Controlling Size When Aligning Multiple Genomic Sequences with Duplications

Minmei Hou[1], Piotr Berman[1], Louxin Zhang[2], and Webb Miller[1]

[1] Department of Computer Science and Engineering, Penn State, University Park, PA 16801, USA,
mhou@cse.psu.edu, berman@cse.psu.edu, webb@bx.psu.edu
[2] Department of Mathematics, National University of Singapore, Science Drive 2, Singapore 117543
matzlx@nus.edu.sg

Abstract. For a genomic region containing a tandem gene cluster, a proper set of alignments needs to align only orthologous segments, i.e., those separated by a speciation event. Otherwise, methods for finding regions under evolutionary selection will not perform properly. Conversely, the alignments should indicate every orthologous pair of genes or genomic segments. Attaining this goal in practice requires a technique for avoiding a combinatorial explosion in the number of local alignments. To better understand this process, we model it as a graph problem of finding a minimum cardinality set of cliques that contain all edges. We provide an upper bound for an important class of graphs (the problem is NP-hard and very difficult to approximate in the general case), and use the bound and computer simulations to evaluate two heuristic solutions. An implementation of one of them is evaluated on mammalian sequences from the α-globin gene cluster.

1 Introduction

The ENCODE project [22] has the goal of identifying all functional genomic segments in 1% of the human genome. As part of the project, genomic sequence data from a number of mammals are being generated for the targeted 1%, in the belief that alignment and analysis of the sequences will help predict the functional segments. Several computer programs for aligning genomic regions have been used for this purpose [15,16]. In our opinion, the current crop of alignment programs performs acceptably in many parts of the genome. However, for regions containing tandem gene clusters, more software development is necessary. The deficiency of current methods can be explained using the following long-accepted concepts.

According to standard biological jargon [6,7], two sequences are *homologs* if they are evolutionarily related, in which case they diverged at either a duplication event (and are called *paralogs*) or a speciation event (and are called *orthologs*). It is widely appreciated that a basic goal of alignment algorithms is to align sequences if and only if they are homologs. We feel that a better statement of the goal is to align precisely the orthologs. That is, we want the evolutionary relationship among aligned sequences to be the same as the phylogenetic tree relating the species for those sequences. A main (and probably *the* main) use of alignments is to identify intervals within the aligned segments in which the similarity/divergence pattern differs from neutral evolution, and modern methods for detecting such intervals [21,5,23] require, for their proper functioning, that aligned rows be orthologs. In regions of the genome where no intervals have been duplicated, orthology is equivalent to homology, and existing alignment methods are effective. However, for tandem gene clusters, we know of no existing aligner that does a good job.

To represent duplications and other large-scale evolutionary rearrangements, our programs for aligning several genomic sequences produce a set of alignment "blocks", each of which is in essence a traditional alignment of segments from the given sequences or their reverse complements [2]. With duplications, the same sequence position can appear in several blocks. It is useful to note that if rows of a block are pairwise orthologous, then no two rows can be from the same species.

To build a new alignment program that obeys the two requirements

(a) any two rows of a computed block are orthologous and
(b) any pair of orthologous positions appears together in at least one block,

two hurdles had to be overcome. The first was to distinguish orthologs from paralogs, and for this there was a large literature to draw from. We provide our solution in another paper [11]. The second difficulty was that the number of possible blocks can grow exponentially with the number of sequences and duplications, which is the topic of this paper. For instance, a straightforward implementation meeting our requirements produced over 900 Mbytes of alignments when applied to intervals containing the α-globin gene clusters of 20 mammals, where the total length of the original sequences was only 3.9 Mb. We designed and implemented a space-saving strategy that decreased the amount of output to 8.7 Mb, while still fulfilling requirements (1) and (2). New ideas were required to achieve this savings, and we were led to the development of a theoretical model that turned out, in the general case, to be equivalent to a previously studied NP-complete combinatorial optimization problem, which we will call MinCliqueCov, namely, finding a minimum cardinality set of cliques that contains all edges of a given undirected graph. We show that the graphs we study have special properties, and they can be utilized to apply divide-conquer techniques, which would not work well with an arbitrary graph.

