

SwiftOrtho: a Fast, Memory-Efficient, Multiple Genome Orthology Classifier

Xiao Hu
and Iddo Friedberg*

*Correspondence:

idoerg@iastate.edu

Department of Veterinary
Microbiology and Preventive
Medicine, College of
Veterinary Medicine, Iowa
State University, 2118 Vet
Med, Ames, IA 50011, USA
Full list of author information
is available at the end of the
article

Abstract

Introduction: Gene homology type classification is a requisite for many types of genome analyses, including comparative genomics, phylogenetics, and protein function annotation. A large variety of tools have been developed to perform homology classification across genomes of different species. However, when applied to large genomic datasets, these tools require high memory and CPU usage, typically available only in costly computational clusters. To address this problem, we developed a new graph-based orthology analysis tool, SwiftOrtho, which is optimized for speed and memory usage when applied to large-scale data.

Results: In our tests, SwiftOrtho is the only tool that completed orthology analysis of 1,760 bacterial genomes on a computer with only 4GB RAM. Using various standard orthology datasets, we also show that SwiftOrtho has a high accuracy. SwiftOrtho enables the accurate comparative genomic analyses of thousands of genomes using low memory computers.

Availability: <https://github.com/Rinoahu/SwiftOrtho>

1 Background

2 Gene homology type classification consists of identifying paralogs and orthologs
3 across species. Orthologs are genes that evolved from a common ancestral gene fol-
4 lowing speciation, while paralogs are genes that are homologous due to duplication.
5 Computationally detecting orthologs and paralogs across species is an important
6 problem, as the evolutionary history of genes has implications for our understand-
7 ing of gene function and evolution.

8 While the proper inference of homology type involves tracing gene history using
9 phylogenetic trees [1], several proxy methods have been developed over the years.
10 The most common method to infer orthologs by proxy is Reciprocal Best Hit or
11 RBH [2, 3]. Briefly, RBH states the following: when two proteins that are encoded
12 by two genes, each in a different genome, find each other as the best scoring match,
13 they are considered to be orthologs [2, 3].

14 Inparanoid extends the RBH orthology relationship to include both orthologs and
15 in-paralogs [4–6]. Specifically, Inparanoid distinguishes between orthologs and in-
16 paralogs, which were duplicated following a given speciation event [4–6]. It is then
17 a matter of course to extend orthologous pairs between two species to an ortholog
18 group, where an ortholog group is defined as a set of genes that are hypothesized to
19 have descended from a common ancestor [6]. Several methods have been developed
20 to identify ortholog groups across multiple species. These methods can be classi-
21 fied into two types: tree-based and graph-based. Tree-based methods construct a
22 gene tree from an alignment of homologous sequences in different species and infer
23 orthology relationships by reconciling the gene tree with its corresponding species
24 tree [1, 7, 8]. Tree-based methods can infer a correct orthology relationship if the
25 correct gene tree and species tree are given [9]. The main limitation of tree-based
26 methods is the accuracy of the given gene tree and species tree. Erroneous trees
27 lead to incorrect ortholog and in-paralog assignments [8–10]. Tree-based methods
28 are also computationally expensive which limits the ability to apply them to large
29 number of species [9, 11–13]. Graph-based methods infer orthologs and in-paralogs
30 (Figure 1) from homologs and then use different strategies to cluster them
31 into orthologous groups [8, 11, 12]. The Clusters of Orthologous Groups or COG
32 database detects triangles of RBHs in three different species and merges triangles
33 with a common side [14]. Orthologous Matrix (OMA) clusters RBHs to orthologous
34 groups by finding maximum weight cliques from the similarity graph [15]. Multi-
35 Paranoid is an extension of Inparanoid, which uses InParanoid to detect triangle or-
36 thologs and in-paralogs in three different species as seeds and then merges the seeds
37 into larger groups [16]. OrthoMCL also uses the InParanoid algorithm to detect or-
38 thologs, co-orthologs, and in-paralogs between two species [17] and then uses Markov
39 Clustering (MCL) [18] to cluster these relationships into orthologous groups. The-
40 oretically, graph-based methods are less accurate than tree-based methods, as the

41 former identify orthologs and in-paralogs using proxy methods rather than directly
42 inferring homology type from gene and species evolutionary history. In practice,
43 graph-based methods have a similar accuracy as tree-based methods [9, 10, 19]. A
44 comparison of several methods that include both tree-based and graph-based meth-
45 ods found that tree-based methods had even a worse performance than graph-based
46 methods on large dataset [10]. One study compared several common methods in-
47 cluding RBH, graph-based and tree-based and found that tree-based methods often
48 give a higher specificity but lower sensitivity [20]. Several studies have also shown
49 that graph-based methods find a better trade-off between specificity and sensitiv-
50 ity than tree-based methods [10, 20, 21]. Due to their better speed and accuracy,
51 graph-based methods are generally preferred for analyzing large data set.

52 Graph-based methods such as OrthoMCL and InParanoid can analyze hundreds
53 of genomes, however they require considerable computational resources that may
54 not be readily available [22, 23].

55 Here we developed a new orthology analysis tool named SwiftOrtho. SwiftOrtho
56 is a graph-based method focused on speed, accuracy and memory efficiency. We
57 compared SwiftOrtho with several existing graph-based tools using the gold stan-
58 dard dataset Orthobench [12], and the Quest for Orthologs service [24]. Using both
59 benchmarks, we show that SwiftOrtho provides a high accuracy with lower CPU
60 and memory usage than other graph-based methods.

61 **Methods**

62 **Algorithms**

63 SwiftOrtho is a graph-based orthology prediction method that performs homology
64 search, orthology inference, and clustering by homology type.

65 *Homology Search*

66 SwiftOrtho employs a seed-and-extension algorithm to find homologous gene
67 pairs [25, 26]. At the seed phase, SwiftOrtho finds candidate target sequences that

68 share common k -mers with the query sequence. k -mer size is an important fac-
69 tor that affects search sensitivity and speed [27, 28]. SwiftOrtho therefore uses
70 long (≥ 6) k -mers to accelerate search speed. However, k -mer length is negatively
71 correlated with sensitivity [27]. To compensate for the loss of sensitivity caused
72 by increasing k -mer size, SwiftOrtho uses two approaches: non-consecutive k -mers
73 and reduced amino-acid alphabets. Non-consecutive k -mer seeds (known as spaced
74 seeds), were introduced in PatternHunter [17, 29]. The main difference between con-
75 secutive seeds and spaced seeds is that the latter allow mismatches in alignment. For
76 example, the spaced seed 101101 allows mismatches at positions 2 and 5. The total
77 number of matched positions in a spaced seed is known as a weight, so the weight of
78 this seed is 4. A consecutive seed can be considered as a special case of spaced seed
79 in which its weight equal its length. Spaced seeds often provide a better sensitivity
80 than consecutive seeds [29, 30]. The default spaced seed patterns of SwiftOrtho are
81 1110100010001011, 11010110111 –two spaced seeds with weight of 8– but the user
82 may define their own spaced seeds. Seed patterns were optimized using SpEED [30]
83 and manual inspection. The choice of the spaced seeds and default alphabet are
84 elaborated upon in the Methods section in the Supplementary Materials. At the ex-
85 tension phase, SwiftOrtho uses a variation of the Smith-Waterman algorithm [31],
86 the k -banded Smith-Waterman or k -SWAT, which only allows for k gaps [32]. k -
87 SWAT fills a band of cells along the main diagonal of the similarity score matrix
88 (Figure 2B), and the complexity of k -swat is reduced to $O(k \cdot \min(n, m))$, where k
89 is the maximum allowed number of gaps.

90 Another method to mitigate the loss of sensitivity is to use reduced amino acid
91 alphabets. Reduced alphabets are used to represent protein sequences using an
92 alternative alphabet that combines several amino acids into a single representa-
93 tive letter, based on common physico-chemical traits [33–35]. Compared with the
94 original alphabet of 20 amino acids, reduced alphabets usually improve sensitiv-

ity [36, 37]. However, reduced alphabets also introduces less specific seeds than the original alphabet, which reduces the search speed.

