebel, 1994; Sogin et al., 2006; Nebel et al., 2011; Koeppel and Wu, 2013). Second, the input order of amplicons strongly influences the clustering results. Previous centroid selections are not re-evaluated as clustering progresses, which can generate inaccurately formed OTUs, where closely related amplicons can be separated and unrelated amplicons can be grouped (Koeppel and Wu, 2013) (Fig. 1a).

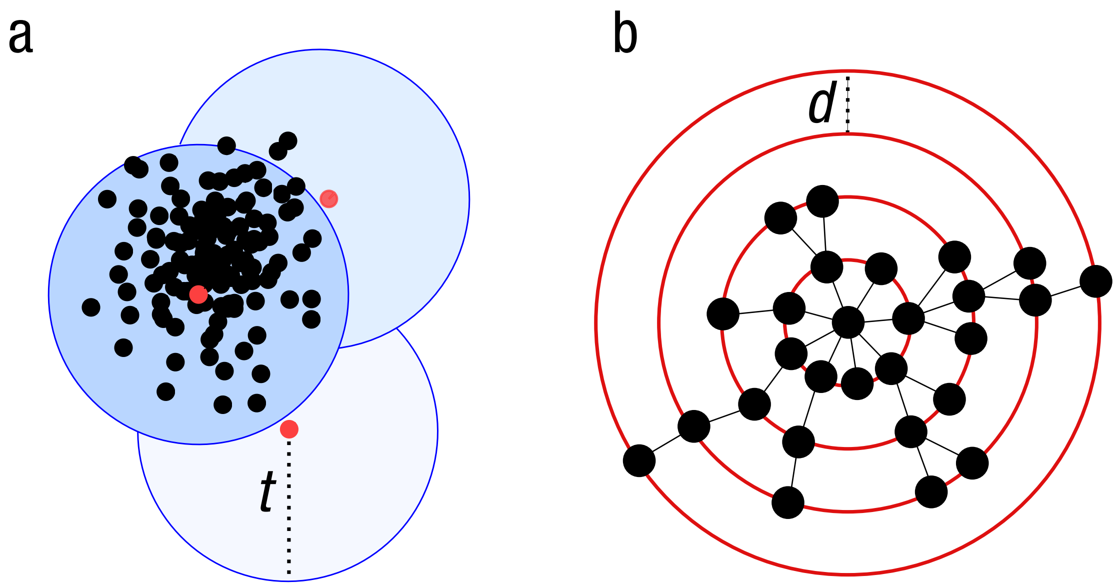

Figure 1. Visualization of the widely used greedy clustering approach based on centroid selection and a global clustering threshold, *t*, where closely related amplicons can be placed into different OTUs. (b) By contrast, Swarm clusters iteratively by using a small local clustering threshold, *d*, allowing OTUs to reach their natural limits.

SWARM'S RATIONALE

While working on two large scale environmental diversity studies using different markers and different sequencing platforms—the BioMarKs project (e.g. Bittner et al., 2013; Dunthorn et al., 2014; Logares et al., 2014) and the TARA OCEANS project (e.g. Karsenti et al., 2011)—the limitations of greedy *de novo* clustering methods became salient. We observed that amplicons from one species can be subsumed into the OTU of a genetically closely related species with a very dissimilar ecology if that second species had a higher abundance, leading to erroneous ecological interpretations. To solve these issues, we developed Swarm—a novel method that avoids both fixed global clustering thresholds, and input-order dependency due to centroid selection. Our objective was to implement an exact, yet fast, *de novo* clustering method that produces meaningful OTUs and reduces the influence of clustering parameters.

Swarm can be defined as a fast and exact agglomerative, unsupervised (*de novo*) single-linkage-clustering algorithm that first computes sequence differences between aligned pairs of amplicons to delineate OTUs, using *k*-mer comparisons and a new and extremely fast global pairwise alignment algorithm; and in a second step, uses amplicon abundance information and OTUs' internal structures to refine the clustering results.

In our view, amplicons can be seen as discrete coordinates in an abstract amplicon-space. Each position of this space represents a possible amplicon, which can be absent from the dataset (null abundance), or present (observed abundance). The direct neighbors of a given amplicon are all the possible amplicons with one nucleotide difference (a mutation, insertion, or deletion). In that amplicon-space, the clusters are regions of the space with non-null abundances. The assumption behind Swarm is that clusters are clearly separated by empty regions, i.e., amplicons do not form a continuum. If this condition holds true, then OTUs can be allowed to grow iteratively until they reach their natural limits.

Swarm explores the amplicon-space as follows: Swarm processes the input file and creates a pool of amplicons. An empty OTU is created, and the first available amplicon in the pool is withdrawn from the pool to become the OTU seed. The seed is then compared to all amplicons remaining in the pool, and the measured number of differences is stored in the memory. The number of differences *d*, the local clustering threshold, is calculated as the number of nucleotide mismatches (mutation, insertion, or deletion) between two amplicons once the optimal pairwise global alignment has been found. Amplicons for which the number of differences is equal to or less than *d* are removed from the pool and added to the OTU, where they become subseeds.

Each subseed is then compared to the amplicons remaining in the pool, but only to those that have at most $2d$ differences with the seed. Indeed, amplicons with more than $2d$ differences with the seed cannot have d or less differences with one of its subseeds. This filtering step can be generalized as such: a subseed of generation k, a k-seed, is compared to the amplicons remaining in the pool, but only to those that have at most $(k+1)d$ differences with the seed.