Here we describe our graph-theoretic model and derive a theoretical upper bound on the number of blocks that are needed to meet our requirements in an important subclass of problems. Also, using the model, we formulate two heuristic methods, and with the help of our upper bound and some computer simulations, we measure where the two methods lie in the tradeoff between computation time and output size. We also compare our solutions to an existing heuristic method for general graphs [13]. Finally, we describe the performance on the α-globin gene cluster of our alignment software that is based on the new ideas.

2 Methods

2.1 A Graph-Theoretic Model

Let $G = (V, E)$ be a graph with vertex set V and edge set E. An m-vertex complete subgraph of G is called an *m-clique*. A *clique cover* of G is a set of cliques whose edges contain every edge $e \in E$. The *clique cover number*, $cc(G)$, of G is defined to be the minimum number of cliques in a clique cover of G.

Assume we align genomic sequences from K species in a genomic region containing a family of tandemly duplicated genes. Suppose that each member of that gene family can be aligned to every orthologous member. In our model, a vertex represents a gene in one of the species, and there is an edge between two vertices if the genes that they represent are orthologous. Thus, we obtain a K-partite graph, called an *alignment graph*, where each part contains the nodes that represent the gene family members in a given species. A multi-alignment block with pairwise orthologous rows corresponds to a clique, and a set of multi-alignment blocks that contains every pairwise alignment (condition (2) in the Introduction) corresponds to a clique cover. Thus it would be helpful to solve MinCliqueCov for the alignment graph. Figure 1 gives an example in which

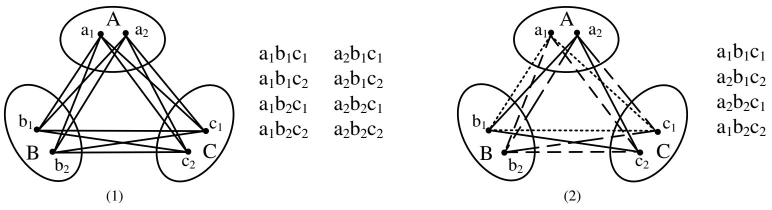

Fig. 1. A trivial example on minimum clique cover. Panel 1 shows 8 cliques to cover all edges, while panel 2 shows 4 cliques to cover all edges.

there are three species (A, B and C), each containing two members of a gene family. Each pair of genes from different species is orthologous. In these example, there are eight possible alignment blocks, but four blocks are sufficient to include each orthologous pair in a block.

Unfortunately, MinCliqueCov is NP-hard [17]. The restriction to multipartite graphs does not make this problem easier since a graph of n vertices is trivially an n-partite graph in which each part has only one vertex. Various techniques have been applied to solve MinCliqueCov and closely related problems [9,18,3]. For instance, when the degree of any vertex in G is at most 4, the problem is solvable in linear time [19].

2.2 A Special Case

In this section, we investigate the graph structure that arises under certain natural conditions, namely when all duplications have occurred after all speciation events. That is, we suppose that each gene is orthologous to every gene in a different species. Moreover, to keep things simple, we supposed that each of K species has precisely P copies of the gene. The resulting alignment graph is a complete K-partite graph: each part has P nodes, and there is an edge between any two nodes that are in two different parts. We denote such a graph by $G_{K,P}$. Note that the shape of such graph is determined for each pair of K and P. Thus MinCliqueCov restricted to graphs of the form $G_{K,P}$ has $O(n)$ distinct instances with n nodes, where $n = KP$ (because n can be factored into KP in less than n ways). It was shown that problems with polynomially many instances per size cannot be NP-hard (unless P=NP) [1]. But even for the case of $P = 2$, we do not know the exact solution. The technical result of this paper does not imply the problem is easy either. The purpose of this section is to derive a non-trivial upper bound on $cc(G_{K,P})$, which will help us to interpret the results of simulations that we report below.

First, though, let us mention lower bounds. For $K \geq 2$, $cc(G_{K,P}) \geq P^2$, since there are P^2 edges between any two parts of the graph, and any clique can contain at most one of those edges. Moreover, it was recently proved that $cc(G_{K,P}) \geq \log_b(KP)$, where $b = \frac{P}{(P-1)^{(P-1)/P}}$ [4]. That lower bound is approximately equal to $P(\log_P K + 1)$.