Orthology Inference

SwiftOrtho employs a graph-based approach as the method to infer orthologs, co-orthologs and in-paralogs from homologs (Figure 1), and uses RBH to identify the orthologs. If the bit score between gene A_1 and A_2 in genome A is higher than that between A_1 and all its orthologs in other genomes, A_1 and A_2 are considered in-paralogs in genome A. If A_1 in genome A and B_1 in genome B are orthologs, in-paralogs of A_1 and B_1 are co-orthologs (Figure 1). This process requires many queries so it is therefore better to store the data in a way that facilitates fast querying. SwiftOrtho sorts the data and uses a binary search algorithm to query the sorted data, which significantly reduces memory usage when compared with an Relational Database Management System or a hash table. With the help of this query system, SwiftOrtho can process data that are much larger than the computer memory.

After inferring orthology, the inferred orthology relationships are treated as the edges of a graph. Each edge is assigned a weight for cluster analysis. Appropriate edge-weighting metrics can improve the accuracy of cluster analysis. Gibbons compared the performance of several BLAST-based edge-weighting metrics and uses the bit score [38]. SwiftOrtho also uses the normalized bit score as edge-weighting metric. The normalization step take the same approach as OrthoMCL [22]: For orthologs or co-orthologs, the weight of (co-)ortholog (Figure 1) A_1 in genome A and B_1 in genome B is divided by the average edge-weight of all the (co-)orthologs between genome A and genome B. For in-paralogs, SwiftOrtho identifies a subset S of all in-paralogs in genome A, with each in-paralog A_x - A_y in subset S, A_x or A_y having at least one ortholog in another genome. The weight of each in-paralog in genome A is divided by the average edge-weight of subset S in genome A [22].

122 *Clustering Orthology Relationships into Orthologous Groups*

123 SwiftOrtho provides two methods to cluster orthology relationships in orthologous
124 groups. One is the Markov Cluster algorithm (MCL), an unsupervised clustering
125 algorithm based on simulation of flow in graphs [18]. MCL is fast and robust on
126 small networks and has been used by several graph-based tools [17, 39–41]. However,
127 MCL may run out of memory when applied on a large-scale network. To reduce
128 memory usage, we cluster each individual connected component instead of the whole
129 network because there is no flow among components [18]. However, for large and
130 dense networks a single connected component could still be too large to be loaded
131 into memory.

For the large networks, SwiftOrtho uses an Affinity Propagation Clustering algorithm (APC)[42]. The APC algorithm finds a set of centers in a network, where the centers are the actual data points and are called “exemplars”. To find exemplars, APC needs to keep two matrices of the responsibility matrix R and the availability matrix A . The element $R_{i,k}$ in R reflects how well-suited node k is to serve as the exemplar for node i while the element $A_{i,k}$ in A reflects how appropriate node i to choose node k as its exemplar [42]. APC uses Equation 1 to update R , and Equation 2 to update A , where i, k, i', k' denote the node number, and $S_{i,k'}$ denotes the similarity between node i and node k' .

$$R_{i,k} = S_{i,k} - \max_{k' \neq k} \{A_{i,k'} + S_{i,k'}\} \quad (1)$$

$$A_{i,k} = \begin{cases} \min\{0, R_{k,k} + \sum_{i' \notin \{i,k\}} \max\{0, R_{i',k}\}, & \text{if } i \neq k \\ \sum_{i' \neq k} \max\{0, R_{i',k}\}, & \text{if } i = k \end{cases} \quad (2)$$

132 The node k that maximizes $A_{i,k} + R_{i,k}$ is the exemplar of node i , and each node
133 i is assigned to its nearest exemplar. APC can update each element of matrix R
134 and A one by one, so, it is unnecessary to keep the whole matrix of R and A in
135 memory. Generally, the time complexity of APC is $O(N^2 \cdot T)$ where N is number
136 of nodes and T is number of iterations [42]. In this case, the time complexity is
137 $O(E \cdot T)$, where E stands for edges which is number of orthology relationships and
138 T is number of iterations. We implemented APC in Python, using Numba [43] to
139 accelerate the numeric-intensive calculation parts.

140 Application to Real data

141 Data Sets

142 We applied SwiftOrtho to three data sets to evaluate its predictive quality and
143 performance:

- 144 1 The *Euk* set was used to evaluate the quality of predicted orthologous groups.
145 This set contains 420,415 protein sequences from 12 eukaryotic species, in-
146 cluding *Caenorhabditis elegans*, *Drosophila melanogaster*, *Ciona intestinalis*,
147 *Danio rerio*, *Tetraodon nigroviridis*, *Gallus gallus*, *Monodelphis domestica*,
148 *Mus musculus*, *Rattus norvegicus*, *Canis familiaris*, *Pan troglodytes* and *Homo*
149 *sapiens*. The protein sequences for these genes were downloaded from EMBL
150 v65 [44].
- 151 2 The *QfO 2011* set was used to evaluate the quality of predicted orthology
152 relationships. This set was the reference proteome dataset (2011) of The Quest
153 for Orthologs[24], which contains 754,149 protein sequences of 66 species.
- 154 3 The large *Bac* set was used to evaluate performance, including CPU time, real
155 time and RAM usage. This set includes 5,950,817 protein sequences from 1,760
156 bacterial species. The protein sequences were downloaded from GenBank [45].
157 For a full list, see the additional file 1.

158 Comparing SwiftOrtho with existing Tools

159 We compared SwiftOrtho with several existing orthology analysis tools for pre-
160 dictive quality and performance. The methods compared were: OrthoMCL(v2.0),
161 FastOrtho, OrthAogue, and OrthoFinder.

162 Orthology Analysis Pipeline

163 The pipeline for all the tools follows the standard steps of graph-based orthology
164 prediction, (1) all-*vs*-all homology search, (2) orthology inference, and (3) cluster
165 analysis.

166 *Homology Search*

167 SwiftOrtho used its built-in module to perform all-*vs*-all homology search. For all
168 the three sets, the E-value was set 10^{-5} . The amino acid alphabet was set to the
169 regular 20 amino acids for the three sets. The spaced seed parameter was set to
170 1011111,11111 for the Euk, 11111111 for the *QfO 2011*, and 111111 for *Bac*.

171 OrthoMCL, FastOrtho, OrthAogue, and OrthoFinder used BLASTP (v2.2.26)
172 to perform all-*vs*-all homology search. The first three tools require the user to do this
173 manually. In order to be able to compare, the -e (e-value), -v (number of database
174 sequences to show one-line descriptions), and -b (number of database sequence to
175 show alignments) parameters of BLASTP were set to 10^{-5} , 1,000,000, and, 1,000,000
176 for OrthoMCL, FastOrtho, and OrthAogue. The OrthoFinder calls BLASTP, and
177 the E-value of BLASTP have been set to 10^{-3} .

178 *Orthology Inference*

179 SwiftOrtho, OrthoMCL, FastOrtho, OrthAogue, and OrthoFinder were applied to
180 perform orthology inference on the homologs. The first four tools are able to identify
181 (co-)orthologs and in-paralogs, and the coverage (fraction of aligned regions) was set
182 to 50%, while other parameters were set to their default values, see Supplementary
183 Materials for full details. FastOrtho does not report (co-)orthologs and in-paralogs

184 directly. However, the relevant information is stored in an intermediate file, from
185 which we have extracted that information. Orthofinder does not report orthology
186 relationships.

187 *Cluster Analysis*

188 All the tools in this study use MCL [18] for clustering. To control the granularity of
189 the clustering, MCL performs an inflation operation controlled by *-I* option [18, 46].
190 In this study, *-I* was set to 1.5. To take advantage of multiprocessor capabilities,
191 we set the thread number of MCL to 12. SwiftOrtho has an alternative clustering
192 algorithm APC, which we have also applied to *Euk* and *Bac*.

193 Evaluation of Prediction Quality

194 *Evaluation of Predicted Orthologous Group*

195 The OrthoBench set was used to evaluate the quality of predicted orthologous
196 groups in *Bac*. This set contains 70 manually curated orthologous groups of the 12
197 species from *Bac* and has been used as a high quality gold standard benchmark
198 set for orthologous group prediction [12], we used OrthoBench v2 (Supplementary
199 Table S1). In this study, each manually curated group of OrthoBench v2 set finds
200 the best match in the predicted orthologous groups, where the best match means
201 that the number of genes shared between manually curated and predicted orthologs
202 is maximized, and the precision and recall are calculated (Figure 3A.).