After each series of comparisons, amplicons for which the number of differences is equal to or less than d are removed from the pool and added to the OTU to become $(k+1)$-seeds. This iterative growth process is repeated for each generation of subseeds as long as new amplicons are captured. The OTU is then closed. The first available amplicon is removed from the pool, a new OTU is created, and the process is repeated until no more amplicons remain in the pool.

This clustering process generates stable OTUs, regardless of the first seed choice. Thus, an OTU organically grows to its natural limits where it cannot recruit any more amplicons with d or fewer differences. Operating in this way, Swarm removes the two main sources of variability inherent in greedy *de novo* clustering methods: the need to designate an OTU center (centroid selection), and the need for an arbitrary global clustering threshold. Swarm outlines OTUs without imposing one particular shape or size, and produces the same OTUs regardless of the initially selected amplicon.

Under certain conditions, when using short or slowly evolving markers for instance, the assumption that amplicons do not form a continuum can be violated. To solve this issue, Swarm takes a post-clustering step, and uses the internal structure of the cluster and abundance values to find the natural frontiers and to delineate higher-resolution OTUs (see “Refining the clusters”).

SWARM’S MECHANICS

Swarm input files

Swarm input is a standard fasta file of unambiguous DNA or RNA amplicons, preferably fully sequenced (from the forward to the reverse primer), and with unique identifiers. The amplicon identifier is defined as the string comprised between the “>” and the first space or the end of the line, whichever come first. Swarm expects amplicon identifiers to be unique to avoid ambiguity in the clustering results. The amplicon sequence can only be composed of A, C, G and Ts (or Us) (case insensitive), and Swarm exits with an error message if any other symbol is present. To reduce the volume of data and to further increase analysis speed, it is recommended to merge amplicons with strictly identical sequences (*dereplication*). The Swarm README file gives examples of commands to properly format, and dereplicate the input fasta file.

Avoiding unnecessary pairwise global alignments

Global pairwise alignment is an exact but computationally expensive way to count the number of differences between two amplicons. It is possible to avoid costly comparisons by simply comparing amplicon lengths: two amplicons cannot have less than d differences if their length difference is greater than d. Unfortunately, that costless filtering is rather limited in scope. A more general and powerful way to estimate amplicon identity is to compare k-mer compositions (Ukkonen, 1992). Swarm starts by listing all possible DNA oligomers of length 5 (i.e. 5-mers; $4^5 = 1024$ possibilities). For a given amplicon, Swarm counts the number of occurrences of each 5-mer and builds a 1024-bit vector of zeros (absence or even number of occurrences) and ones (odd number of occurrences). For d differences between two amplicon, the maximum number of differences in their k-mer vectors is $2dk$. Therefore, two amplicons with one difference in their sequences will have at most 10 differences in their 5-mer vectors. Only pairs of amplicons with 10 or less differences in their 5-mer vectors need to be considered as candidates for a pairwise global alignment. It is possible for a pair of dissimilar amplicons to have similar bit vectors, but it is not possible for two similar amplicons to have dissimilar bit vectors. In other words, false positives are possible but not false negatives. As comparing bit vectors can be done rapidly in modern computers using XOR and POPCOUNT bitwise instructions, Swarm can efficiently, and in a lossless way, filter out many unnecessary pairwise global alignments.

Pairwise Global Alignment Implementation

To speed up the remaining pairwise comparisons, Swarm implements a novel exact global pairwise alignment algorithm (Needleman and Wunsch, 1970) using SIMD vectorial instructions of modern CPUs, similar to the local pairwise alignment (Smith and Waterman, 1981; Gotoh, 1982) implemented in Swipe by Rognes (2011). Swarm is able to compare up to 16 pool amplicons simultaneously with one seed (or *k*-seed) per CPU core. Swarm is also multi-threaded and efficiently uses multiple cores. In addition to computing the alignment score, it also stores backtracking data during amplicon comparisons in order to subsequently reconstruct the highest scoring alignments, and to count the number of differences. Since amplicons are usually of limited length (less than 1,000 bp) for the intended purpose, the amount of storage required for this data is not a major concern.

To achieve the most efficient parallelization of the algorithm, using as many parallel operations as possible (i.e., 16 simultaneous pairwise comparisons per CPU core), the magnitude of the score values of the highest scoring alignments to be calculated should be small, preferably fitting in one byte of memory (a byte can store one integer value ranging from 0 to 255). Swarm calculates the alignment score as follows: instead of computing the optimal global alignment similarity score as in the Needleman-Wunsch algorithm, Swarm identifies the alignments with the minimum edit distance, as described by Sellers (1974), by transforming the given similarity scoring system into an equivalent edit distance system. In the default scoring system a match is given a score of $+5$, a mismatch -4, gap opening -12, and gap extension -4. Swarm will transform this scoring scheme into an equivalent system of positive integers where each mismatch corresponds to a penalty of $+9$, gap-opening $+12$, and gap extension $+9$. These distances are all divided by their greatest common divisor, 3, resulting in the final edit distances of 3 for mismatches, 4 for gap openings, and 3 for gap extensions. This modified scoring system yields pairwise alignments strictly identical to the pairwise alignments produced by the original scoring system.

Swarm first tries to compute the global pairwise alignment using a single byte of memory, allowing scores up to 255, which corresponds to a maximum of 85 differences. Should the score be larger than 255, Swarm will re-compute the pairwise alignment using 2 bytes of memory or more. Using two bytes of memory reduces the speed of the pairwise alignment by 50%; however the upstream *k*-mer filtering implemented in Swarm limits the number of alignments overflowing one byte of memory.

After having identified the best alignment(s), Swarm will backtrack the optimal alignment (or one of the co-optimal alignments) using the backtracking data saved earlier. It will then simply count the number of mismatches and indels in that alignment and use it as the final difference between two amplicons.