For an upper bound, it has been known for some time that $cc(G_{K,2}) = \Theta(\log_2 \mathrm{K})$ [9], but we seek a bound for general P. Assume $G_{K,P}$ has the following node set and edge set:

$$\begin{aligned} V &= \{u(i,j):\ 0 \leq i < K \text{ and } 0 \leq j < P\} \\ E &= \{\{u(i,j), u(k,l)\} \subset V:\ i \neq k\} \end{aligned}$$

To present a recursive construction of small clique covers of $G_{K,P}$, we start with two simple observations.

Observation 1. *Let $U \subseteq V$. If $\mathcal{C}$ is a clique cover of G, then $\{C \cap U:\ C \in \mathcal{C}\}$ is a clique cover of $G(U)$. Thus $cc(G(U)) \leq cc(G)$.*

Observation 2. *If $\mathcal{C}_i$ is a clique cover of $< V, E_i >$ for $i = 1, 2 \ldots, k$ then $\bigcup_{i=1}^{k} \mathcal{C}_i$ is a clique cover of $< V, \bigcup_{i=1}^{k} E_i >$.*

The edges of $G_{K,P}$ can be split into two sets

$$\begin{aligned} E_0 &= \{\{u(i,j), u(k,l)\} \in E:\ j = l\} \\ E_1 &= E - E_0 \end{aligned}$$

We can cover $< V, E_0 >$ with cliques $C_j = \{u(i,j) \in V\}$, $j = 0, \ldots, P-1$. Now it remains to find a clique cover for $G^1_{K,P} =< V, E_1 >$.

If there are $K = M \times L$ species, the edges of $G^1_{ML,P}$ can be represented as $E_2 \cup E_3$, where

$$\begin{aligned} E_2 &= \{\{u(i,j), u(k,l)\} \in E_1:\ \lfloor i/L \rfloor \neq \lfloor k/L \rfloor\} \\ E_3 &= \{\{u(i,j), u(k,l)\} \in E_1:\ i \textbf{ mod } L \neq k \textbf{ mod } L\} \end{aligned}$$

To make that split more intuitive, put the ML parts of $G_{ML,P}$ into a matrix with M rows and L columns. An edge between different parts will either connect parts from different rows, or parts from different columns. Denote the set of edges connecting parts from different rows by E_2, and the set of edges connecting parts from different columns by E_3. Note that E_2 and E_3 are not necessarily disjoint.

Lemma 1. $cc(< V, E_2 >) \leq cc(G^1_{M,P})$.

Proof. Consider a clique cover $\mathcal{C}$ of $G^1_{M,P}$. Obtain $\mathcal{C}'$ by transforming each clique $C \in \mathcal{C}$ into

$$C' = \{u(i,j) \in V : u(\lfloor i/L \rfloor, j) \in C\}.$$

C' is still a clique, because if $\lfloor i/L \rfloor \neq \lfloor k/L \rfloor$ than $i \neq k$. Now an edge $e = \{u(i,j), u(k,l)\}$ of E_1 is covered by $\mathcal{C}'$ unless $\lfloor i/L \rfloor = \lfloor k/L \rfloor$, hence $e \notin E_2$. ❒

Lemma 2. $cc(< V, E_3 >) \leq cc(G^1_{L,P})$.

Proof. Similar to Lemma 1. ❒

Lemma 3. $cc(G^1_{P,P}) \leq P(P-1)$ *if* P *is prime.*

Proof. We construct a set of cliques $C_{a,b} = \{\{u(i, ai + b \ \mathbf{mod}\ P) : \ 0 \leq i < P\}$ where $0 < a < P$ and $0 \leq b < P$. Given an edge $\{u(i,j), u(k,l)\} \in E_1$ we can find the parameters a, b of the clique that covers it by solving linear system

$$ai + b = j \ \mathbf{mod}\ P$$
$$ak + b = l \ \mathbf{mod}\ P$$

This system yields the following equation for a: $a(i-k) = j - l \ \mathbf{mod}\ P$. Because $i \neq k$, $i - k$ has a reciprocal $\mathbf{mod}\ P$, and because $j \neq l$, the a computed from this is non-zero. ❒

Theorem 1. $cc(G_{K,P}) \leq P + P(P-1)\lceil \log_P K \rceil$ *if* P *is a prime.*