203 *Evaluation of Predicted Orthology Relationships*

204 The *Quest of Orthologs* web-based service (QfO) was employed to evaluate the qual-
205 ity of the orthology relationships predicted from the *QfO 2011* set [24]. QfO service
206 evaluates the predictive quality by performing four phylogeny-based tests of *Species*
207 *Tree Discordance Benchmark*, *Generalized Species Tree Discordance Benchmark*,
208 *Agreement with Reference Gene Phylogenies: SwissTree*, and *Agreement with Refer-*
209 *ence Gene Phylogenies: TreeFam-A*, and two function-based tests of *Gene Ontology*

210 *conservation test* and *Enzyme Classification conservation test* [24]. We also applied
211 two more orthology prediction tools, SonicParanoid[47] and InParanoid (v4.1)[4],
212 on the *QfO 2011* set and used their results as control. The pairwise orthology rela-
213 tionships were extracted from the predicted orthologous groups of all the tools, in-
214 cluding SonicParanoid and InParanoid, and then submitted to the QfO web-service
215 for further evaluation.

216 Hardware

217 Unless specified otherwise, all tests were run on the Condo cluster of Iowa State
218 University with Intel Xeon E5-2640 v3 at 2.60GHz, 128GB RAM, 28TB free disk.
219 The Linux command `/usr/bin/time -v` was used to track CPU and peak memory
220 usage.

221 Results

222 We compared the orthology analysis performance of SwiftOrtho, OrthoMCL, Fas-
223 tOrtho, OrthAogue, and OrthFinder using *Euk*, *QfO 2011*, and *Bac*. The orthology
224 analysis consists of homology search, orthology inference, and cluster analysis.

225 Orthology Analysis on *Euk*

The results of orthology analysis on *Euk* are summarized in Table 1:

		SwiftOrtho	OrthoMCL	FastOrtho	OrthAogue	OrthFinder
Homology Search	Method	SO built-in	BLASTP			
	Hits	162,620,048	947,203,546			654,792,861
	Uniq Hits	162,620,048	297,107,872			266,104,611
<hr/>						
Orthology Inference	(Co-)orthologs	1,199,783	8,279,424	3,297,613	1,265,553	N/A
	In-paralogs	557,593	2,517,166	2,546,296	759,989	N/A
<hr/>						
Clustering	Algorithm	MCL	APC	MCL		
	Orthologous Groups	48,270	43,114	36,901	40,943	51,297

Table 1 Comparative orthology analysis on the *Euk* set. N/A: not available, SO: SwiftOrtho, MCL: Markov Clustering, APC: Affinity Propagation Cluster.

227 *Homology Search*

228 The homology search results show that BLASTP detected the largest number
229 of homologs (947,203,546). SwiftOrtho found 57.5% of the homologs detected by
230 BLASTP but was 38.7 times faster than BLASTP. SwiftOrtho used longer k -mers,
231 which reduced both specific and non-specific seed extension. The longer k -mers
232 cause seed-and-extension methods to ignore low similarity sequences. According to
233 the RBH rule, orthologs should have higher similarity than non-orthologs, so, the
234 decrease in homologs of SwiftOrtho does not significantly affect the next orthology
235 inference. We compared RBHs inferred from homologs detected by BLASTP and
236 SwiftOrtho, and the numbers of RBHs for BLASTP and SwiftOrtho are 654,730 and
237 645,091, respectively. Identical RBHs are 497,286 (76.0% of BLASTP). These results
238 shows that although SwiftOrtho found fewer homologs than BLASTP, SwiftOrtho
239 does not significantly reduce the number of RBHs. The following results in Figure 4
240 also show that there is no significant difference between SwiftOrtho and BLASTP
241 in orthologous groups prediction.

242 *Orthology Inference*

243 OrthoMCL and FastOrtho found more orthology relationships than SwiftOrtho and
244 OrthAogue. This is because OrthoMCL and FastOrtho use the negative log ratio
245 of the e-value as the edge-weighting metric. The BLASTP program rounds E-value
246 $< 10^{-180}$ to 0. Consequently, for homologs with an e-value $< 10^{-180}$, OrthoMCL
247 and FastOrtho treat them as the RBHs, overestimating the number of orthologs.
248 An example showing the OrthoMCL and FastOrtho overestimation can be found in
249 Table S4.

250 **Computational resource use:** OrthoMCL v2.0 used the most CPU time and
251 real time because of the required I/O operations. The RAM usage of OrthoMCL
252 was 3.45GB, at the same time, the generated intermediate file occupied >19 TB
253 disk space. OrthAogue was the most real time efficient because its ability to ex-

254 exploit a multi-core processor. However, the RAM usage of OrthAgoque was more
255 than 100GB which exceeds most workstations and servers. The orthology inference
256 module of FastOrtho was the most memory-efficient among all the tools and it is
257 also fast. SwiftOrtho was the most CPU time efficient although its real time was
258 twice as OrthAgoque. Because the orthology inference module of SwiftOrtho was
259 written in pure Python, we retested it by using the PyPy interpreter, an alter-
260 nate implementation of Python [48]. The results show that the real run time of
261 SwiftOrtho was close to OrthAgoque's (Table S5)

262 *Cluster Analysis*

263 OrthoFinder identified the smallest number of orthologous groups. Other tools iden-
264 tified many more orthologous groups than OrthoFinder, ranging from 36,901 to
265 51,297. The APC algorithm find fewer clusters than the MCL algorithm.

266 *Evaluation of Predicted Orthologous Groups*

267 The quality of predicted orthologous groups is shown in Figure 3. OrthoFinder
268 has the best recall, while SwiftOrtho and OrthAgoque have top precision values
269 but lower recall values than other tools. Since SwiftOrtho and OrthAgoque use a
270 more stringent standard to perform orthology inference, this strategy often increases
271 precision but decreases recall [10, 20, 21].

272 Because SwiftOrtho uses its built-in homology search module and its recall is
273 lower than BLASTP's, this may also cause a reduction in the recall of orthol-
274 ogous groups. To eliminate this possibility, we made two replacements. We re-
275 placed SwiftOrtho's homology module with BLASTP for SwiftOrtho and replaced
276 BLASTP with SwiftOrtho's homology module for OrthoMCL, FastOrtho, OrthA-
277 goque, and OrthoFinder. We then reran the orthology analysis on *Euk*. The results
278 show that for most tools replacing BLASTP with SwiftOrtho's built-in homology
279 search module does not significantly reduce the recall (Figure 4). The difference in
280 recall between using SwiftOrtho's homology search and using BLASTP is less than

281 3% except for OrthoMCL and FastOrtho. The recall for OrthoMCL and FastOrtho
 282 decreased by 5% and 8%, respectively. The most likely reason is that the E-value
 283 of SwiftOrtho’s homology search module is more precise than that of BLASTP,
 284 which reduces the false RBHs as mentioned above. These results also show that
 285 SwiftOrtho’s homology search module is a reliable and fast alternative to BLASTP.

286 Since SwiftOrtho uses an APC clustering algorithm, we ran SwiftOrtho with MCL
 287 and APC on the same data. The results (Figure 5) show that performance of APC
 288 is very close to that of MCL. APC improves the recall of most tools (Figure 5).
 289 These results also show that APC is a reliable alternative to MCL. APC requires
 290 less memory and can be used to cluster large-scale data.

291 Orthology Analysis on *QfO 2011*

The results of the orthology analysis on *QfO 2011* are shown in Table 2:

		SwiftOrtho	OrthoMCL	FastOrtho	OrthoAgogue	OrthoFinder
Homology Search	Method	SO built-in	BLASTP			
	Hits	183,883,417	642,372,369			935,579,809
	Uniq Hits	183,883,417	317,333,885			462,876,579
Orthology Inference	(Co-)orthologs	2,209,243	3,743,779	2,588,851	2,716,128	N/A
	In-paralogs	6,929,058	11,427,118	13,649,582	13,694,208	N/A
Clustering	Algorithm	MCL				
	Orthologous Groups	60,418	50,970	55,530	50,203	166,217

Table 2 Comparative orthology analysis on the Quest for Orthologs reference proteome 2011 dataset. SO: SwiftOrtho; MCL: Markov Clustering; APC: Affinity Propagation Cluster; N/A: not available.