In a very few cases the co-optimal alignments might contain a different number of gaps. As a result, the final number of differences counted may depend on which alignment is followed during the backtracking. The effect of this alignment issue is that a very small variability may exist in some results. We are currently solving this minor issue, and it will be implemented in a future Swarm release.

Refining the clusters

Swarm is an agglomerative, unsupervised, single-linkage-clustering algorithm. Single-linkage clustering is known to produce chains of amplicons that can potentially link closely related OTUs and decrease clustering resolution (Huse et al., 2010). As Swarm works with small local clustering threshold values, this chaining effect is rather limited. Nevertheless, this risk exists, especially when using short or slowly evolving molecular markers. To solve this issue, we implemented an algorithm that explores the internal structure of the OTUs produced by Swarm, and detects and breaks possible chains by using abundance values associated with each amplicon.

OTUs present an internal structure where the most abundant amplicon usually occupies a central position and is surrounded by less abundant amplicons. The way Swarm explores the amplicon-space naturally produces a graph representation of the OTU, in the form of a star-shaped minimum spanning tree. In this context, chains of amplicons appear as links between such star-shaped sub-graphs. To identify and break the chains, our algorithm finds paths linking abundant amplicons (*peaks*) and monitors the abundance variations along these paths. If abundances decrease, go through a minimum, and then go up again (*valley shape*), this indicates a possible amplicon chain. Depending on the depth of the valley (i.e. the ratio between the minimum and maximum observed abundances), the algorithm will decide whether or not to break the graph into independent OTUs. In its current form, this step in Swarm is a python companion script set with the following parameters: only amplicons with 100 or more copies are considered as peaks; for a given pair of peaks, the path always starts from the highest peak (primary peak) and joins the secondary peak; valleys are considered deep when the ratio between the lowest point and the secondary peak is equal or greater than 50. Observations we made on environmental data indicated that chains of amplicons tend to form in the largest clusters, and are unlikely to occur in clusters with peaks smaller than 100. This small clusters are not removed from the clustering results, they are merely not searched for potential chains.

As this OTU breaking step improves Swarm's precision at all clustering levels, we recommend to always apply this companion script. We are currently testing a faster, less parametrized and more elegant solution using graph properties to quickly identify potential breaking points. We plan to include it soon in Swarm itself.

On the influence of the value *d*

Swarm's most important user-chosen parameter is d, the local clustering threshold (for other parameters, see the Swarm Manual). Empirical results show that the choice of the d value has far less impact on the clustering results than the choice of the global clustering threshold (see results below where we show that the choice of d has minimal impact). By default, d is set to 1 to obtain the finest partition of the amplicon-space, and to maximize the yield of biological information. Several factors can motivate the use of higher d values: longer amplicons, fast evolving markers, shallow sequencing (i.e., under-sampling of the amplicon-space), or the need to work with fewer but more inclusive OTUs. The stability of Swarm results also allows hierarchical clustering approaches: tracking OTU coalescing events when d increases allows to identify the d value best suited for the targeted amplicon-space region. We invite users to test several d values to find the value, or the range of values, best fitting their data and scientific questions.

Speed and general behavior

Swarm clustering is a non-linear process. For speed purposes, it is suggested to sort the input data by decreasing amplicon abundance (if applicable); however results will not change, only the speed. Abundant amplicons often occupy a central position in OTUs, and when used as seeds, rapidly capture large amounts of subseeds. Using this strategy, the first OTUs produced are generally among the largest. As OTUs are outputted as soon as they are closed, most of the input amplicons can be quickly removed from the pool early in the clustering process, and can be passed to post-clustering analyses. The duration of a Swarm analysis depends on several parameters: e.g. the d value used, the amplicon number and length, the number of CPU cores available, the molecular diversity of the dataset, and the amplicon abundance distribution. While being slower than Usearch, the fastest greedy heuristic for amplicon clustering, Swarm is fast enough to deal with datasets with millions of unique amplicons in a few days. For example, the clustering of an unpublished dataset of 1.5 million raw reads (312,503 unique amplicons, 382 nucleotides on average) takes less than 10 minutes ($d = 1$) and 20 minutes ($d = 7$), using a single core. It only takes 4 minutes using 16 cores ($d = 1$).

Swarm output files

The dereplication command example provided in Swarm's README includes amplicon copy number within the amplicon identifier. This amplicon copy number information is retained in the output file, and used by Swarm to output statistics on the number of unique amplicons in the swarm, total copy number of amplicons in the swarm, identifier of the initial seed, initial seed abundance, number of singletons (amplicons with an abundance of 1), maximum number of iterations, the maximum radius of the swarm.

Swarm default behavior is to output results using a format similar to DNAclust's format (but with space separated amplicon identifiers instead of tab-separated). An optional behavior is to output results using the Usearch format, which allow easy integration with extant amplicon-analysis pipelines, such as QIIME (Caporaso et al., 2010). We are working on a similar option for easy integration with Mothur (Schloss et al., 2009).

COMPARISON WITH POPULAR *DE NOVO* CLUSTERING METHODS

As Swarm was designed to analyze extremely large high-throughput sequencing datasets, we compared it to other stand-alone, fast, *de novo* clustering methods. Other clustering methods, such as average linkage, complete linkage or hierarchical clustering, do not scale up to this large datasets, and were therefore not compared.