Proof. By Observation 1, it suffices to prove that $cc(G_{K,P}) \leq P + aP(P-1)$ for $K = P^a$, where a is an integer. Because we can cover E_0 with P cliques, it suffices to prove that $cc(G^1_{K,P}) \leq aP(P-1)$. We can show it by induction on a. For $a = 1$ this is proven in Lemma 3. Assuming it is proven for $a-1$, we have $K = ML$ where $M = P$ and $L = P^{a-1}$. This allows to apply Lemmas 1 and 2 to show that $cc(G^1_{K,P}) \leq (a-1)P(P-1) + P(P-1) = aP(P-1)$. ❒

When P is not a prime, we can use the above result for P', where P' is the smallest prime larger than P. For example, when $P = 6$ and $K = 2$, we have a trivial solution with 36 cliques (indeed, edges), but when $K = 3$, we can use a solution for $P' = 7$ that has 49 cliques, and this solution works for $K \leq 7$, while for $K \leq 343$ we have a solution with $7 + 2 \times 42 = 91$ cliques.

Of course, better solutions may exist. It is easy to find a solution for $G^1_{3,4}$ with 12 cliques; see Figure 2. We can apply the reasoning of Theorem 1 to show that $cc(G_{K,4}) \leq 4 + 12\lceil \log_3 K \rceil$ which, except for $K = 4, 5$, is better than $5 + 20\lceil \log_5 K \rceil$. Thus, $cc(G_{K,P})$ is at most $P(P-1)\log_P K + P$ when P is a prime number and K is in power of P. In cases that P and K do not satisfy these conditions, the values can be approximated by the nearest prime number and its power. This upper bound is conjectured to be tight, but this has not been proved.

$$\overbrace{\begin{array}{c}\{a_0,b_0,c_0\}\\ \{a_1,b_1,c_1\}\\ \{a_2,b_2,c_2\}\\ \{a_3,b_3,c_3\}\end{array}}^{E_0}\qquad \overbrace{\begin{array}{cccc}\{a_0,b_1,c_2\} & \{a_1,b_0,c_3\} & \{a_2,b_0,c_1\} & \{a_3,b_0,c_2\}\\ \{a_0,b_2,c_3\} & \{a_1,b_2,c_0\} & \{a_2,b_1,c_3\} & \{a_3,b_1,c_0\}\\ \{a_0,b_3,c_1\} & \{a_1,b_3,c_2\} & \{a_2,b_3,c_0\} & \{a_3,b_2,c_1\}\end{array}}^{E_1}$$

Fig. 2. Clique cover of $G_{3,4}$ with 16 cliques

2.3 Heuristic Solutions

MinCliqueCov is NP-hard, but also has no efficient approximation algorithm unless NP=P [14], so we have to use heuristics. We propose two heuristic methods for generating clique covers for complete multi-partite graphs studied in the previous section. It is then relatively straightforward to adapt these methods for the multi-alignment problem in tandem gene clusters, even in the general case of arbitrary orthology relationships; the next section discusses some of the issues that arise. However, in the idealized setting of $G_{K,P}$, with the upper bound derived above, we can evaluate how close they come to an ideal reduction in output size.

Both heuristic methods follow a divide-and-conquer strategy: Partition G into two graphs, G_1 and G_2, each having about $K/2$ parts, find clique covers CC_1 and CC_2 of G_1 and G_2 respectively, and merge them to obtain a clique cover CC of G. While these methods do not give the fewest cliques, they efficiently find a relatively small clique cover. The merge procedures can be described as follows, where "uncovered" refers to an edge not currently in a clique in CC.

Heuristics for MinCliqueCov were studied already in 1978 [13]. Recently an exact solution was also proposed [8] for the general graph. Unfortunately, our complete multi-partite graph is quite dense, and the reduction rules from [8] cannot be applied here. Thus we only compare our heuristic methods' performance with [13]'s method.

Merge I forms a new clique from the pair of cliques that maximizes the number of additional covered edges. Merge II processes cliques of CC_1 and CC_2 in a random order and forms any new clique that covers at least one new edge. Running times are dominated by the number of executions of the loops $2a$ and $2b$, respectively, which are essentially the number of cliques generated. The lowest-level operation inside loop $2a$ is to examine whether an edge is covered or not; for loop $2b$ it is to decide whether a clique contains a certain edge or not. Both operations involve an array access and can be regarded as taking unit time. The number of unit operations inside loop $2a$ is $|CC_1| \cdot |CC_2| \cdot K_1 \cdot K_2$, where K_1 and K_2 are the number of partitions (species) of G_1 and G_2. The number of unit operations inside loop $2b$ is $|CC_1| \cdot K_1 + |CC_2| \cdot K_2$. The performance of these two methods, in terms of both actually running time and the number of cliques generated, is analyzed below.