292

293 *Homology Search*

294 SwiftOrtho found 183,883,417 unique hits while BLASTP found 462,876,579 unique
 295 hits. However, SwiftOrtho is about 163 times faster than BLASTP.

296 *Orthology Inference*

297 OrthoMCL found many more orthologs and co-orthologs than the other tools.
 298 SwiftOrtho found fewer in-paralogs than other available tools. The CPU time of
 299 SwiftOrtho is the least of all tools. When using the PyPy interpreter, the real time

300 of SwiftOrtho is also close to that of the fastest one, OrthAogue (Supplementary
301 Table S6).

302 *Cluster Analysis*

303 Overall, the clustering numbers of SwiftOrtho, OrthoMCL, FastOrtho, and Orth-
304 Aogue are similar. However, the number of clusters found by OrthoFinder is three
305 times that of other tools, and the next evaluation also shows that OrthoFinder
306 performed poorly on *QfO 2011*.

307 *Evaluation of Predicted Ortholog Relationships*

308 The evaluation shows that the performance of SwiftOrtho is close to that of Inpara-
309 noid (Figure 6). In some tests (Figure 6, D-E), SwiftOrtho outperformed Inparanoid.
310 SwiftOrtho had the best performance in the Generalized Species Tree Discordance
311 Benchmark and Agreement with Reference Gene Phylogenies: TreeFam-A tests. In
312 the Species Tree Discordance Benchmark, SwiftOrtho had the minimum Robinson-
313 Foulds distance. In the Enzyme Classification (EC) conservation test, SwiftOrtho
314 had the maximum Schlicker similarity. These two metrics reflect the performance
315 of the algorithm in accuracy and the results show that SwiftOrtho has an overall
316 higher accuracy than the other tools, at the same time, the recall of SwiftOrtho
317 was lower in some of the QfO tests. The most probable reason is that when we
318 performed all-*vs*-all homology search, we used a long seed which resulted in fewer
319 homologs being detected.

320 *Orthology Analysis On Bac*

321 The results of orthology analysis on *Bac* are shown in Table 3:

322 *Homology Search*

323 SwiftOrtho detected 8,966,131,536 homologs on the *Bac* set within 1,247 CPU
324 hours. Because it takes long time to perform all-*vs*-all BLASTP search on the full
325 *Bac*, we randomly selected 1,000 protein sequences from *Bac* and searched them

		SwiftOrtho	OrthoMCL	FastOrtho	OrthAogue	OrthoFinder
Homology Search	Method	SO built-in				N/A
	Hits	8,478,732,753				N/A
	Uniq Hits	8,478,732,753				N/A
Orthology Inference	(Co-)orthologs	876,766,940	N/A	950,683,849	N/A	N/A
	In-paralogs	622,292	N/A	663,052	N/A	N/A
Clustering	Algorithm	MCL	APC	MCL		
	Orthologous Groups	240,162	167,355	N/A	242,816	N/A

Table 3 Comparative orthology analysis on the *Bac* set. **SO:** SwiftOrtho; **MCL:** Markov Clustering; **APC:** Affinity Propagation Cluster; **N/A:** not available.

326 against the full *Bac* set. It took BLASTP 5.1 CPU hours to find the homologs of
327 these 1,000 protein sequences. We infer that the estimated CPU time of BLASTP
328 on the full *Bac* set should be around 30,000 CPU hours. SwiftOrtho was almost 25
329 times faster than BLASTP on *Bac*.

330 *Orthology Inference*

331 SwiftOrtho, OrthoMCL, FastOrtho, and OrthAogue were used to infer (co-
332)orthologs and in-paralogs from the homologs detected by the homology search
333 module of SwiftOrtho in the *Bac* set. We did not test Orthofinder, because Or-
334 thofinder does not accept a single file of homologs as input. For the 1,760 genomes in
335 *Bac*, OrthoFinder needs to perform 3,097,600 pairwise genome comparisons, which
336 will generate the same number of files. Then, OrthoFinder performs the orthol-
337 ogy inference on these 3,097,600 files. Even at one minute per file, it will take an
338 estimated six CPU years to process all the files.

339 Due to memory limitation, only SwiftOrtho and FastOrtho finished the orthol-
340 ogy inference on *Bac*. The results are shown in Table 3. The numbers of (co-
341)orthologs and in-paralogs inferred by SwiftOrtho and FastOrtho are similar. The
342 number of common orthology relationships between SwiftOrtho and FastOrtho was
343 861,619,519 (98.2% of SwiftOrtho and 90.57% of FastOrtho). Compared with *Euk*,
344 SwiftOrtho and FastOrtho have a similar predictive quality on *Bac*. There are three
345 possible explanations for these results. The first one is that *Euk* contains many pro-
346 tein isoforms which cause FastOrtho to overestimate the number of orthologs and

347 in-paralogs. The second one is that the gene duplication rate in Bacteria is lower
348 than that in Eukaryotes [49, 50]. For *Bac*, each gene in one species has only small
349 number of homologs in other species, which makes FastOrtho unlikely to overesti-
350 mate the number of RBHs. The third one is that SwiftOrtho uses double-precision
351 floating-point to store the E-value, which increases the precision of E-value from
352 10^{-180} to 10^{-308} . This improvement also reduces the possibility that FastOrtho
353 may report false RBHs.

354 **Computational resource use:** FastOrtho and OrthAogue did not finish the
355 tests due to insufficient RAM, OrthoMCL aborted after running out of disk space, as
356 it needed more than 18TB. Only SwiftOrtho and FastOrtho finished the orthology
357 inference step. The Peak RAM usage of SwiftOrtho and FastOrtho were 90.6GB and
358 99.5GB, respectively. When we used the PyPy interpreter, the Peak RAM usage
359 of SwiftOrtho was reduced to 72.1GB. FastOrtho was about 1.52 times faster than
360 SwiftOrtho which ran the tests in the CPython interpreter. When using the PyPy
361 interpreter, SwiftOrtho ran 1.58 times faster than FastOrtho. The memory usage
362 and CPU time are shown in Supplementary Table S7

363 *Cluster Analysis*

364 The clustering numbers of SwiftOrtho and FastOrtho are similar. We compared the
365 APC algorithm and the MCL algorithm, and APC found fewer clusters than MCL.
366 The APC used much less memory and less CPU time than MCL. However, due to
367 the lack of support for multi-threading and a large number of I/O operations, the
368 real run time of APC is longer than that of MCL.

369 *Test on Low-memory System*

370 Because SwiftOrtho is designed to handle large-scale data on low-memory comput-
371 ers, we used it to analyze *Bac* on a range of computers with different specifications.
372 The results (Supplementary Table S8) show that the memory usage of SwiftOrtho is
373 flexible and adaptes to the size of the computer's memory. In the tests, SwiftOrtho

374 finished an orthology analysis of *Bac* on computers with only 4GB RAM in a rea-
375 sonable time (Table S8).

376 Discussion

377 We present SwiftOrtho, a new high performance graph based homology classifica-
378 tion tool. Unlike most tools that can only perform orthology inference, SwiftOrtho
379 integrates all the modules necessary for orthology analysis, including homology
380 search, orthology inference and cluster analysis. SwiftOrtho is designed to ana-
381 lyze large-scale genomic data on a normal desktop computer in a reasonable time.
382 In our tests, SwiftOrtho's homology search module was nearly 30 times faster than
383 BLASTP. The orthology inference module of SwiftOrtho was nearly 500 times faster
384 than OrthoMCL when applied to *Euk*. When applied to the large-scale dataset, *Bac*,
385 SwiftOrtho was the only one that finished orthology inference test on a workstation
386 with 32GB RAM. The cluster module of SwiftOrtho using APC can handle data
387 that is much larger than the computer memory. In our test, APC has comparable
388 recall and accuracy, but requires much less memory than MCL. APC even improved
389 F_1 -measure score by increasing recall in most cases. With the help of these opti-
390 mized modules, SwiftOrtho has successfully finished an orthology analysis of 1,760
391 bacterial genomes on a machine with only 4GB RAM. SwiftOrtho is not only fast
392 but also accurate, as showing the results produced when running on orthobench
393 and QfO[12, 24].