Swarm's performance was tested on two mock communities each comprising genome isolates from 49 bacterial and 10 archaeal species: one with even abundances with 143,163 unique amplicons (1,576,869 raw reads of average length 254.6 bp), the other with uneven populations with 55,622 unique amplicons (637,693 raw reads of average length 253.9 bp). In both cases, sequencing on the MiSeq platform with 250 bp paired-end reads was performed following amplification of the V4 region of the 16S rRNA gene with fusion Golay adaptors barcoded on the reverse end. The forward 16S rRNA primer sequence 515f was used (GTGNCAGCMGCCGCGGTAA). The reverse primers, barcodes and adaptors were identical to Caporaso et al. (2011).

Sequences were trimmed for quality and adaptors using Sickle (https://github.com/najoshi/sickle) and then forward and reverse reads were overlapped with PandaSeq (Masella et al., 2012) insisting on a 50 bp overlap. Strictly identical amplicons were merged with the command described in Swarm's README, and the two datasets were subjected to five different stand-alone clustering methods: CD-HIT-454 (Fu et al., 2012, v4.6 2012-11-12), DNAclust (Ghodsi et al., 2011, v64bit, release_3, 2013-08-12), Swarm (v1.2.3), Usearch (Edgar, 2010, v7.0.1001_i86linux32) with presorting of the amplicons by decreasing length (option -cluster_fast) and without presorting (options -usersort -cluster_smallmem). Different clustering thresholds were used: $d = 1$ to 20 local differences for Swarm, and $t = 1$ to 20% global divergence for the other methods. For each clustering threshold and each clustering method, the first analysis was done on a fasta file sorted by decreasing abundance, and repeated 100 times with amplicon input order randomly shuffled.

The clustering results of the two communities were then evaluated with three metrics using the known assignments to the 59 species as the ground truth, which was determined by matching with Usearch against the known 16S rRNA reference sequences and only using reads that were within 5% sequence difference using Usearch with the usearch_global alignment option. The three metrics used were: *recall*, which quantifies the extent to which amplicons assigned to the same species are grouped together in the same OTU (i.e. not over-splitting); *precision*, which asks if all amplicons in an OTU are assigned to the same species (i.e. not over-grouping); and the *adjusted Rand index*, which summarizes both precision and recall as the proportion of pairs of amplicons that are placed in the same OTU and are from the same species, but adjusting for the expected proportions through random chance (Rand, 1971; Hubert and Arabie, 1985). The results were synthesized in Fig. 2 (uneven community) and Fig. 3 (even community) using R (R Development Core Team, 2014) and the ggplot2 library (Wickham, 2009). The command lines and scripts we used to analyze the data, and to visualize the results are provided in the Supplementary File 1.

For both uneven (Fig. 2), and even communities (Fig. 3), almost all clustering thresholds for CD-HIT-454 and Usearch (with or without presorting) show extreme variability due to strong input-order dependency, while Swarm and DNAclust do not. Because of the pervasive variability in other methods, we then compared the median output values (Table 1 and 2). Using the adjusted Rand index for both the uneven and even populations, Swarm outperforms the other methods at small d or t values (1 to 2); these small clustering thresholds are of critical importance when making fine-scale partitions of amplicon datasets, and strongly indicate that Swarm is more robust than the other methods to sequencing noise. At medium values (3 to 5), Swarm is either better, tied, or within 0.011 points to DNAclust. Swarm outperforms all other methods at larger values (6 to 20). These superior results for Swarm reflect its ability to conserve good precision and good recall over a wide range of clustering thresholds, which is critical as the true threshold for species-level assignments will in general not be known in advance and will vary with the choice of the sequenced marker or genomic region. Additionally, the adjusted Rand index results show that Swarm results are little affected by the choice of d; that is, it limits the effect of the choice of clustering threshold and is adaptative across a large array of organisms and genes.

PERSPECTIVES

Swarm is efficient enough to deal with today's largest datasets, and several new optimizations are now in development to handle even larger future datasets. For example, with the 256-bit integer SIMD instructions of the new Intel Haswell CPUs (released in late 2013) Swarm's pairwise alignment throughput can double. We are also testing more efficient parallelization strategies, as well as new filters to avoid further unneeded pairwise alignments. These hardware evolutions and software optimizations will improve Swarm's scalability even further. In parallel, improvements to our abundance-based chain breaking model will increase Swarm's capacity to produce high-resolution OTUs and meaningful biological results, even for the most intricate species complexes.

In summary, Swarm is a novel and robust approach that solves the problems of arbitrary global clustering thresholds and centroid selection induced input-order dependency, and creates robust and more natural OTUs than current greedy, *de novo*, scalable clustering algorithms. Swarm's high-resolution capacities improve the amount of biological information that can be obtained from environmental and clinical amplicon-based studies. Swarm is a scalable C++ program able to handle many millions of amplicons. It is freely available at https://github.com/torognes/swarm under the GNU Affero General Public License version 3.

ACKNOWLEDGMENTS

F.M. and C.deV. were supported by the EU EraNet BiodivErsA program BioMarKs (grant # 2008-6530) and the French government "Investissements d'Avenir" project OCEANOMICS (ANR-11-BTBR-0008) and the EU FP7 program MicroB3 (contract number 287589). F.M and M.D. were supported by the Deutsche Forschungsgemeinschaft (grant # DU1319/1-1). T.R. was supported by a Centre of Excellence grant from the Research Council of Norway to CMBN. C.Q. is funded by an EPSRC Career Acceleration Fellowship – EP/H003851/1. We would like to thank Umer Ijaz for help in running analyzes. We are grateful to the CNRS-UPMC ABiMS bioinformatics platform (http://abims.sb-roscoff.fr/) and to the Regional Computing Center at the University of Kaiserslautern for providing computational resources and support.