2.4 Application in the Aligner Program

The above divide-and-conquer methods can be adapted to so-called "progressive" multiple alignment programs, which work leaves-to-root in the phylogenetic tree

```
Merge I ($CC_1$, $CC_2$)
1   $CC \leftarrow \phi$
2a  **while** there exists an uncovered edge
3       For each pair of cliques $c_i$ and $c_j$ from $CC_1$ and $CC_2$ respectively
4           $u_{ij} \leftarrow$ the number of uncovered edges of $c_i \cup c_j$
5       $(c_{maxi}, c_{maxj}) \leftarrow$ the pair with maximum $u_{ij}$
6       insert $c_{maxi} \cup c_{maxj}$ to $CC$
7   Output $CC$
```

```
Merge II ($CC_1$, $CC_2$)
1   $CC \leftarrow \phi$
2b  **while** there exists an uncovered edge $(u, v)$ between subproblems
3       $c_1 \leftarrow$ a clique from $CC_1$ that contains $u$
4       $c_2 \leftarrow$ a clique from $CC_2$ that contains $v$
5       insert $c_1 \cup c_2$ into $CC$
// We still have to incorporate unused cliques from each subproblem
6   **while** there exists an unused clique $c_1$ from $CC_1$
7       $c_2 \leftarrow$ an unused clique in $CC_2$; if none, then any clique in $CC_2$
8       insert $c_1 \cup c_2$ into $CC$
9   **while** there exists an unused clique $c_2$ from $CC_2$
10      $c_1 \leftarrow$ an unused clique in $CC_1$; if none, then any clique in $CC_1$
11      insert $c_1 \cup c_2$ to $CC$
12  Output $CC$
```

for the given species. At a tree node, these aligners merge multiple alignments from the sub-trees, which is analogous to merging cliques for subgraphs G_1 and G_2, except that the split into subproblems might not be balanced. The process of merging blocks from the left and right subtree is guided by pairwise alignments between a species in the left subtree and a species in the right subtree. The set of multi-alignment blocks corresponding to the tree's root constitutes the multiple alignment of the original K species.

Thus, the use of the guide tree by the aligner can be viewed as the recursive partition of the alignment graph, and such a partition can be used both by MERGE I and MERGE II. In our current BOAST aligner, we have decided to apply MERGE II since our tests indicated that MERGE I could be about 1000 times slower (see Figure 4). While MERGE I produced fewer cliques(see Figure 3), i.e., alignment blocks, MERGE II produced a number that was acceptably small.

We need to address the following issue. The alignment graph generally has fewer edges than a complete K-partite graph, since some speciation events are preceded by duplications. Consider what we should expect if we have a perfect alignment graph, with all ortholog/paralog relationships properly diagnosed. We have two siblings T_1 and T_2 with common ancestor T and let c_i be a clique in the subgraph of T_i, $i = 1, 2$. One can see that either $c_1 \cup c_2$ is also a clique, or there are no edges between c_1 and c_2 (see [11] for the analysis of orthologous inference). In this case, when MERGE II selects a pair of cliques with union that covers at least one new edge (lines 3-4), this union is a clique. When MERGE II processes an unused clique c_1 in line 7 and 10, we first look for a clique c_2 in another sub-tree that is connected to c_1, and the union of them becomes a new clique; if none exists we simply add c_1 to the output.

Clearly, this adaptation may be incorrect if we have an imperfect alignment graph, ie., with some edges established wrongly and some correct edges missing. Consequently, it may happen that a pair of selected cliques, c_1 and c_2, has some connections, but not all. If only a minority of the possible edges between c_1 and c_2 are present, we behave as if c_1 and c_2 were not connected at all, and if the majority is present, we behave as if $c_1 \cup c_2$ was a clique. This majority criterion can have the effect of correcting some errors created by incorrect identification of orthologous pairwise alignments [11]. Note that we have to choose between two kinds of discrepancies in the output. One is that we do not include all actual orthologous relationship in a blocks. The second is that we contaminate a block with a paralogous relationship. If one kind of discrepancy is more harmful than the other, we can replace the majority criterion with some other ratio.