394 Conclusion

395 In summary, SwiftOrtho is a fast, accurate orthology prediction tool that can an-
396alyze a large number of sequences with minimal computational resource use. The
397 installation and configuration of SwiftOrtho is simple and does not require the user
398 to have any experience in database configuration. It is easy to use, the only input
399 required by SwiftOrtho is a FASTA format file of protein sequences with taxonomy

400 information in the header line. Furthermore, SwiftOrtho is highly modular, and can

401 be used to

402 SwiftOrtho can be integrated into various common pipelines where fast orthology

403 classification is required such as pan-genome analysis, large-scale phylogenetic tree

404 construction, and other multi-genome analyses. It is specifically suited for microbial

405 community analyses, where large number of sequences and species are involved.

406 **Availability of data and materials**

407 SwiftOrtho was written in Python 2.7 and is available at <https://github.com/Rinoahu/SwiftOrtho>

408 under a GPLv3 license.

24. Altenhoff, A.M., Boeckmann, B., Capella-Gutierrez, S., Dalquen, D.A., DeLuca, T., Forslund, K., Huerta-Cepas, J., Linard, B., Pereira, C., Pryszcz, L.P., Schreiber, F., Da Silva, A.S., Szklarczyk, D., Train, C.M., Bork, P., Lecompte, O., Von Mering, C., Xenarios, I., Sjölander, K., Jensen, L.J., Martin, M.J., Muffato, M., Gabaldón, T., Lewis, S.E., Thomas, P.D., Sonnhammer, E., Dessimoz, C.: Standardized benchmarking in the quest for orthologs. *Nat. Methods* (2016). doi:[10.1038/nmeth.3830](https://doi.org/10.1038/nmeth.3830)
25. Pearson, W.R., Lipman, D.J.: Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci.* **85**(8), 2444–2448 (1988). doi:[10.1073/pnas.85.8.2444](https://doi.org/10.1073/pnas.85.8.2444)
26. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. *J. Mol. Biol.* **215**(3), 403–410 (1990). doi:[10.1016/S0022-2836\(05\)80360-2](https://doi.org/10.1016/S0022-2836(05)80360-2). arXiv:[1611.08307v1](https://arxiv.org/abs/1611.08307v1)
27. Kent, W.J.: BLAT — The BLAST -Like Alignment Tool. *Genome Research* **12**, 656–664 (2002). doi:[10.1101/gr.229202](https://doi.org/10.1101/gr.229202).
28. Shirayev, S.A., Papadopoulos, J.S., Schäffer, A.A., Agarwala, R., Schaffer, A.A., Agarwala, R.: Improved BLAST searches using longer words for protein seeding. *Bioinformatics* **23**(21), 2949–2951 (2007). doi:[10.1093/bioinformatics/btm479](https://doi.org/10.1093/bioinformatics/btm479)
29. Ma, B., Tromp, J., Li, M.: PatternHunter: faster and more sensitive homology search. *Bioinformatics* **18**(3), 440–445 (2002). doi:[10.1093/bioinformatics/18.3.440](https://doi.org/10.1093/bioinformatics/18.3.440)
30. Ilie, L., Ilie, S., Khoshraftar, S., Bigvand, A.M.: Seeds for effective oligonucleotide design. *BMC Genomics* **12**(1), 280 (2011). doi:[10.1186/1471-2164-12-280](https://doi.org/10.1186/1471-2164-12-280)
31. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. *J. Mol. Biol.* **147**(1), 195–197 (1981). doi:[10.1016/0022-2836\(81\)90087-5](https://doi.org/10.1016/0022-2836(81)90087-5)
32. Chao, K.M., Pearson, W.R., Miller, W.: Aligning two sequences within a specified diagonal band. *Bioinformatics* **8**(5), 481–487 (1992). doi:[10.1093/bioinformatics/8.5.481](https://doi.org/10.1093/bioinformatics/8.5.481)
33. Landès, C., Risler, J.L.: Fast databank searching with a reduced amino-acid alphabet. *Computer applications in the biosciences : CABIOS* **10**(4), 453–454 (1994)
34. Murphy, L.R., Wallqvist, A., Levy, R.M.: Simplified amino acid alphabets for protein fold recognition and implications for folding. *Protein Eng. Des. Sel.* **13**(3), 149–152 (2000). doi:[10.1093/protein/13.3.149](https://doi.org/10.1093/protein/13.3.149)
35. Peterson, E.L., Kondev, J., Theriot, J.A., Phillips, R.: Reduced amino acid alphabets exhibit an improved sensitivity and selectivity in fold assignment. *Bioinformatics (Oxford, England)* **25**(11), 1356–1362 (2009). doi:[10.1093/bioinformatics/btp164](https://doi.org/10.1093/bioinformatics/btp164)
36. Edgar, R.C.: Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic acids research* **32**(1), 380–5 (2004). doi:[10.1093/nar/gkh180](https://doi.org/10.1093/nar/gkh180)
37. Ye, Y., Choi, J.-H., Tang, H.: RAPSearch: a fast protein similarity search tool for short reads. *BMC Bioinformatics* **12**(1), 159 (2011). doi:[10.1186/1471-2105-12-159](https://doi.org/10.1186/1471-2105-12-159)
38. Gibbons, T.R., Mount, S.M., Cooper, E.D., Delwiche, C.F.: Evaluation of BLAST-based edge-weighting metrics used for homology inference with the Markov Clustering algorithm. *BMC Bioinformatics* **16**(1) (2015). doi:[10.1186/s12859-015-0625-x](https://doi.org/10.1186/s12859-015-0625-x)
39. Enright, A.J., Van Dongen, S., Ouzounis, C.A.: An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res.* **30**(7), 1575–1584 (2002). doi:[10.1093/nar/30.7.1575](https://doi.org/10.1093/nar/30.7.1575). [journal.pone.0035671](https://doi.org/10.1371/journal.pone.0035671)
40. Emms, D.M., Kelly, S.: OrthoFinder: solving fundamental biases in whole genome comparisons dramatically improves orthogroup inference accuracy. *Genome Biology* **16**(1), 157 (2015). doi:[10.1186/s13059-015-0721-2](https://doi.org/10.1186/s13059-015-0721-2)
41. Davis, J.J., Gerdes, S., Olsen, G.J., Olson, R., Pusch, G.D., Shukla, M., Vonstein, V., Wattam, A.R., Yoo, H.: PATtyFams: Protein families for the microbial genomes in the PATRIC database. *Front. Microbiol.* **7**(FEB), 118 (2016). doi:[10.3389/fmicb.2016.00118](https://doi.org/10.3389/fmicb.2016.00118)
42. Frey, B.J., Dueck, D.: Clustering by passing messages between data points. *Science* **315**(5814), 972–6 (2007). doi:[10.1126/science.1136800](https://doi.org/10.1126/science.1136800). [1401.2548](https://doi.org/10.1126/science.1136800)
43. Lam, S.K., Pitrou, A., Seibert, S.: Numba: A LLVM-based python JIT compiler. *Proc. Second Work. LLVM Compil. Infrastruct. HPC - LLVM '15*, 1–6 (2015). doi:[10.1145/2833157.2833162](https://doi.org/10.1145/2833157.2833162)
44. Curwen, V., Eyras, E., Andrews, T.D., Clarke, L., Mongin, E., Searle, S.M.J., Clamp, M.: The Ensembl automatic gene annotation system. *Genome Res.* **14**(5), 942–950 (2004). doi:[10.1101/gr.1858004](https://doi.org/10.1101/gr.1858004)
45. Benson, D.A.: GenBank. *Nucleic Acids Res.* **28**(1), 15–18 (2000). doi:[10.1093/nar/28.1.15](https://doi.org/10.1093/nar/28.1.15)
46. Brohée, S., van Helden, J.: Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics* (2006). doi:[10.1186/1471-2105-7-488](https://doi.org/10.1186/1471-2105-7-488)
47. Cosentino, S., Iwasaki, W.: Sonicparanoid: fast, accurate and easy orthology inference. *Bioinformatics* **35**(1), 149–151 (2019). doi:[10.1093/bioinformatics/bty631](https://doi.org/10.1093/bioinformatics/bty631)
48. Rigo, A., Pedroni, S.: PyPy ’ s Approach to Virtual Machine Construction. *Companion to 21st ACM SIGPLAN Symp.*, 944–953 (2006). doi:[10.1145/1176617.1176753](https://doi.org/10.1145/1176617.1176753)
49. Bratlie, M.S., Johansen, J., Sherman, B.T., Huang, D.W., Lempicki, R.A., Drablos, F.: Gene duplications in prokaryotes can be associated with environmental adaptation. *BMC Genomics* (2010). doi:[10.1186/1471-2164-11-588](https://doi.org/10.1186/1471-2164-11-588)
50. Katju, V., Bergthorsson, U.: Copy-number changes in evolution: Rates, fitness effects and adaptive significance (2013). doi:[10.3389/fgene.2013.00273](https://doi.org/10.3389/fgene.2013.00273)