REFERENCES

Bittner, L., Gobet, A., Audic, S., Romac, S., Egge, E. S., Santini, S., Ogata, H., Probert, I., Edvardsen, B., and de Vargas, C. (2013). Diversity patterns of uncultured Haptophytes unravelled by pyrosequencing in Naples Bay. *Molecular Ecology*, 22(1):87–101.

Caporaso, J. G., Kuczynski, J., Stombaugh, J., Bittinger, K., Bushman, F. D., Costello, E. K., Fierer, N., Pena, A. G., Goodrich, J. K., Gordon, J. I., Huttley, G. A., Kelley, S. T., Knights, D., Koenig, J. E., Ley, R. E., Lozupone, C. A., McDonald, D., Muegge, B. D., Pirrung, M., Reeder, J., Sevinsky, J. R., Turnbaugh, P. J., Walters, W. A., Widmann, J., Yatsunenko, T., Zaneveld, J., and Knight, R. (2010). QIIME allows analysis of high-throughput community sequencing data. *Nature Methods*, 7(5):335–336.

Caporaso, J. G., Lauber, C. L., Walters, W. A., Berg-Lyons, D., Lozupone, C. A., Turnbaugh, P. J., Fierer, N., and Knight, R. (2011). Global patterns of 16S rRNA diversity at a depth of millions of sequences per sample. *Proceedings of the National Academy of Sciences of the United States of America*, 108(Supplement 1):4516–4522.

Dunthorn, M., Otto, J., Berger, S. A., Stamatakis, A., Mahé, F., Romac, S., de Vargas, C., Audic, S., The BioMarKs Consortium, Stock, A., Kauff, F., and Stoeck, T. (2014). Placing Environmental Next-Generation Sequencing Amplicons from Microbial Eukaryotes into a Phylogenetic Context. *Molecular Biology and Evolution*, 31(4):993–1009.

Edgar, R. C. (2010). Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, 26(19):2460–2461.

Fu, L., Niu, B., Zhu, Z., Wu, S., and Li, W. (2012). CD-HIT: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152.

Ghodsi, M., Liu, B., and Pop, M. (2011). DNACLUST: accurate and efficient clustering of phylogenetic marker genes. *BMC Bioinformatics*, 12(1):271.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708.

Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2:193–218.

Huse, S. M., Welch, D. M., Morrison, H. G., and Sogin, M. L. (2010). Ironing out the wrinkles in the rare biosphere through improved OTU clustering. *Environmental Microbiology*, 12(7):1889–1898.

Karsenti, E., Acinas, S. G., Bork, P., Bowler, C., De Vargas, C., Raes, J., Sullivan, M., Arendt, D., Benzoni, F., Claverie, J.-M., Follows, M., Gorsky, G., Hingamp, P., Iudicone, D., Jaillon, O., Kandels-Lewis, S., Krzic, U., Not, F., Ogata, H., Pesant, S., Reynaud, E. G., Sardet, C., Sieracki, M. E., Speich, S., Velayoudon, D., Weissenbach, J., Wincker, P., and the Tara Oceans Consortium (2011). A Holistic Approach to Marine Eco-Systems Biology. *PLoS Biology*, 9(10):e1001177.

Koeppel, A. F. and Wu, M. (2013). Surprisingly extensive mixed phylogenetic and ecological signals among bacterial Operational Taxonomic Units. *Nucleic Acids Research*, 41(10):5175–5188.

Logares, R., Audic, S., Bass, D., Bittner, L., Boutte, C., Christen, R., Claverie, J.-M., Decelle, J., Dolan, J. R., Dunthorn, M., Edvardsen, B., Gobet, A., Kooistra, W. H. C. F., Mahé, F., Not, F., Ogata, H., Pawlowski, J., Pernice, M. C., Romac, S., Shalchian-Tabrizi, K., Simon, N., Stoeck, T., Santini, S., Siano, R., Wincker, P., Zingone, A., Richards, T. A., de Vargas, C., and Massana, R. (2014). Patterns of Rare and Abundant Marine Microbial Eukaryotes. *Current Biology*, 24(8):813–821.

Masella, A., Bartram, A., Truszkowski, J., Brown, D., and Neufeld, J. (2012). PANDAseq: paired-end assembler for illumina sequences. *BMC Bioinformatics*, 13(1):31.

Nebel, M., Pfabel, C., Stock, A., Dunthorn, M., and Stoeck, T. (2011). Delimiting operational taxonomic units for assessing ciliate environmental diversity using small-subunit rRNA gene sequences. *Environmental Microbiology Reports*, 3(2):154–158.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. 48(3):443–453.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850.

R Development Core Team (2014). *R: a language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org.

Rognes, T. (2011). Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation. *BMC Bioinformatics*, 12(1):221.

Schloss, P. D., Westcott, S. L., Ryabin, T., Hall, J. R., Hartmann, M., Hollister, E. B., Lesniewski, R. A., Oakley, B. B., Parks, D. H., Robinson, C. J., Sahl, J. W., Stres, B., Thallinger, G. G., Van Horn, D. J., and Weber, C. F. (2009). Introducing mothur: Open-Source, Platform-Independent, Community-Supported Software for Describing and Comparing Microbial Communities. *Applied and Environmental Microbiology*, 75(23):7537–7541.

Sellers, P. H. (1974). On the Theory and Computation of Evolutionary Distances. *SIAM Journal on Applied Mathematics*, 26:787–793.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. 147(1):195–197.

