COMPUTING HIGH-STRINGENCY COGS USING TURÁN-TYPE GRAPHS

CRAIG FALLS, BRADFORD POWELL, AND JACK SNOEYINK

Abstract. Clusters of Orthologous Groups (COGs) are a popular tool for identifying groups of genes in different genomes that may be derived from a common ancestor. In a graph whose edges represent mutually best-fit genes, one forms COGs of stringency m by merging m-cliques that overlap in $(m-1)$-cliques, and reporting the resulting sets of vertices. The presence of paralogs produces large, complete k-partite (Turán-type) graphs, which makes straightforward enumeration of cliques infeasible—there are simply too many. By recognizing and exploiting these extremal graphs, however, we obtain an algorithm to compute COGs that is orders of magnitude faster than clique enumeration.

1. Introduction

Clusters of Orthologous Groups (COGs) were described by Tatusov *et al.* [18] as a technique for comparing genomes of several species to identify groups of *orthologs*—genes in the different species that are related by evolution. COGs have proven a popular tool for comparative genome studies, including function prediction [19], phylogenetic classification [13, 18, 20], and ancestor studies [7, 10]. The online COG database is a useful, and growing, resource [19, 20].

The original clustering procedure was based upon finding overlapping sets of three mutually best-fit genes (BeTs), and was run on 7 genomes from 5 *clades*—families of related organisms. Specifically, starting from a 7-partite graph whose nodes are genes and edges represent related genes in different species, COGs were defined by finding all triangles (3-cliques), then merging triangles into groups if they shared an edge (2-clique).

As more genomes are added—comparisons of 20 to 50 genomes are common, using tens of thousands of genes—it is natural to ask for overlapping sets of m-cliques, for $m > 3$, and merging groups that share an $(m-1)$-clique to determine highly-conserved orthologs. Montague and Hutchinson [13] therefore gave this as a definition of *COGs of stringency* m; we say more about this definition in the next section.

The difficulty of enumerating cliques in a graph of related genes grows exponentially as m grows, making straightforward computation of COGs exponentially more difficult as the number of genomes grows. The number of groups remains tractable, however. In fact, we observe that the graphs created by genome search can be described by relatively few complete k-partite subgraphs, reminiscent of the Turán graphs [22] from extremal graph theory. By recognizing maximal Turán-type subgraphs, we are able to compute COGs much more efficiently than by brute force.

2. Definitions and Preliminaries

We rephrase the computation of COGs [18] in the language of graph theory [5].

Given a set of sequenced genomes, tools such as BLAST [2] can measure the similarity between pairs of genes. A gene g_1 in species s_1 will have some gene g_2 as its "best-hit" in species s_2—the gene in s_2 with the best similarity score. In the case of ties, it is possible for a gene to have more

Key words and phrases. COG, clique, Turán, genome, stringency, ortholog, paralog, algorithm

Address of corresponding author: Craig Falls, CB 3175 Sitterson Hall, Chapel Hill, NC 27516 USA, cfalls@cs.unc.edu.

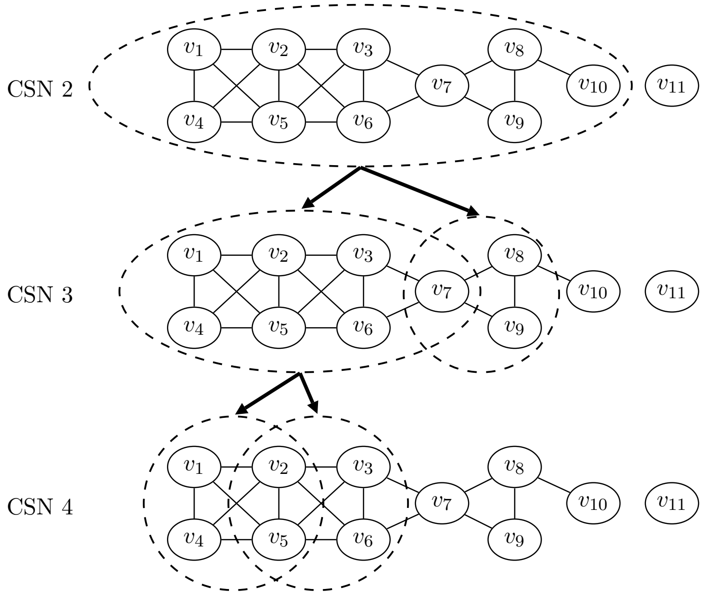

FIGURE 2.1. Increasing COG stringency number (CSN) forms a tree structure of COG splits. The entire set of vertices (genes) to be a COG of stringency 1. Isolated vertices (v_{11}) drop out and connected components ($v_1, \ldots, v_{10}$) form the COGs of stringency 2. Each component splits into COGs of stringency 3 by finding the clusters of triangles that share edges. Articulation points (v_7) are repeated in the split, and vertices (v_{10}) in no triangles disappear. The tetrahedra $\{v_1, v_2, v_4, v_5\}$ and $\{v_2, v_3, v_5, v_6\}$ are each COGs of stringency 4; they do not merge because they do not share a triangle.