Figures

Figure 1 Orthology Inference Algorithm. Nodes are gene names, edges are similarity score of pairwise genes. 1. A_1 - B_1 are putative orthologs identified by RBH. 2. A_1 - A_2 and B_1 - B_2 are putative in-paralogs as the bit scores of these pairs greater than A_1 - B_1 ; 3. A_2 - B_1 and A_2 - B_2 are putative co-orthologs as these pairs are not orthologs but A_1 - B_1 are orthologs and A_1 - A_2 , B_1 - B_2 are in-paralogs.

Figure 2 Comparing Standard Smith-Waterman with Banded Smith-Waterman. **A.** Similarity score matrix for Standard Smith-Waterman. Standard Smith-Waterman algorithm need to calculate all the entries. **B.** Similarity score matrix for Banded Smith-Waterman. Banded Smith-Waterman algorithm only need to calculate the entries on and near the diagonal.

Figure 3 Evaluation of predicted orthologous groups. **A.** Definition of precision and recall. **OG:** orthologous group, **FN:** genes only found in true orthologous group, **TP:** genes shared between true and predicted orthologous group, **FP:** genes only found in predicted orthologous group; **B.** Evaluation of different tools on OrthoBench2. **SwiftOrtho+MCL:** SwiftOrtho with MCL; **SwiftOrtho+APC:** SwiftOrtho with Affinity Propagation Clustering.

Figure 4 Comparing BLASTP and SwiftOrtho's homology search module on the quality of orthologous groups prediction. BLASTP and SwiftOrtho's search module perform an all-*vs*-all search on the *Euk* set, respectively. Then, all the orthology prediction tools were employed for orthology inference. Finally, the predicted orthology relationships were clustered into orthologous groups by MCL algorithm.

Figure 5 Markov Clustering versus Affinity Propagation Clustering. Both algorithms were applied to cluster the orthology relationships of the *Euc* set inferred by different orthology prediction tools, into orthologous groups. As OrthFinder does not report orthology relationships, the Affinity Propagation can not apply to its results. **MCL:** Markov Clustering algorithm; **APC:** Affinity Propagation Clustering.

Figure 6 The Benchmarking in Quest for Orthologs. **A:** Species Tree Discordance Benchmark. InParanoid has minimum average Robinson-Foulds distance. SwiftOrtho's average RF distance is close to InParanoid. The prediction inferred by OrthFinder is not available in this test; **B:** Generalized Species Tree Discordance Benchmark. InParanoid has minimum average Robinson-Foulds distance. The prediction inferred by OrthFinder is not available in this test; **C:** Agreement with Reference Gene Phylogenies of SwissTree. SwiftOrtho has the highest positive prediction value rate(Recall). InParanoid has the highest true positive rate(Precision); **D:** Agreement with Reference Gene Phylogenies of TreeFam-A. SonicParanoid has the highest positive prediction value rate(Recall), however, its true positive rate(Precision) is close to zero. SwiftOrtho has the second highest Recall and Precision; **E:** Gene Ontology conservation test. OrthoMCL has the highest average Schlicker similarity; **F:** Enzyme Classification conservation test. SwiftOrtho has the highest average Schlicker similarity. OrthoMCL detected the most orthology relationships and has the highest Recall.

Tables

Additional Files

Additional file 1 —

Metadata for the genome assemblies of the *Bac* set (tab-delimited text file).