Sogin, M. L., Morrison, H. G., Huber, J. A., Welch, D. M., Huse, S. M., Neal, P. R., Arrieta, J. M., and Herndl, G. J. (2006). Microbial diversity in the deep sea and the underexplored "rare biosphere". *Proceedings of the National Academy of Sciences of the United States of America*, 103(32):12115–12120.

Stackebrandt, E. and Goebel, B. M. (1994). Taxonomic Note: A Place for DNA-DNA Reassociation and 16S rRNA Sequence Analysis in the Present Species Definition in Bacteriology. *International Journal of Systematic Bacteriology*, 44(4):846–849.

Ukkonen, E. (1992). Approximate string-matching with *q*-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211.

Wickham, H. (2009). *ggplot2: elegant graphics for data analysis*. Springer New York.

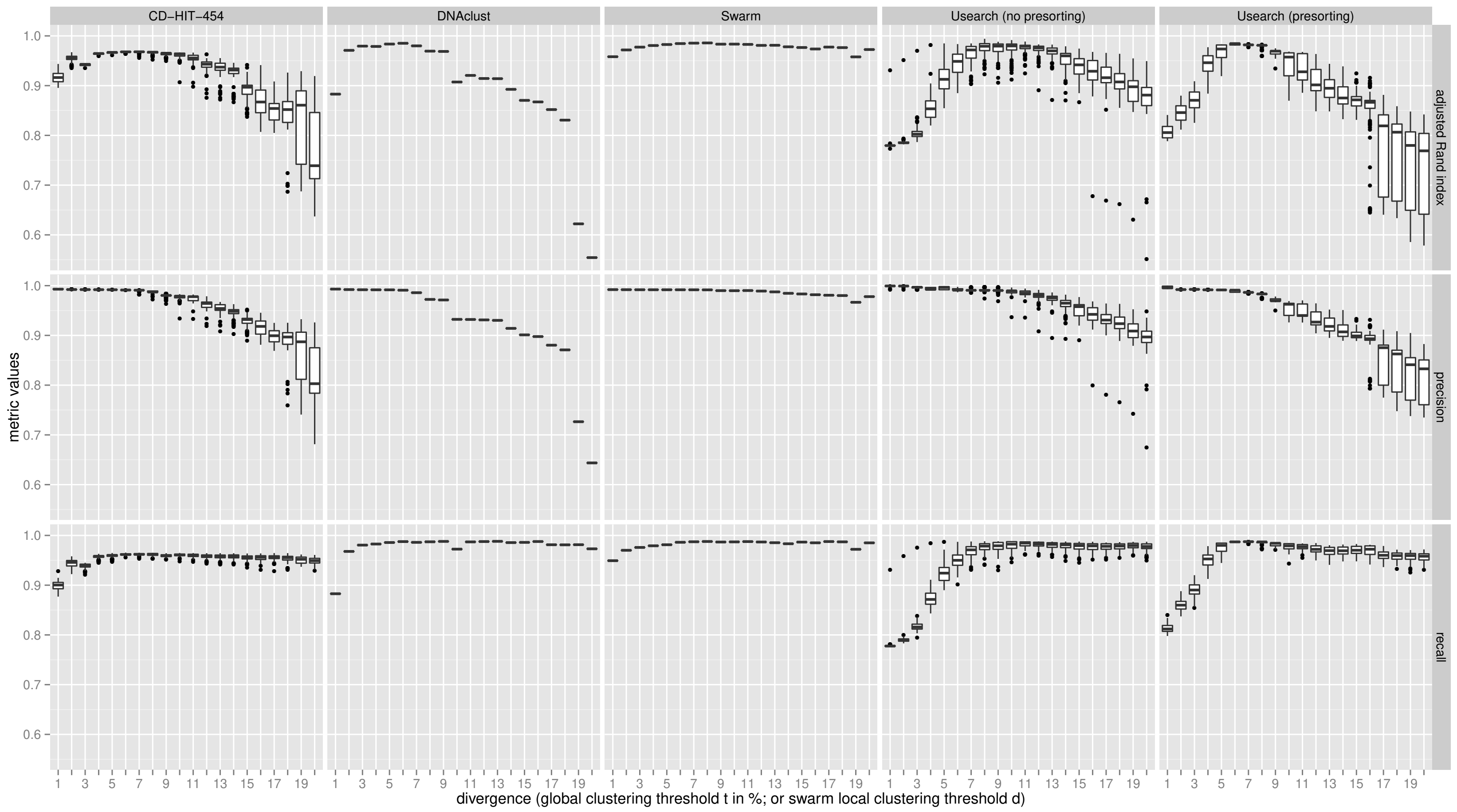

Figure 2. Uneven mock-community. Comparisons of five clustering methods, over 20 different clustering thresholds, and 100 amplicon input-order shufflings of a community composed of species of uneven abundances. Clustering *precision* and *recall* are estimated using amplicon taxonomic assignments as ground truth, and are summarized by the *adjusted Rand index*.