3 Results

3.1 Simulations

Methods Merge I and II were tested on graphs of the form $G_{K,P}$, i.e, complete K-partite graphs, where each part has P nodes. The results are plotted in Figure 3(a) and 3(c), which shows that Merge I substantially outperforms Merge II. Values of upper bound $P + P(P-1)\lceil \log_p K \rceil$ are shown in Figure 3(b). The bounds for P=4 are determined by $cc(G_{K,4}) \leq 4 + 12\lceil \log_3 K \rceil$ discussed below Theorem 1. The lower bound from [4] is lower than the trivial P^2 bound in most of our cases, so we do not show it in the figure.

Though Merge I produces fewer cliques, it requires a longer running time. To estimate the difference in CPU requirements, we counted the numbers of previously described unit operations, so the impact of different number of cliques produced by the two methods is included. Although it is possible to improve the time efficiency of Merge I by designing better data structures, Merge I will always be slower than Merge II.

The heuristic method from [13] takes even more time than Merge I, so its running time analysis is not shown here. However, we plot its clique numbers. As shown in Figure 3(d), it produces more cliques than Merge II for any instances.

3.2 α-Globin Gene Cluster

A recent study carefully identified ortholgous genes in the α-globin clusters of a number of mammals [12]. There are four types of genes in those clusters: ζ, αD, α and θ. Each species discussed below has exactly one αD-related gene, so those genes are not pertinent to this analysis. The studied species have 1 to 3 ζ-related genes, 1 to 4 α-related genes, and 0 to 3 θ-related genes (counts include pseudogenes). Table 1 shows details. Each gene copy is regarded as a node in our graph.

Our earlier TBA program [2,15,16] guarantees that every position in the reference sequence (human in this case) is in exactly one multiple alignment block,

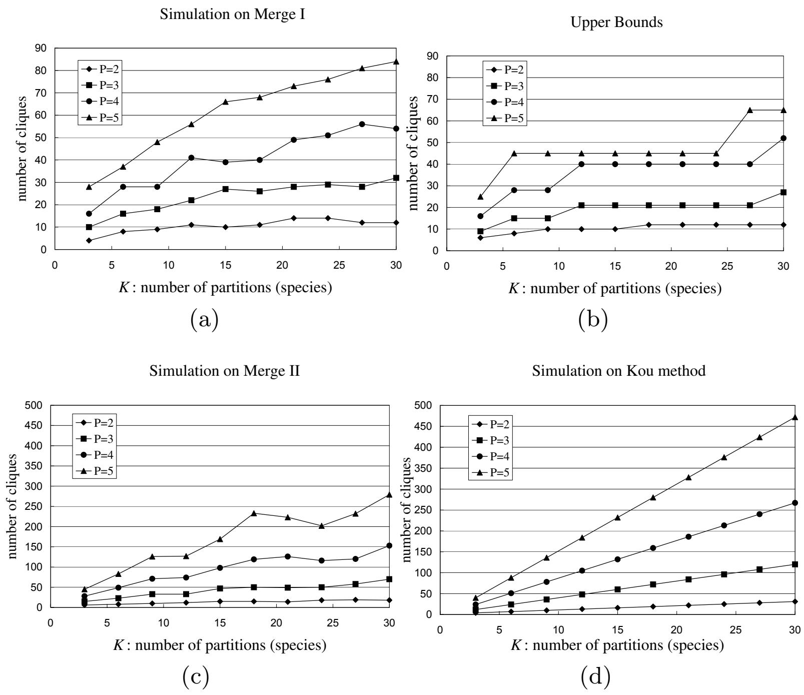

Fig. 3. Simulations of three heuristic procedures together with plots of upper bounds, with different values of K and P. The curves, top to bottom, refer to P=2,3,4,5.

and thus TBA is not able to capture all pairwise orthologous relationship in a tandem gene cluster. For example, each human α gene is aligned to only one rat α gene, despite the fact that a human α is actually orthologous to two more rat α genes, and those alignments are lost. Our new aligner, called BOAST, which implements Merge II, captures all pairwise orthologous relationship.

