Accelerating Exhaustive Pairwise Metagenomic Comparisons

Esteban Pérez-Wohlfeil[1], Oscar Torreno[1], and Oswaldo Trelles[1]✉⋆

[1] Department of Computer Architecture, University of Malaga,
Boulevard Louis Pasteur 35, Malaga, Spain
{estebanpw,oscart,ortrelles}@uma.es

Abstract. In this manuscript, we present an optimized and parallel version of our previous work IMSAME, an exhaustive gapped aligner for the pairwise and accurate comparison of metagenomes. Parallelization strategies are applied to take advantage of modern multiprocessor architectures. In addition, sequential optimizations in CPU time and memory consumption are provided. These algorithmic and computational enhancements enable IMSAME to calculate near optimal alignments which are used to directly assess similarity between metagenomes without requiring reference databases. We show that the overall efficiency of the parallel implementation is superior to 80% while retaining scalability as the number of parallel cores used increases. Moreover, we also show that sequential optimizations yield up to 8x speedup for scenarios with larger data.

Keywords: High Performance Computing · Pairwise Comparison · Parallel Computing · Next Generation Sequencing · Metagenome Comparison

1 Background

A metagenome is defined as a collection of genetic material directly recovered from the environment. In particular, a metagenome is composed of a large number of reads (DNA strings) drawn from the species present in the original population. To this day, the field of comparative metagenomics has become big-data driven [1] due to new technological improvements in high-throughput sequencing. However, the analysis of large metagenomic datasets represents a computational challenge and poses several processing bottlenecks, specially to sequence comparison algorithms.

Traditional metagenomics comparison involve intermediate pairwise (and individual) comparisons against a reference database. This procedure allows to extract a mapping distribution between reads and species, and thus enables to later on compare these distributions. A similarity measure can then be computed from the two distributions. However, due to the unknown and complex composition

⋆ Corresponding author.

of metagenomes, traditional comparisons based on a reference require databases to be large, which often introduce bias and drastically increase execution times. In this line, direct comparisons between metagenomes to assess overall similarity gain interest as running times can be shortened and bias avoided. Furthermore, there is yet no accepted consensus on how similarity should be assessed, and in several scenarios certainty comes at the expense of exhaustive and optimal alignments. Still, optimal alignment of large datasets is not feasible without making use of parallel infrastructure and optimization techniques.

Next Generation Sequencing platforms are generating larger amounts of data per run, of higher quality and at a lower price. However, the performance of computational approaches used to process metagenomic data suffer inversely proportional to that of the size of generated samples. High Performance Computing techniques can be applied in order to overcome the processing bottlenecks and accelerate running times.

Several parallelism strategies have been already applied to software for both comparative genomics and metagenomics, such as the multipurpose BLAST [2] family, where different types of architectures have been exploited (e.g. mpiBLAST [3] for distributed memory or TERABLAST [4] for its use on FPGAs). Parallelization of sequence alignment algorithms have also been applied to GPUs such as GSWABE [5] or CUSHAW2-GPU [6]. FPGAs have also been employed to accelerate sequence comparisons (e.g. SWAPHI [7]). However, GPUs and FPGAs are expensive and their specificity often force the use of a reduced subset of programs due to platform dependence restrictions. Other general parallel approaches which make use of CPU multithreading such as BOWTIE [8] or PARALLEL-META3 [10] use POSIX threads [11] for UNIX-based environments in a shared memory architecture.

However, the above mentioned sequence aligners are not specifically designed to compare read to reads or contigs and particularly not to assess similarity between metagenomes. For instance, BOWTIE works best when aligning short reads to large reference genomes. Moreover, in [12] the effect of introducing intermediary agents to ultimately assess similarity between metagenomes was argued. In this line, alternative approaches that were capable of direct comparisons were discussed (e.g. MASH [13], SIMKA [14]) and when possible (BLAST, COMMET [15]), compared to IMSAME. Additionally, it was shown that coarse-grained approaches to metagenomics comparison could lead to results that were highly dependent on hyperparameters (such as the initial seed size).

To fulfill the gap, IMSAME was presented as a parallel, fine-grained, and exhaustive gapped read-to-read (including contigs and scaffolds) aligner. In this manuscript, we present an optimized version of IMSAME which is able to compute faster while using a linear and controlled amount of memory. Moreover, High Performance Computing techniques have been applied to balance the workload among threads to reduce thread synchronization.