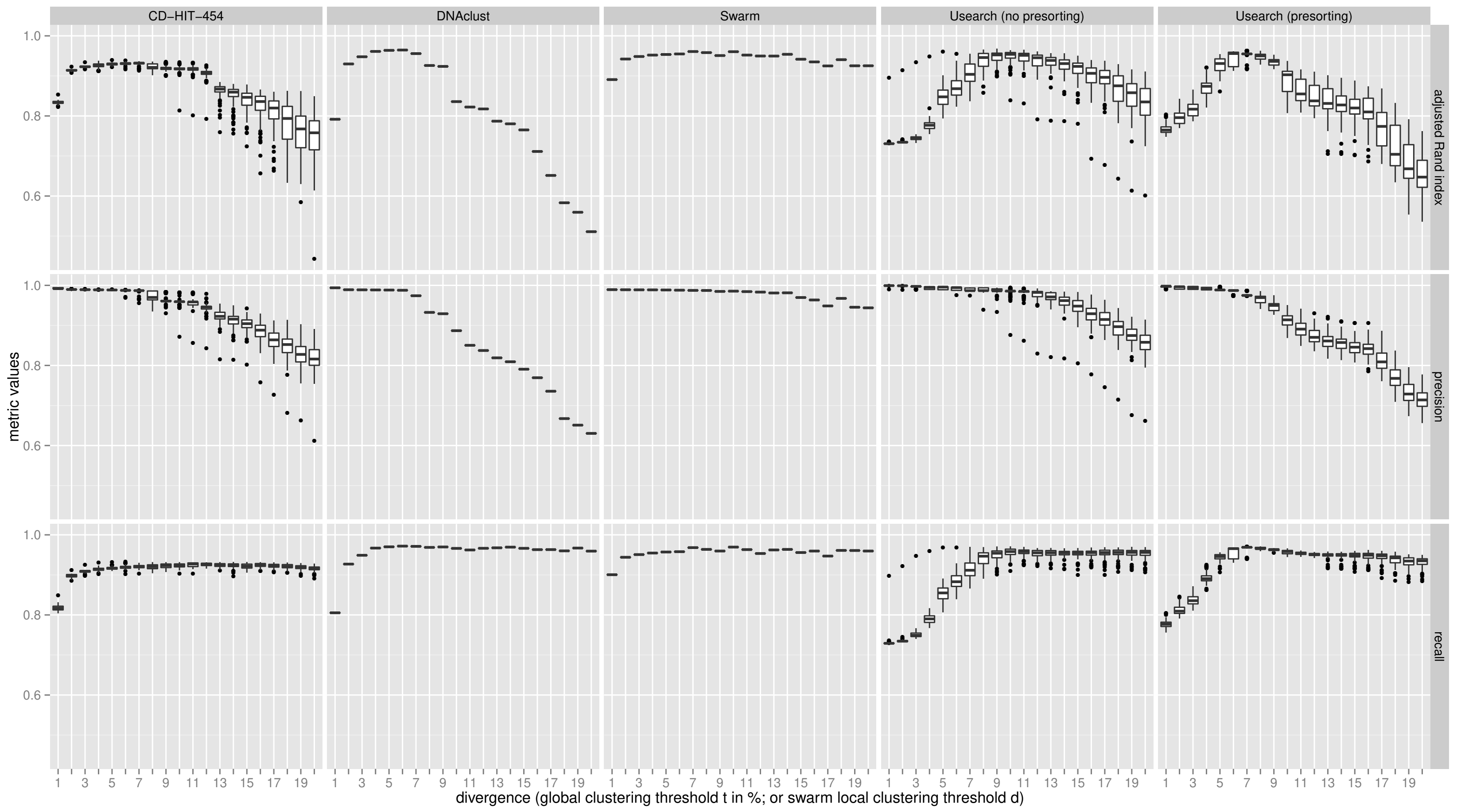

Figure 3. Even mock-community. Comparisons of five clustering methods, over 20 different clustering thresholds, and 100 amplicon input-order shufflings of a community composed of species of even abundances. Clustering *precision* and *recall* are estimated using amplicon taxonomic assignments as ground truth, and are summarized by the *adjusted Rand index*.

Divergence	Metric	Clustering algorithm				
		CD-HIT-454	DNAclust	Swarm	Usearch (no presorting)	Usearch (presorting)
1	*adjusted Rand index*	0.916	0.883	**0.958**	0.780	0.806
2		0.956	0.971	**0.972**	0.785	0.846
3		0.943	**0.979**	0.977	0.802	0.871
4		0.965	0.979	**0.981**	0.853	0.946
5		0.967	**0.984**	0.983	0.913	0.974
6		0.968	**0.985**	**0.985**	0.949	0.984
7		0.968	0.980	**0.985**	0.972	0.982
8		0.967	0.969	**0.986**	0.979	0.981
9		0.965	0.969	**0.983**	0.979	0.969
10		0.964	0.907	**0.983**	0.980	0.957
11		0.955	0.921	**0.983**	0.979	0.928
12		0.943	0.914	**0.981**	0.976	0.901
13		0.937	0.914	**0.981**	0.970	0.895
14		0.931	0.893	**0.978**	0.960	0.875
15		0.897	0.870	**0.977**	0.942	0.871
16		0.867	0.867	**0.974**	0.929	0.867
17		0.854	0.852	**0.978**	0.916	0.819
18		0.852	0.831	**0.976**	0.907	0.806
19		0.861	0.622	**0.958**	0.898	0.780
20		0.739	0.554	**0.973**	0.881	0.769
1	*precision*	0.993	0.993	0.992	**0.999**	0.996
2		0.992	0.992	0.992	**0.999**	0.992
3		0.992	0.992	0.992	**0.997**	0.993
4		**0.992**	**0.992**	**0.992**	**0.992**	**0.992**
5		0.992	0.992	0.992	**0.996**	0.991
6		0.991	0.991	**0.992**	0.991	0.991
7		0.991	0.986	**0.992**	0.991	0.986
8		0.987	0.972	**0.992**	0.990	0.983
9		0.980	0.971	**0.990**	**0.990**	0.970
10		0.979	0.932	**0.990**	**0.990**	0.963
11		0.978	0.932	**0.990**	0.985	0.940
12		0.964	0.931	**0.989**	0.981	0.927
13		0.954	0.930	**0.987**	0.976	0.918
14		0.948	0.914	**0.985**	0.965	0.907
15		0.932	0.901	**0.983**	0.958	0.899
16		0.918	0.898	**0.982**	0.942	0.893
17		0.899	0.880	**0.981**	0.931	0.875
18		0.897	0.871	**0.980**	0.924	0.863
19		0.887	0.726	**0.967**	0.909	0.841
20		0.803	0.644	**0.978**	0.897	0.833
1	*recall*	0.900	0.883	**0.949**	0.778	0.812
2		0.946	0.968	**0.970**	0.790	0.860
3		0.940	**0.981**	0.976	0.816	0.890
4		0.958	**0.983**	0.979	0.871	0.953
5		0.960	**0.986**	0.981	0.924	0.980
6		0.962	**0.988**	0.986	0.950	0.987
7		0.962	0.986	**0.987**	0.971	0.987
8		0.962	0.987	**0.988**	0.978	0.987
9		0.959	**0.988**	0.987	0.979	0.984
10		0.961	0.972	**0.987**	0.982	0.979
11		0.961	0.987	**0.988**	0.985	0.977
12		0.958	**0.988**	0.987	0.983	0.972
13		0.958	**0.988**	0.985	0.982	0.969
14		0.959	**0.986**	0.983	0.981	0.969
15		0.956	0.986	**0.987**	0.980	0.970
16		0.956	**0.988**	0.985	0.979	0.972
17		0.957	0.981	**0.988**	0.978	0.960
18		0.955	0.981	**0.987**	0.979	0.959
19		0.952	**0.981**	0.972	0.980	0.959
20		0.949	0.973	**0.985**	0.978	0.958