We aligned sequences containing the α-globin gene clusters from 20 mammals. Each sequence is around 200K bases. Both TBA and BOAST utilize pairwise alignments computed by blastz [20]. For BOAST, the pairwise alignments are filtered by a program called TOAST [11], which retains only the putatively orthologies (i.e., deletes paralogous matches). After computation of pairwise alignments, TBA produces 7.5 Mb of alignments after 170 CPU seconds, while BOAST produces 8.7 Mb of alignments in around 112 CPU seconds.

Each aligner outputs a set of blocks, whose endpoints do not in general correspond to gene or exon boundaries. Moreover, each functional globin gene has three exons, and many alignments do not extend from one exon to the next. We estimated how many cliques were formed for each type of genes as follows. For a given gene sequence, we manually determined three positions distributed roughly evenly throughout the gene, and counted the number of times each position appeared in the multi-alignment blocks; the maximum of the three counts was used to estimate the number of times the gene copy appears in the blocks.

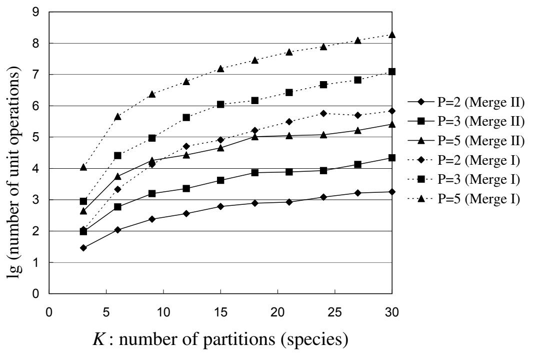

Fig. 4. Comparisons of running time on methods Merge I and II

In this example (α-globin clusters of 20 species), the clique sizes (i.e., the number of rows in a multi-alignment block) vary from 4 to 20, with most between 16 and 20. Alignment blocks containing θ-related genes have at most 17 rows since three species do not have a θ-related gene. Other blocks with less than 20 rows result from a number of factors, including inconsistent pairwise alignments, pseudogenes, or retention of a non-orthologous alignment. With many-to-many orthologous relationships, there will be combinatorial increase on the number of alignment blocks. Table 1 shows that utilizing Merge II, we reduce the number of alignment blocks to 26 for ζ-related alignments and 58 for α-related alignment. It means that 26 and 58 multiple alignment blocks contain all pairwise orthologous relationships of a certain region for ζ-related and α-related genes respectively.

The BOAST alignments are reference-independent, which means that no sequence data from any of the species is missing from the alignment. One of our tools extracts a reference-sequence-based alignment from the BOAST alignment for any specified reference. This gives an alignment similar in size to the output of a typical reference-based multiple aligners, for example multiz [2]. Moreover, the BOAST alignments capture complete and accurate orthology information, which is currently lost by other aligners.

Table 1. Number of copies for each type of α-globin genes of 20 mammals, and number of cliques formed for each type of genes in the multiple alignment employing the heuristic Merge II. When the number of genes is given in the form x-y, x refers to genes, and y refers to genes together with pseudogenes.

gene	armadillo	baboon	cat	chimp	colobus	cow	dog	dusktit	galago	hedgehog	human	lemur	macaca	marmoset	mouse	owlm	pig	rat	rfbat	sqmonkey	#cliques
ζ-related	1	2	2	2	2	2	2	1	2	2	1-2	1	2	1	1	1	1	1	1	2	26
α-related	1	1-2	1	1-3	2-4	2	1	2	2	2-3	2-3	2	2	1-2	2-3	2	1	3	1	2	58
θ-related	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1-3	1	1	3	0	1	7

4 Conclusion and Further Work

In gene clusters having a significant level of lineage-specific duplications (i.e., producing many-to-many orthology relationships), it is not practical to enumerate all possible multi-alignment blocks having pairwise orthologous rows. However, we have shown here that it is still frequently feasible to produce a set of blocks with the property that every pair of orthologs appears together in one of the blocks. The essence of the situation is captured by the problem of finding a minimum-cardinality clique cover. Both the problems of the finding the minimum number of cliques to cover all edges and the minimum number of cliques to cover all nodes (see below) are NP-complete. However, our simulations show that sizes can in practice be reduced (especially by Merge I) to be close to the upper bound. Though our alignment program has only incorporated Merge II, it still dramatically reduces the alignment size, and makes it feasible to align tandem gene clusters from many species. In the future, we hope to implement Merge I in our alignment program.