2 Methods

IMSAME (”Incremental Multi-Stage Alignment of MEtagenomes”) is intended to compare reads to reads directly, i.e. without using a reference database, and to assess similarity between them while providing a confident level of certainty. It proceeds by combining a different set of alignment-free, gapped-free and gapped alignments. Each of these procedures is intended to yield a different level of speed and sensitivity, depending on the alignment stage. For instance, the initial detection of seeds between reads (to be referred as hits) is performed using k-mers (words of length k), whereas probabilistic filtering is applied when hits are extended into High-scoring Segment Pairs (HSPs). HSPs with sufficiently small probability of belonging by chance to the underlying distribution are kept and used as anchors for a bounded Needleman-Wunsch (NW in advance) global alignment [16]. Figure 1 shows the overall architecture of IMSAME. The following sections illustrate each of the methods employed in IMSAME.

The computation of hits, extended fragments (HSPs) and gapped alignments will often represent more than 85% of the computation time. Therefore the parallel strategy in IMSAME is focused in these stages, whereas loading the database and workload generation and distribution is performed sequentially.

2.1 Computation of alignments

This section depicts the internal procedure followed by IMSAME to compute pairwise alignments between the sequences contained within the inputs.

Hash table generation and diagonal filtering A hash table is built for words of size 12 (i.e. 12-mers) for the reference metagenome. This procedure starts by linearly scanning the reference metagenome and adding an entry in the hash table for each 12-mer. The position in the file and the read number to which it belongs is stored. Each entry in the hash table will hold a linked list in order to handle collisions. Since the reference metagenome is considered as a large sequence (i.e. the coordinates of reads is global in respect to the file), then the insertion of 12-mers in the hash-table is sorted in terms of the diagonal (as in [17]) between query and reference metagenome.

Hits detection and extension of fragments Once the hash table is built for the reference metagenome, the algorithm proceeds by loading the query metagenome and matching 12-mer words to those stored in the hash table. These hits serve as seeds to extend the alignments. For every hit, a linear, ungapped extension which allows mutations -but not indels- is performed in both directions, forward and backward respective to the sequence. This extension works by optimizing a scoring function that takes into account the length and number of shared identities.

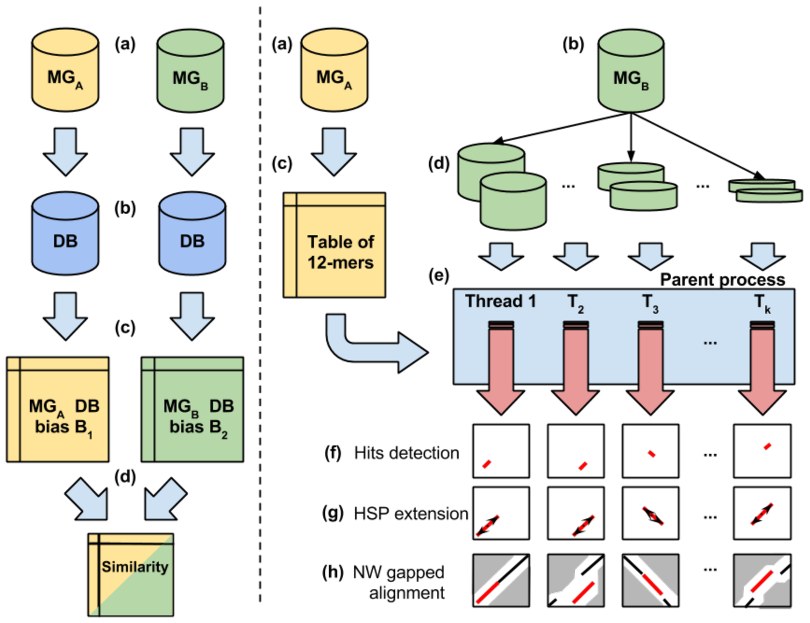

Fig. 1. Traditional metagenomics comparison (left) and overall diagram of the working procedure of IMSAME (right). Left: (a) Both metagenomes are compared individually against a chosen reference (b) database. (c) Individual results are merged into a unified (d) result which propagates individual biases. Right: (a) and (b) represent the input metagenomes. (c) Computation of the hash table of initial seeds. (d) Workload distribution to threads. (e) Each thread asks for a job (i.e. block of reads). (f) Each thread detects hits between reads. (g) After detection of hits, an HSP is computed by extending the hit linearly. (h) If the computed HSP has not been generated by chance, then a gapped alignment between the two reads is performed.