than one best hit. If this relationship is mutual—i.e., g_1 is a best-hit for g_2 and vice versa—then $g_1$and g_2 are called BeTs [18].

We can form a graph whose nodes are genes and edges are mutual best-hit pairs. Because genes aren't compared within their own genomes, the graph formed from k genomes is k-partite—its vertex set is a union of k pairwise-disjoint sets (the species) $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that each edge (u, v) satisfies $u \in \mathcal{S}_i$ and $v \in \mathcal{S}_j$ with $i \neq j$. Tatusov *et al.* [18] use the mutual best-hit graph to define COGs. One could imagine other types of graphs such as mutual-best-in-5 hits.

A COG of *stringency m,* defined by Montague and Hutchinson [13], is a maximal m-tree in graph theory terminology [9]. In other words, all COGs of stringency m could be created by placing each m-clique in its own group, then merging two groups whenever two m-cliques, one in each group, intersect in a common $(m-1)$-clique.

As the COG stringency number m increases, COGs will lose vertices and split, as illustrated in FIGURE 2.1. Thus, COGs of decreasing stringency form a hierarchy that reveals the strength of the relationship and justifies what was already observed in the original COG papers—that some COGs computed at stringency 3 need to be split to separate functional groups. FIGURE 4.1 is an example of some COG splits resulting from 20 bacterial genomes.

Examination of high-stringency COGs could have several biological applications. Increased stringency implies a greater degree of conservation of a gene among organisms. Such genes may be more likely to be essential than genes that are not in COGs or are only in less stringent COGs. Highly conserved sets of genes may form a core of genes that are resistant to lateral gene transfer (LGT) and thus may be useful for determining phylogeny [11].

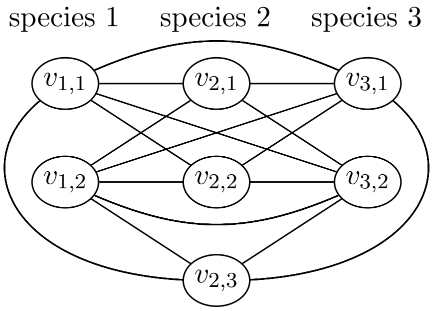

FIGURE 2.2. The subgraph $K_{2,3,2}$ above could arise in a mutual best-hit graph if three species each inherit either two or three paralogs from a common ancestor. Vertices $v_{*,j}$ represent genes that are orthologs, and $v_{i,*}$ represent paralogs. This single *Turán graph* has 12 maximal cliques.

The straightforward way to compute COGs of stringency m is to enumerate all cliques up to size 3, 4, …, m, since we can form m-cliques from $(m-1)$-cliques. Then, a simple connected components computation where vertices are cliques of size k and edges are cliques of size $k-1$ shared between cliques of size k will build the COGs at level k. Unfortunately, large cliques contain many smaller cliques, and even finding the largest clique in a graph is NP-hard [8].

An obvious improvement is to enumerate only the *maximal* cliques—those that are not a subset of any other clique. The smaller internal cliques are redundant; the connected components can be found without explicitly enumerating them. Unfortunately, even this is not sufficient in the presence of *paralogs*—two or more genes in a single organism that share a common gene-ancestor either through gene duplication within an organism or through lateral gene transfer between species. Paralogs naturally cause 'ties' in the best-hit graph, and lead to large multi-partite subgraphs, such as Figure 2.2, which contain too many maximal cliques to enumerate. In one of our examples with 20 species, one complete 19-partite subgraph had more than 10^{10} maximal 19-cliques.