<https://figshare.com/s/19a006d6fea9c2494ab8>

Figure 1

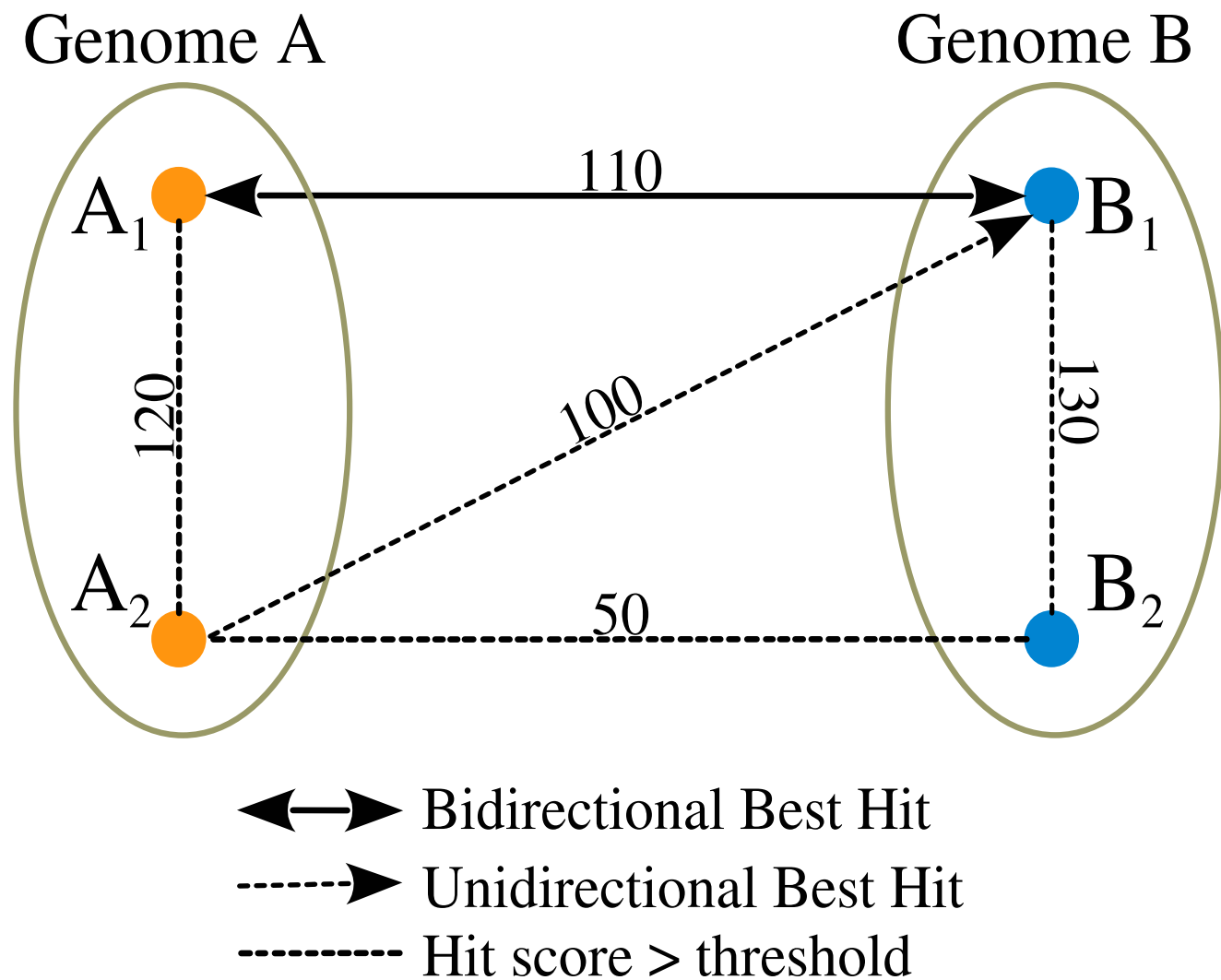


Figure 2

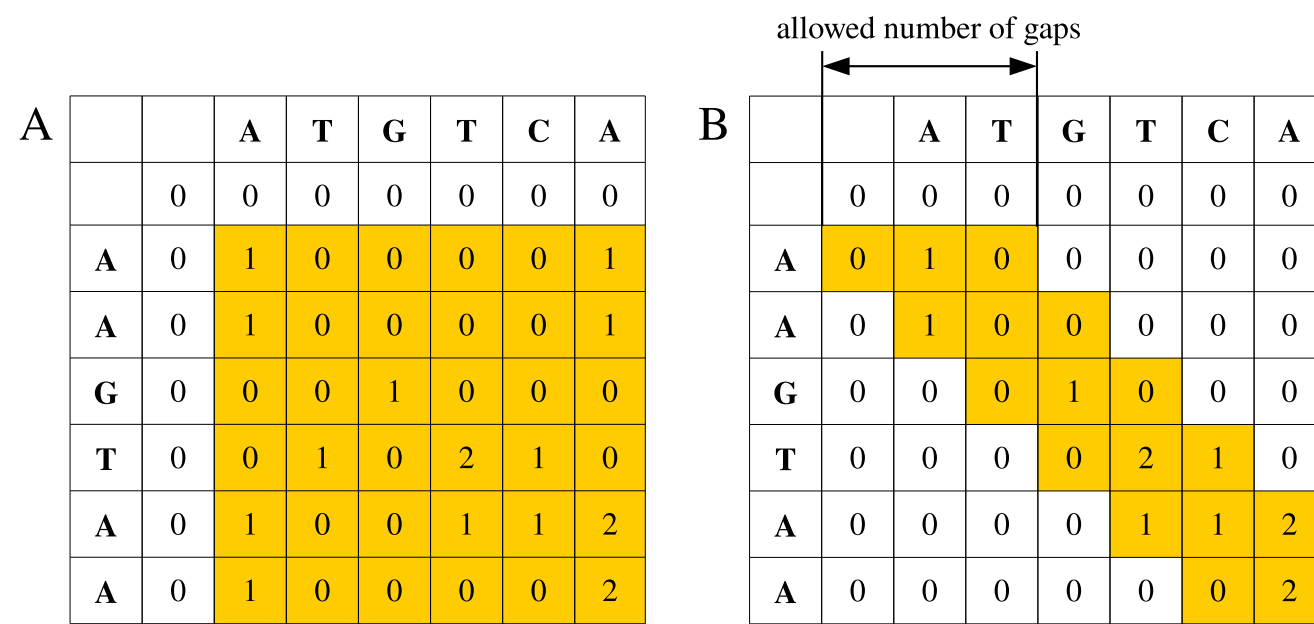


Figure 3

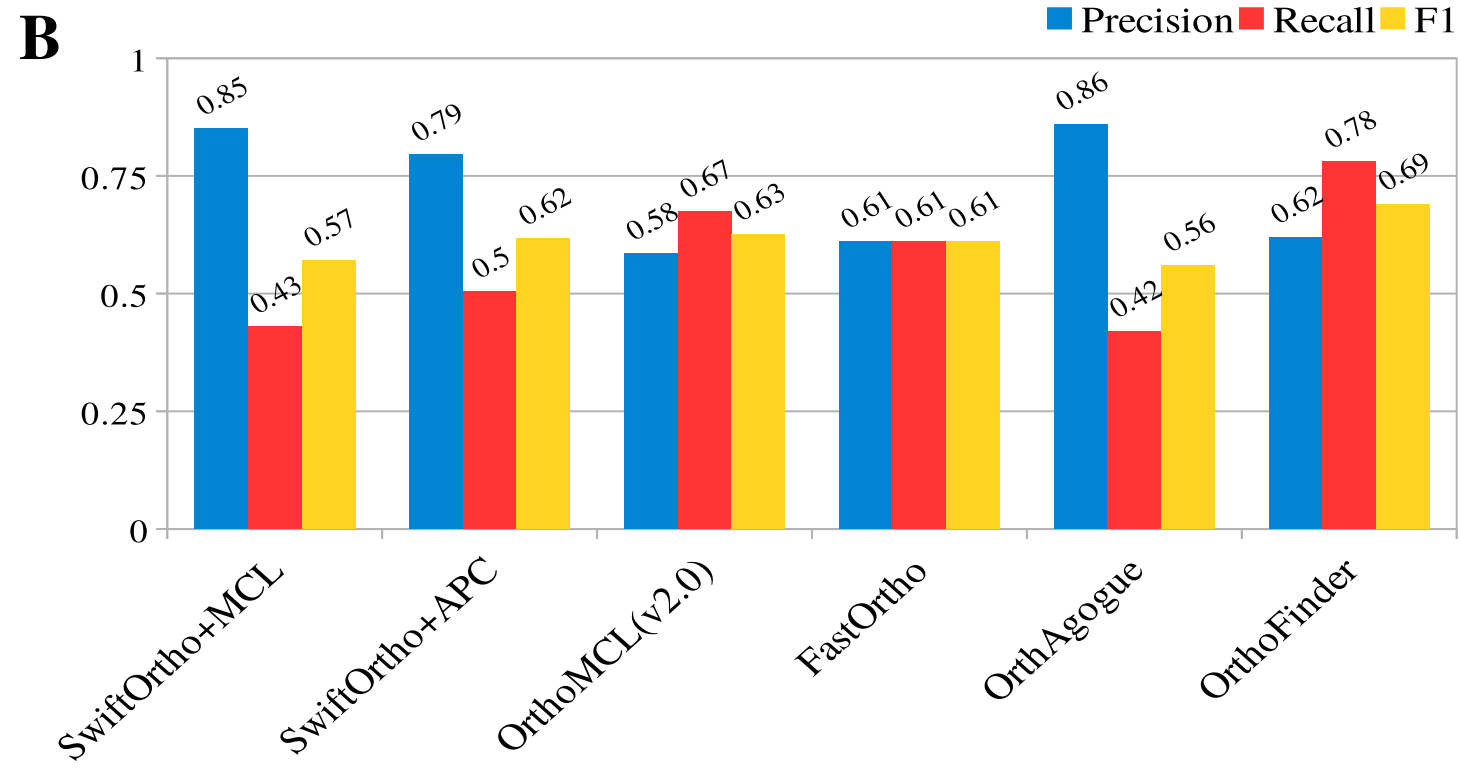
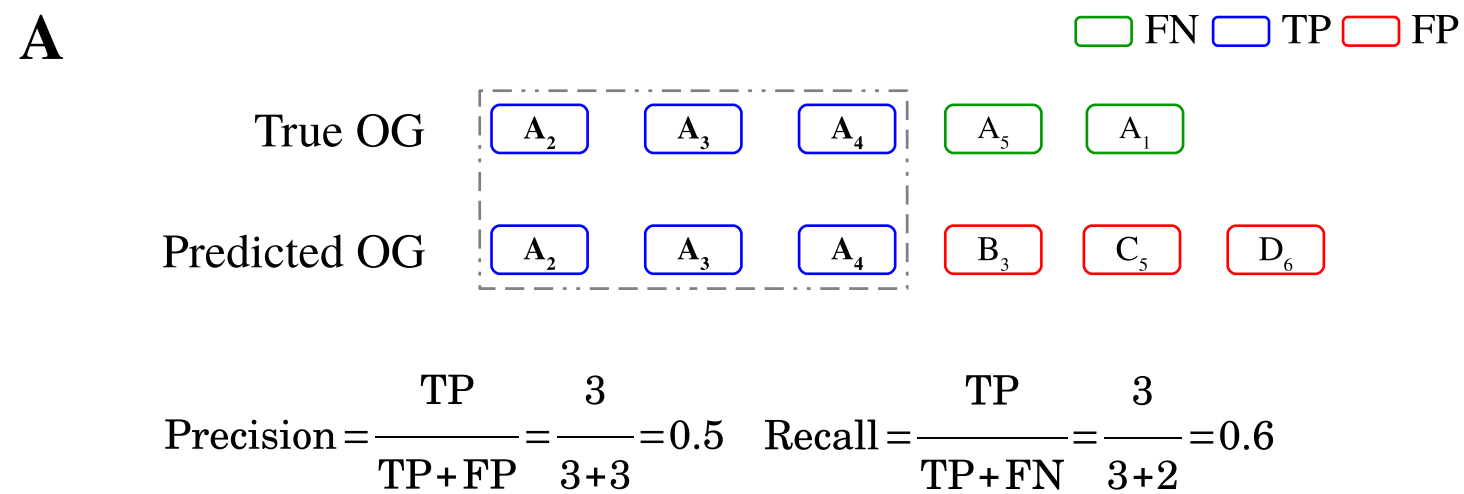