Anchored gapped alignment In order to apply a bounded computation for both CPU time and memory requirements, it is necessary to use heuristic methods to explore a reduced subspace of the whole search space. Hence, the quality of the results will be directly affected by the used heuristic method. IMSAME uses a simple yet powerful anchoring procedure, which is illustrated as follows: Once a hit has been detected and extended, the expected value of the resulting extended fragment is computed. If the expected value is sufficiently small, then it is used as anchor for the alignment. A straight line is computed between the global start (0,0) of the two sequences and the anchored fragment. Another line is computed from the ending of the anchored fragment to that of the two sequences. These two lines are then discretized using Bresenhams [18] algorithm. The Bresenhams algorithm will define a discrete succession of numbers that represent the guides for the alignment procedure. A window of variable size is used to explore a subset of cells in the NW matrix to the left and to the right respective to the center of the guides (and thus conforming a window). This procedure enables a much faster computation based on the reduced search space. At the same time, high quality alignments are still produced due to the anchored computation over regions that are known to be similar.

Bounded computation in CPU time and memory Since only a subset of the search space is explored, less memory is required to store the table computed in the dynamic programming algorithm. Therefore, the size of the bounded window will determine the reduction in memory and CPU time over the algorithm. Generally, a NW algorithm will require $\mathcal{O}(n^2)$ time and space in the size of the input. In this case, for sequences of length n and m, $\mathcal{O}(nm)$ will be required, which grows quadratically. Using a bounded window in the computation reduces one of the variables to a constant. Therefore if we take n as the size of one of the inputs and k as the size of the window, we will have that $k << n$. Hence, the space and time complexity drops from quadratic to linear in the size of the input. However, in certain cases the difference in length between the sequences to be aligned can be considerably large. For example, consider two sequences whose length difference is of one order of magnitude. If this difference is not taken into account, the algorithm will probably explore matrix cells that are outside the boundaries of the anchoring, and thus resulting in more computation time. In this sense, we propose using the geometric mean ($\sqrt{nm}$) to produce a mapping between sequences length ratio and window size. Moreover, k is further adjusted applying a user-defined parameter. Notice that the geometric mean will not assign the same window size to two different sets of sequences s_1 and s_2 of length $10^{l+1}, 10^{l+1}$ and s_3, s_4 of length $10^{l+2}, 10^{l}$, although the size of the search space would be equal.

2.2 Dynamic workload partitioning and distribution to threads

IMSAME uses a modified Guided Self Scheduling (GSS) [19] to handle workload assignment. In this line, the query metagenome (the reference metagenome is only processed to generate the hash table) is separated into M partitions each of which will contain m_i subpartitions, with i ranging from 1 to M. Each of these subpartitions will hold a number of reads that will be dynamically assigned to threads using a thread-safe queue when these run out of work. Moreover, the number of partitions is user-defined, and can be adjusted depending on the size of the inputs. Each subpartition is likewise divided by the number of threads t into blocks of reads. Thus, the number of reads contained in each block is determined by the current level of partitioning i, the total number of reads R and the number of threads t. The expression to calculate the number of reads assigned to a block at partition i is as follows:

$$B_i = \frac{\frac{R}{M}}{t * i} = \frac{R}{M * t * i} \tag{1}$$

Where R is the total number of reads in the metagenome, M is the number of partitions, t is the number of available threads and i is the current partition.

That is, the metagenome is firstly divided by the number of partitions, and each of these is then divided by the number of threads multiplied by the subpartition depth. Figure 2 shows how the query metagenome is decomposed per partition.