A *complete k-partite graph* $K_{n_1,\ldots,n_k}$ is a graph with $\sum_i n_i$ vertices partitioned into k sets, and all $\sum_{i<j} n_i n_j$ possible edges between sets. FIGURE 2.2 provides an example. Turán's theorem [22] characterizes graphs with no $(k+1)$-clique in terms of such graphs. To avoid constant reference to complete multi-partite subgraphs where each partition comes from a species, we call these *Turán graphs.* Abusing the terminology slightly, we do not insist that the number of genes from each species be nearly equal. A Turán graph is *maximal* if it is not a subset of any other Turán graph.

3. COG ALGORITHM

Once we recognize that maximal Turán graphs are an obstruction to enumerating maximal cliques, we can use them to help find COGs—since an entire Turán graph is in a single COG, its individual cliques need not be enumerated. Although a graph can have an exponential number of Turán graphs, we have found that best-hit pairs tend to produce relatively few maximal Turán graphs, as shown in TABLE 1.

The algorithm to compute COGs of stringency m runs in three phases. The first enumerates all maximal Turán graphs in the mutual best-hit graph G. The second finds the largest cliques connecting pairs of Turán graphs, which is just the number of species in which the two Turán graphs share vertices. The third creates the graph $G_{\mathcal{T}}$ defined by a vertex of weight m for each Turán graph containing m-cliques and an edge of weight w between two vertices if their Turán graphs have w-cliques in their common intersection. A depth-first search on the subgraph of $G_{\mathcal{T}}$ defined by the vertices of weight at least m and edges of weight at least $m-1$ partitions the Turán graphs into connected components. A COG of stringency m is simply the union of the vertices of the maximal Turán graphs of G that lie in one connected component of $G_{\mathcal{T}}$. The rest of this section elaborates upon these phases. The pseudocode for the algorithm and proven bounds on complexity are in the Appendix.

3.1. **Phase 1: Find all maximal Turán graphs.** This phase enumerates all the maximal Turán graphs in a mutual best-hit pair graph G on the species $\mathcal{S}$. Specifically, when Phase 1 completes, $\mathcal{M}[v]$ will store, for each vertex $v \in V[G]$, all the maximal Turán graphs containing v.

Phase 1 can be implemented by adapting maximal clique enumeration algorithms (section 3.1.1) or by a new branch-and-bound heuristic (section 3.1.2). The former has the advantage of output-sensitive bounds on complexity, but the disadvantage of requiring more space and time. The latter performs well in practice, as shown in Results, but lacks good bounds on worst-case behavior.

For either method of enumerating the maximal Turán graphs, it helps to filter the input graph some first. In particular, since we are only interested in COGs of stringency 3 and higher, any vertices and edges that do not participate in a 3-clique can be removed. More substantially, some paralogs can be removed. If two vertices v_1 and v_2 have the same set of neighbors, they will be in the same COG at all stringencies. The vertex v_2 can be removed and then placed back into any COGs containing v_1 in the output. These simple preprocessing steps remove 10%–20% of the vertices in our best-hit graphs and reduce the runtime by over an order of magnitude by diminishing the large Turán graphs.

3.1.1. *Using maximal clique enumeration to enumerate maximal Turán graphs.* Finding maximal Turán graphs is equivalent to finding maximal cliques in a graph in which an edge has been added between every pair of vertices in the same species. This makes all maximal Turán graphs into maximal cliques and adds only the maximal cliques within each species. Unfortunately, it also adds n^2/s edges to the graph G, where there were typically $\mathcal{O}(n)$ edges, with a small constant.

After adding edges, any algorithm for enumerating maximal cliques [1, 3, 4, 6, 12, 16, 17, 21] can then be used. The algorithms of Bierstone (noted by Augstson-Minker [3] and corrected by Mulligan-Corneil [15]) and Tsukiyama [21] run in $\mathcal{O}(\mu^2)$ and $\mathcal{O}(nm\mu)$ time, where $n = |V|$, $m = |E|$, and μ is the number of maximal cliques.

The Bron-Kerbosch [6] algorithm is empirically efficient but lacks a theoretical estimation of complexity. We experimented with the Bron-Kerbosch algorithm, but the $\Theta(n^2)$ storage for edges quickly becomes impractical for large genomes, even after eliminating low degree vertices and partitioning G into connected components to reduce n.