Table 1. Uneven mock-community. Median values for Figure 2.

Divergence	Metric	Clustering algorithm				
		CD-HIT-454	DNAclust	Swarm	Usearch (no presorting)	Usearch (presorting)
1		0.834	0.792	**0.891**	0.731	0.765
2		0.914	0.930	**0.942**	0.734	0.796
3		0.923	**0.948**	**0.948**	0.744	0.817
4		0.928	**0.961**	0.952	0.777	0.874
5		0.930	**0.964**	0.953	0.848	0.931
6		0.931	**0.965**	0.955	0.868	0.956
7		0.932	0.956	**0.961**	0.904	0.955
8		0.921	0.926	**0.958**	0.946	0.950
9		0.919	0.924	0.951	**0.952**	0.936
10	*adjusted Rand index*	0.917	0.836	**0.960**	0.954	0.903
11		0.917	0.822	**0.952**	0.951	0.855
12		0.908	0.818	**0.950**	0.945	0.838
13		0.867	0.787	**0.950**	0.939	0.831
14		0.859	0.780	**0.954**	0.931	0.827
15		0.847	0.765	**0.942**	0.924	0.820
16		0.836	0.711	**0.935**	0.906	0.810
17		0.820	0.651	**0.925**	0.896	0.774
18		0.794	0.583	**0.941**	0.875	0.704
19		0.768	0.559	**0.925**	0.858	0.668
20		0.758	0.511	**0.925**	0.835	0.647
1		0.992	0.994	0.989	**0.999**	0.998
2		0.990	0.989	0.989	**0.999**	0.997
3		0.989	0.989	0.989	**0.997**	0.995
4		0.989	0.989	0.989	**0.995**	0.992
5		0.988	0.988	0.988	**0.996**	0.988
6		0.987	0.988	0.988	**0.994**	0.987
7		**0.987**	0.974	**0.987**	**0.987**	0.975
8		0.970	0.932	**0.987**	**0.987**	0.969
9		0.961	0.929	0.985	**0.986**	0.950
10	*precision*	0.959	0.887	0.985	**0.986**	0.914
11		0.959	0.850	0.984	**0.985**	0.891
12		0.946	0.837	**0.983**	0.982	0.870
13		0.923	0.819	**0.981**	0.971	0.861
14		0.916	0.809	**0.981**	0.962	0.857
15		0.904	0.791	**0.970**	0.948	0.846
16		0.888	0.769	**0.964**	0.929	0.842
17		0.864	0.736	**0.949**	0.915	0.809
18		0.852	0.667	**0.967**	0.897	0.768
19		0.828	0.651	**0.945**	0.875	0.729
20		0.816	0.630	**0.944**	0.858	0.714
1		0.817	0.805	**0.900**	0.729	0.777
2		0.899	0.927	**0.944**	0.734	0.809
3		0.909	0.949	**0.951**	0.749	0.836
4		0.914	**0.967**	0.955	0.790	0.891
5		0.916	**0.970**	0.957	0.855	0.946
6		0.919	**0.972**	0.958	0.883	0.966
7		0.921	**0.971**	0.968	0.912	0.969
8		0.921	**0.969**	0.964	0.947	0.965
9		0.924	**0.970**	0.960	0.954	0.963
10	*recall*	0.923	0.966	**0.969**	0.958	0.958
11		0.927	0.962	**0.963**	0.958	0.954
12		0.927	**0.966**	0.953	0.955	0.951
13		0.926	**0.968**	0.962	0.955	0.950
14		0.925	**0.969**	0.964	0.955	0.950
15		0.922	**0.966**	0.956	0.955	0.949
16		0.925	**0.963**	0.960	0.956	0.949
17		0.923	**0.963**	0.947	0.956	0.948
18		0.921	0.960	**0.961**	0.956	0.943
19		0.919	**0.967**	0.961	0.956	0.935
20		0.915	0.959	**0.960**	0.956	0.935

Table 2. Even mock-community. Median values for Figure 3.