Figure 4

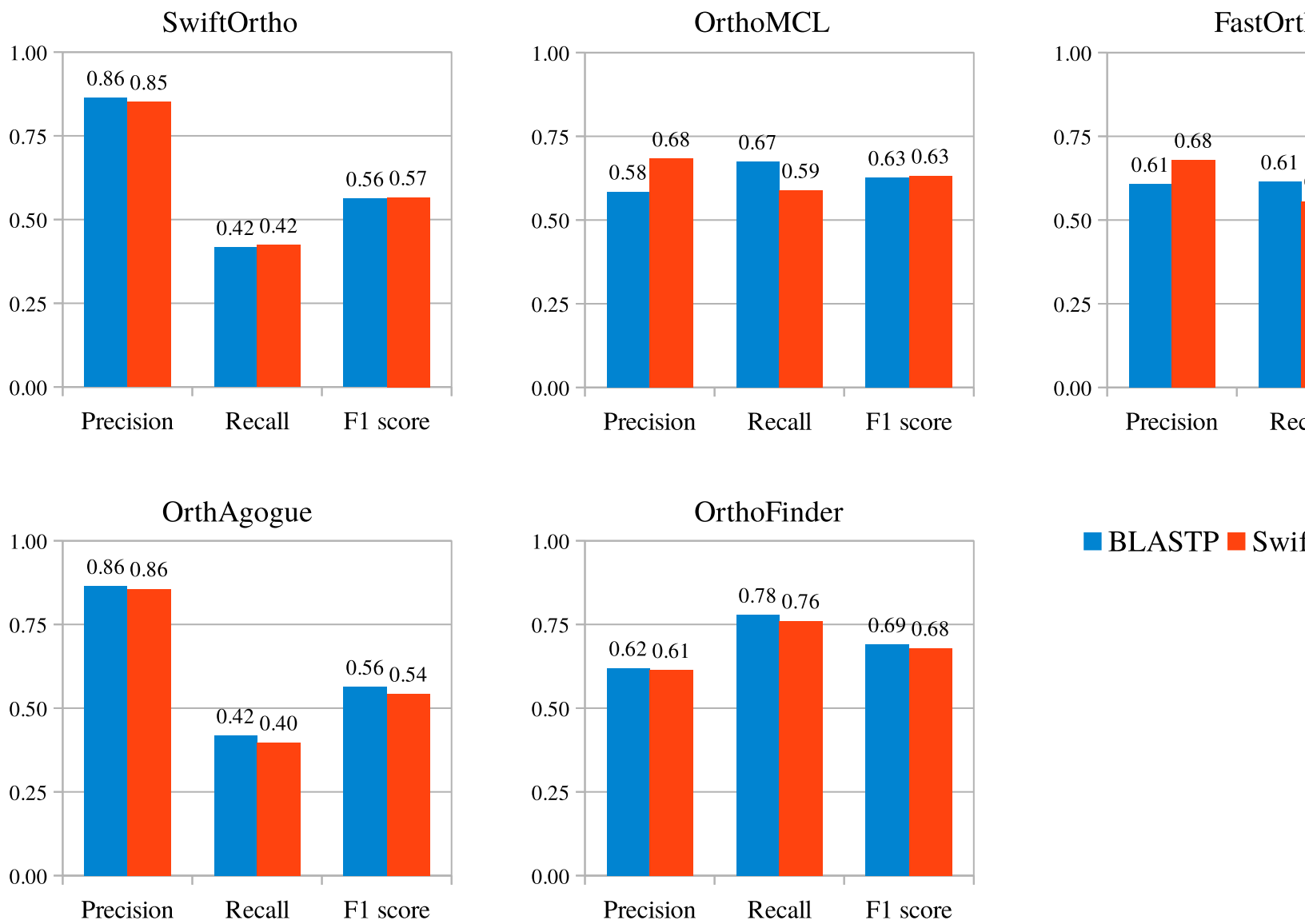


Figure 5

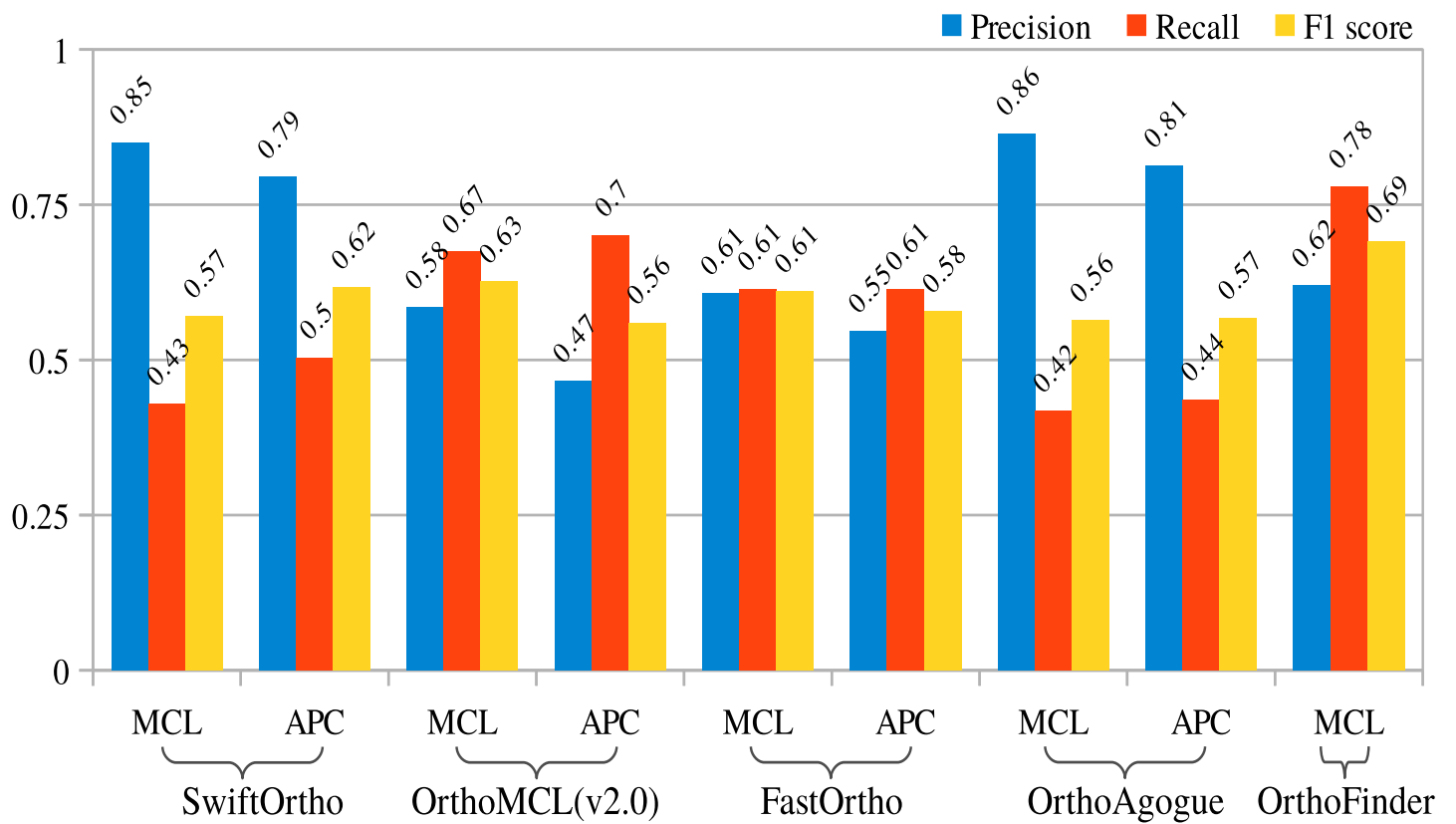


Figure 6