3.1.2. *Branch and bound to enumerate maximal Turán graphs.* Maximal-Turans is a branch and bound algorithm that finds maximal Turán graphs directly. In the pseudo-code for Algorithm 1, *vertex* is the vertex currently being added, *cur-turan* is the set of vertices that are currently being considered, and *feasible* is the set of vertices that can be added to *cur-turan*, maintaining the property that *cur-turan* is a Turán graph. Specifically, for every vertex $v_c \in$ *cur-turan* and vertex $v_f \in$ *feasible*, either $(v_c, v_f) \in E$ or *species* $[v_c] =$ *species* $[v_f]$. Vertices, species, and maximal Turán graphs are assigned unique integer representations. Sets of vertices are stored in Patricia trees [14] so that overlapping sets may share address space.

Finding all maximal Turán graphs is the most computationally expensive phase of the algorithm. We use three methods to bound the search space. First, we consider only sets of vertices that form Turán graphs. This is easily accomplished by adding to *cur-turan* only vertices from the set *feasible* (Branch line 3).

Second, we ensure that the algorithm never visits the same set of vertices more than once. One easy way to ensure this would be to consider vertices ordered and edges directed; we prefer a more involved way of reducing the feasible set of vertices. Each node of the recursion subtracts visited vertices (Branch line 13) from the feasible set and passes this reduced feasible set to its children. By not considering edges directed, we have more flexibility in the order we visit vertices, which is exploited by our third method of reducing the search space.

Third, we avoid searching subsets of maximal Turán graphs that have already been found (Branch line 5). If *cur-turan* is a subset of some known maximal Turán graph $\mathcal{T}$, then we may safely consider only vertices from outside of $\mathcal{T}$ at the current level of recursion. To accomplish this, we maintain the set *max-turans* of known maximal Turán graphs that contain *cur-turan*. Since $\mathit{max\text{-}turans} = \bigcap_{v \in \mathit{cur\text{-}turan}} \mathcal{M}[v]$ and only one vertex v is added to *cur-turan* at a time, the set *max-turans* can be maintained at a cost of one set intersection per branch-and-bound node (Bound line 2).

The properties of the best-hit graphs make this an effective algorithm. In a best-hit graph, a vertex connects to at most one other vertex per species, unless there are ties (usually caused by paralogs). Since each new vertex is chosen to break out of a known maximal Turán graph, the only way to explore unproductive branches of the tree is for several large Turán graphs to overlap significantly. We can cause such overlaps by deleting j independent edges from a large Turán graph—this creates 2^j maximal Turán graphs, by chosing one of the two endpoints of each of the j deleted edges. These choices are not made independently when computing best-hit graphs, however, so the overlap remains small. We would like to give a non-trivial analysis of time and space requirements for the algorithm on a model that reflects the best-hit graphs observed in practice, but have not been able to do so.

3.2. Phase 2: Find largest clique shared between maximal Turán graphs.

Our algorithm named Shared-Species-Build-Hashtable finds the largest cliques connecting pairs of Turán graphs, which is just the number of species in which the two Turán graphs share vertices. Note that the number of species in which two Turán graphs share vertices is the size of the largest clique shared between them, which is also the level at which all their vertices will be merged into a common COG. The remainder of the algorithm hinges on this observation.

The Shared-Species-Build-Graph algorithm replaces bit vectors that mark species with appropriately weighted edges. The weight of each edge is exactly the size of the largest clique shared between the two maximal Turán graphs, which is determined by counting the number of species in which the Turán graphs share a vertex.

3.3. Phase 3: Find connected components.

Output-Maximal-Turans performs a depth-first search through the maximal Turán graphs to create the connected components. A COG is then simply the union of the vertices of all maximal Turán graphs in one connected component. The function Depth-First-Search $(G_{\mathcal{T}}, i, j)$ performs a depth-first search of the vertex of weight at least i and edges of weight at least j in the graph $G_{\mathcal{T}}$ and reports the connected components. The vertices of one connected component of $G_{\mathcal{T}}$ correspond to the maximal Turán graphs of G that are in one COG. The time spent in Phase 3 is a negligible proportion of COG computation.

4. Experimental Results

We formed the mutual best-hit graph Small from all of the annotated gene products (53,210) in 20 bacterial genomes. The species included members of Gram positive, mollicute, Gram negative and spirochete groups. Each of the gene products were compared to each other using BLASTP with a `blosum62` similarity matrix. The comparisons resulted in 159,081 mutual best-hit pairs, representing a total of 46,619 gene products. Figure 4.1 shows a sampling of eight COGs at stringency 3 that split at least twice; Figure 4.2 maps these eight COGs onto a phylogenetic tree of these genomes.