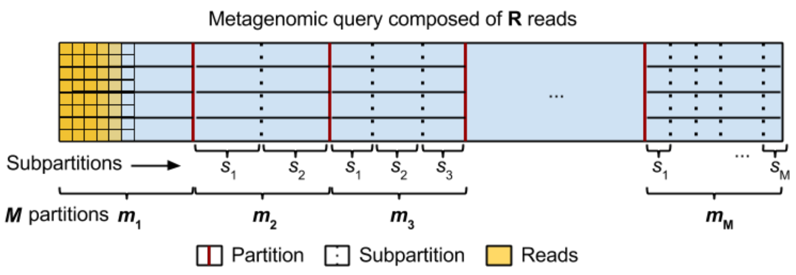

Fig. 2. Query decomposition into workload blocks. Initially, the query metagenome is divided into partitions (red solid lines). Each partition is again divided by the current partitioning level (vertical dashed lines). Finally, each subpartition is divided into blocks of equal number of reads depending of the number of threads available.

The workload distribution function belongs to the family of $1/x$ functions and shows a decay in the size of blocks in the early partitions in order to assign smaller jobs to the threads as these are consumed. Additionally, the function shows a horizontal asymptote that guarantees blocks of reads of a minimum size despite the number of partitions chosen.

3 Results and discussion

Two separate comparisons were carried out in order to test the two different aspects that have been improved over the original IMSAME version. All sequences used in the comparisons belong the to the Human Microbiome Project[1] (HMP), in particular to the Illumina WGS Assemblies. The run identifiers are provided at each comparison performed in order to allow reproducibility. The testing scenario is set up as follows:

1. A single comparison involving scaffolds to test sequential with optimizations.
2. A comparison involving three different datasets varying in sizes, from small to large. These comparisons will serve to account for the scalability of the parallel improvements in respect to the sequential version.

At last, the speedup from both optimization perspectives are discussed and addressed.

[1] `http://hmpdacc.org/HMASM/`

3.1 Infrastructure

The Picasso supercomputer located at the University of Malaga (Malaga, Spain) [20] was used to test the parallelization strategies. The computation was performed using only the fat nodes which contain 8 Intel E7-4870 processors and 2 TB of RAM each. The storage is managed by a Lustre file system supported by a DDN storage rack with five three-dimensional disk enclosures and two redundant SFA10000 controllers. The executions described in this manuscript range from 1 to 32 cores increasing by steps of powers of two. Runtime executions were measured using the *time* command from UNIX-based environments.

3.2 Comparison between the original and bounded IMSAME

Besides the improved parallelization strategies, IMSAME has additionally been improved with bounded CPU time and memory usage. In order to perform comparisons to measure the speedup produced by the parallelization techniques, the improvements are tested between the original IMSAME and the bounded in a sequential fashion. For this purpose, two runs composed of scaffolds, namely SRS016105 and SRS017451 were taken from the HMP database and compared using both versions. Since scaffolds are longer than reads, higher penalties were used for affine gap model (-8 for insertion of gap and -4 for extension of gap). Results remained equal for both executions, although variations can be observed if large-scale rearrangements take place (e.g. long range transpositions). The original version of IMSAME took 7 minutes and 29 seconds, whereas the bounded version took 55 seconds, representing a speedup of approximately 8x. This speedup is mostly produced by two facts (1) the diagonal filtering procedure reducing the number of linear extensions performed prior to a gapped alignment and (2) the bounded window applied to the NW algorithm, which substantially reduces the space search. However, it is important to note that the latter speedup is directly proportional to the size of the reads (i.e. smaller reads, smaller speedup).

3.3 Speedup evaluation of the parallelization strategy

The speedup introduced by the parallelization strategy is measured by using three datasets: (1) a small-sized one, (2) a medium-sized one and (3) a large-sized one. Table 1 summarizes the three datasets. This procedure enables us to evaluate the stability of the speedup as a function of the input size.

Table 2 shows the execution times for each of the datasets along with the speedup and efficiency of each execution. In the same line, Figure 4 shows the speedup evaluation plot. The speedup is calculated as the time needed by the algorithm run using only one core divided by the time needed using more cores. The efficiency is calculated as the ratio between the achieved speedup and the optimal speedup (equal to the number of cores). As can be seen in Table 2 and Figure 4, the speedup is nearly optimal in the scenario of enough data, i.e. the large dataset. However, in the case of the small and mid-sized dataset, a decay

Table 1. Summary of the dataset used for the speedup evaluation. From left to right: (1) Metagenome pairs compared, (2) Sum of reads from both metagenomes, (3) Sum of the size of both metagenomes in megabytes and (4) Average size of reads in base pairs.