Though the clique cover problem for an arbitrary graph is NP-complete, it is open whether the problem's restriction to alignment graphs is intractable. The structure of the graphs is constrained by the phylogenetic tree for the species in question and by properties of the orthology relationship [11], and these restriction might be helpful for determining clique covers.

It also remains to investigate other criteria for aligning regions containing duplicated segments or genes. For instance, one could loosen the requirement that each orthologous pair of positions occur in two rows of the same block, and ask only that a position of one species that has an ortholog in a second species must appear in the same block as some (perhaps different orthologous) position in that second species. In essence, this can be modeled as seeking the minimum number of cliques to cover all *nodes* in the graph constructed above, which in general requires fewer cliques than the problem studied here. We think that the problem of aligning tandem gene clusters is sufficiently important that a variety of approaches should be investigated.

References

1. Berman, P.: Relationship between density and deterministic complexity of NP-complete languages. *Lecture Notes in Compute Science* **62**, 1978 63–71
2. Blanchette, M. et al.: Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research* **14**, 2004 708–715
3. Cacceta, L., P. Erdos, E.T. Ordman and N.J. Pullman: On the difference between clique numbers of a graph. *Ars Combinatoria* **19A**, 1985 97–106.
4. Cavers, M.: Clique partitions and coverings of graphs 2005 (Masters thesis, University of Waterloo)
5. Cooper, G. M., et al.: Distribution and intensity of constraint in mammalian genomic sequences. *Genome Research* **15**, 2005 901–913.
6. Fitch, W. M.: Distinguishing homologous from analogous proteins. *Syst. Zool.* **19**, 1970 99–113.

7. Fitch, W. M.: Homology, a personal view on some problems. *Trends Genet.* **16**, 2000 227–231.
8. Gramm, J., et al: Data reduction, exact, and heuristic algorithms for clique cover. *ALENEX* , 2006 86–94.
9. Gregory, D. A., and N.J. Pullman: On a clique covering problem of Orlin. *Discrete Math.* **41**, 1982 97–99.
10. Hall, M. Jr.: A problem in partition, Bull. *Amer. Math. Soc.* **47**, 1941 801–807.
11. Hou, M., et al: Aligning multiple genomic sequences that contain duplications. Manuscript.
12. Hughes, J. R., et al: Annotation of cis-regulatory elements by identification, subclassification, and functional assessment of multispecies conserved sequences. *Proc. Natl. Acad. Sci. USA* **102**. 2005 9830–9835
13. Kou, L.T., et al: Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Communications of the ACM* **21(2)**. 1978 135–139
14. Lund C. and M. Yannakakis, On the hardness of approximation minimization problems. *J. Assoc. for Comput. Mach.* **41**, 1994 961–981.
15. Margulies, E. H., et al. Relationship between evolutionary constraint and genome function in 1% of the human genome. Submitted to *Nature*.
16. Margulies, E. H., et al. Annotation of the human genome through comparisons of diverse mammalian sequences. Submitted to *Genome Research*.
17. Orlin, J. Contentment in graph theory: covering graphs with cliques. *Indag. Math.* **39**, 1977 406–424.
18. Pullman, N. J., and A. Donald: Clique coverings of graphs II: complements of cliques. *Utilitas Math.* **19**, 1981 207–213.
19. Pullman, N. J.: Clique coverings of graphs IV: algorithms. *SIAM J. on Computing* **13**, 1984 57–75.
20. Schwartz, S., et al. Human-Mouse Alignments with BLASTZ. *Genome Res.* **13(1)**, 2003 103–107.
21. Siepel, A., et al. Evolutionarily conserved elements in vertebrate, insect, worm, and yeast genomes. *Genome Research* **15**, 2005 1034–1050.
22. The ENCODE Project Consortium: The ENCODE (ENCyclopedia of DNA Elements) Project. *Science* **306**, 2004 636–640.
23. Wakefield, M. J., P. Maxwell and G. A. Huttley: Vestige: maximum likelihood phylogenetic footprinting. *BMC Bioinformatics* **6**, 2005 130.