Dataset (Run ID)	Number of reads	Size (MB)	Average read length (bp)
SRS017697 SRS019119	613,983	77	91
SRS064376 SRS065347	3,401,514	463	100
SRS018359 SRS057022	14,295,910	1809	99

in efficiency can be observed due to the size of data not being large enough for the number of cores. Moreover:

1. Small dataset (in purple in Figure 4): The peak of efficiency is achieved using 2 and 4 cores (reaching 84% efficiency). When using more than 4 cores, the dataset size becomes too small and some threads become inactive while others are still processing. To improve efficiency on the small dataset, a higher number of partitions should be used to remove thread balance synchronization at the end of the computation.
2. Medium dataset (in red in Figure 4): The medium dataset represents a 143% increase in size over the small dataset, and shows a much higher efficiency, with a peak at 86% using 8 cores. However, similarly to the smaller case, the efficiency (and thus the speedup) decays when using 32 cores.
3. Large dataset (in green in Figure 4): The larger dataset shows the overall best efficiency and speedup, with a peak at 8 cores and slow decay up to 32 cores, where 81% efficiency is achieved. However, the fact that the speedup is optimal at 2 and 4 cores indicates that probably still more partitioning levels are required in order to avoid thread synchronization.

Table 2. Execution times, speedup and efficiency for the executions of IMSAME using from 1 to 32 cores. The rows indicate the number of cores whereas the columns refer to time consumption (in seconds), speedup and efficiency per each of the datasets.

	Small			Medium			Large		
Cores	Time (s)	Speedup	Efficiency	Time (s)	Speedup	Efficiency	Time (s)	Speedup	Efficiency
1	425	1.00	**1.00**	8,634	1.00	**1.00**	76,253	1.00	**1.00**
2	252	1.69	**0.84**	4,770	1.81	**0.91**	38,009	2.01	**1.00**
4	126	3.37	**0.84**	2,571	3.36	**0.84**	18,867	4.04	**1.00**
8	76	5.59	0.70	1,262	6.84	**0.86**	9,838	7.75	**0.97**
16	42	10.12	0.63	683	12.64	0.79	5,282	14.44	**0.90**
32	30	14.17	0.44	388	22.25	0.70	2,937	25.96	**0.81**

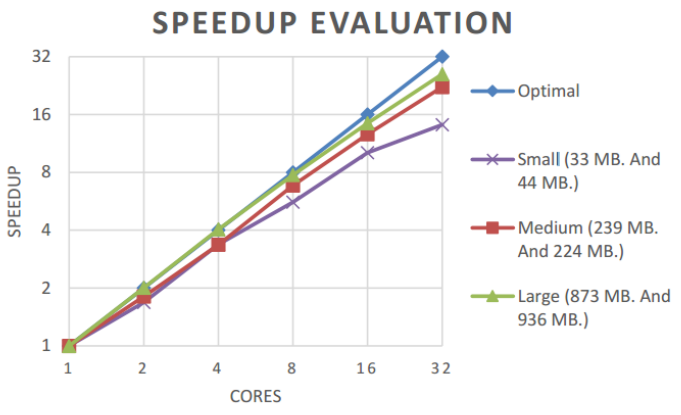

Fig. 3. The speedup is shown for the different datasets (in purple, green and red) along with the optimal speedup (in blue). The x-axis shows the number of cores used per comparison, whereas the y-axis shows the calculated speedup in respect to the number of cores used.

4 Conclusions

In this manuscript, we have shown an optimized version of IMSAME in which we applied two parallelization strategies, namely (1) a dynamic scheduler for the distribution of work and (2) an n-level parallelization in the computation of alignments using POSIX threads in a shared-memory environment. We have also applied several sequential improvements over the original version, which have improved the overall algorithm complexity and efficiency. Additionally, we have carried out two separate comparisons to prove the performance of IMSAME, that is, firstly, one to validate the sequential improvements over the original version and secondly, another one using three different datasets ranging in sizes to evaluate the achieved speedup and the parallel efficiency.

In order to keep developing IMSAME, we are currently working on:

1. Parallelization of the loading stage.
2. Improve workload distribution by building a regression model to automatically set the number of partitioning levels.
3. Use ROC curves [21] to set the optimal percentage thresholds.
4. Use genetic algorithms to determine optimal scheduling.

Acknowledgments

This work has been partially supported by the European project ELIXIR- EXCELERATE (grant no. 676559), the Spanish national projects Plataforma de Recursos Biomoleculares y Bioinformticos (ISCIII-PT13.0001.0012) and RIRAAF (ISCIII-RD12/0013/0006) and the University of Malaga.

References

1. Alyass, A., Turcotte, M., Meyre, D. (2015). From big data analysis to personalized medicine for all: challenges and opportunities. BMC medical genomics, 8(1), 33.
2. Altschul, S. F., Madden, T. L., Schffer, A. A., Zhang, J., Zhang, Z., Miller, W., Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic acids research, 25(17), 3389-3402.
3. Darling, A., Carey, L., Feng, W. C. (2003). The design, implementation, and evaluation of mpiBLAST. proceedings of ClusterWorld, 2003, 13-15.
4. `http://www.timelogic.com/catalog/757`. Last accessed 9/5/2017.
5. Liu, Y., Schmidt, B. (2015). GSWABE: faster GPUaccelerated sequence alignment with optimal alignment retrieval for short DNA sequences. Concurrency and Computation: Practice and Experience, 27(4), 958-972.
6. Liu, Y., Schmidt, B. (2014). CUSHAW2-GPU: empowering faster gapped short-read alignment using GPU computing. IEEE Design Test, 31(1), 31-39.
7. Liu, Y., Tran, T. T., Lauenroth, F., Schmidt, B. (2014, September). SWAPHI-LS: Smith-Waterman algorithm on Xeon Phi coprocessors for long DNA sequences. In Cluster Computing (CLUSTER), 2014 IEEE IC (pp. 257-265).
8. Langmead, B. (2010). Aligning short sequencing reads with Bowtie. Current protocols in bioinformatics, 11-7.
9. Li, H., Durbin, R. (2009). Fast and accurate short read alignment with BurrowsWheeler transform. Bioinformatics, 25(14), 1754-1760.
10. Jing, G., Sun, Z., Wang, H., Gong, Y., Huang, S., Ning, K., ... Su, X. (2017). Parallel-META 3: Comprehensive taxonomical and functional analysis platform for efficient comparison of microbial communities. Scientific reports, 7, 40371.
11. Nichols, B., Buttlar, D., Farrell, J. (1996). Pthreads programming: A POSIX standard for better multiprocessing. " O'Reilly Media, Inc.".
12. Perez-Wohlfeil, E., Torreno, O., Trelles, O. (2017, April). Pairwise and incremental multi-stage alignment of metagenomes: A new proposal. In International Conference on Bioinformatics and Biomedical Engineering (pp. 74-80).
13. Ondov, B. D., Treangen, T. J., Melsted, P., Mallonee, A. B., Bergman, N. H., Koren, S., Phillippy, A. M. (2016). Mash: fast genome and metagenome distance estimation using MinHash. Genome Biology, 17(1), 132.
14. Benoit, G., Peterlongo, P., Mariadassou, M., Drezen, E., Schbath, S., Lavenier, D., Lemaitre, C. (2016). Multiple comparative metagenomics using multiset k-mer counting. PeerJ Computer Science, 2, e94.
15. Maillet, N., Collet, G., Vannier, T., Lavenier, D., Peterlongo, P. (2014, November). COMMET: comparing and combining multiple metagenomic datasets. In Bioinformatics and Biomedicine (BIBM), 2014 IEEE IC (pp. 94-98).
16. Gotoh, O. (1982). An improved algorithm for matching biological sequences. Journal of molecular biology, 162(3), 705-708.
17. Torreno, O., Trelles, O. (2015). Breaking the computational barriers of pairwise genome comparison. BMC bioinformatics, 16(1), 250.
18. Pitteway, M. L. V., Watkinson, D. J. (1980). Bresenham's algorithm with Grey scale. Communications of the ACM, 23(11), 625-626.
19. Polychronopoulos, C. D., Kuck, D. J. (1987). Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. IEEE transactions on computers, 100(12), 1425-1439.
20. `http://www.scbi.uma.es/site/scbi/hardware`. Last accessed 9/5/2017.
21. Hanley, J. A., McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology, 143(1), 29-36.