Computing the COGs of stringency 3 through 20 in the mutual best-hit graph Small required 3 hours and 58 minutes by straightforward clique enumeration, but only 6 seconds by Turán graph enumeration. We were unwillling to wait for clique enumeration to complete any of the larger examples, which are graphs created from open reading frames (ORFs) of the same species. Time for Turán graph enumeration remains small, as listed in Table 1. Our collaborators have asked us

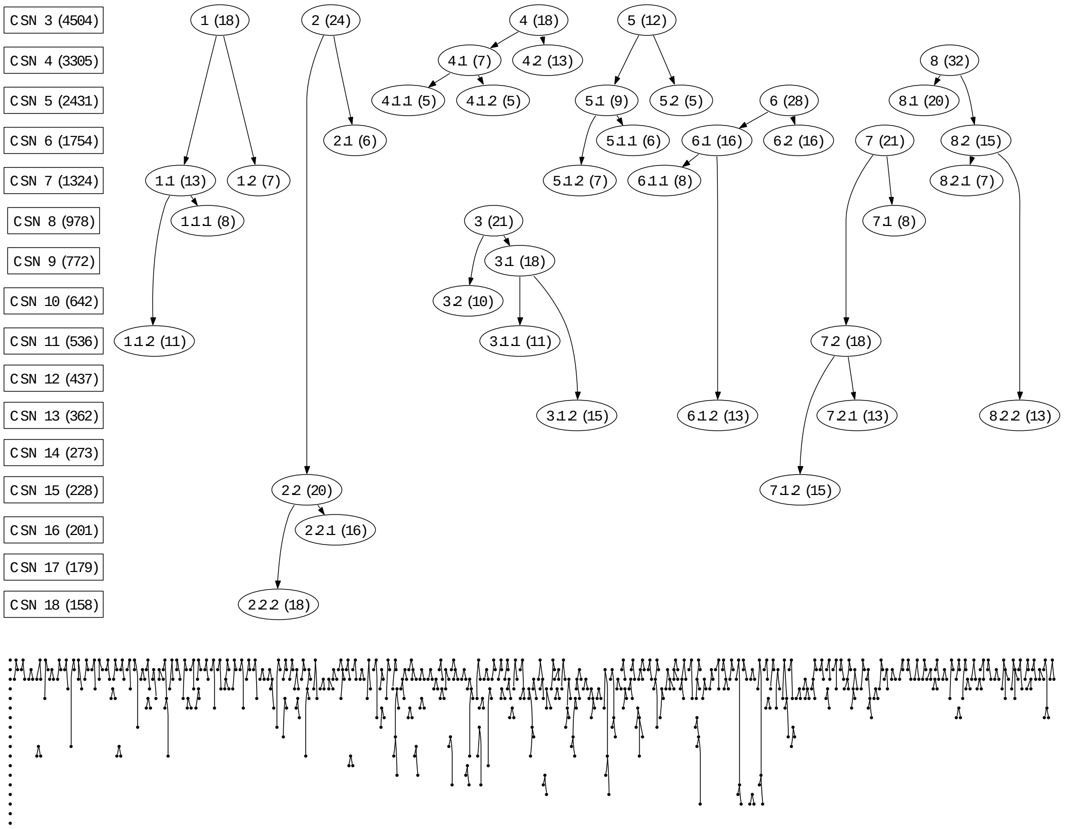

Figure 4.1. A sampling of the richness of COG structure as stringency varies for dataset Small, containing all annotated gene products in 20 bacterial genomes. Each box records the number of COGs of the indicated stringency. Each oval is a COG at its greatest stringency, labeled with hierarchy numbers used in Figure 4.2 and number of genes present. Edges indicate merges. Note that a COG at stringency k is within a COG of all smaller stringencies, and that two COGs that merge at k share at least $k - 1$ genes. The trees below show all COGs that split at stringency greater than 4; each tree appears as only a single COG at stringency 3.

not to release the identity of the ORFs until the paper is published. All results were obtained on an AMD Athlon XP 2400 with 512MB RAM running Linux with the Maximal-Turans heuristic.

We attempted to apply the Bron-Kerbosch [6] maximal clique enumeration algorithm to obtain timing on larger examples by adding intra-species edges, as described in section 3.1.1. Doing so directly fails on even moderate data sets because the adjacency matrix already takes quadratic space. We iteratively removed all vertices of degree 3 or less, then separated the graph into connected components. Within each component, an edge was added between every pair of vertices in the same species, as described in section 3.1.1, if the number of maximal cliques far exceeded the number of maximal Turán graphs. Finally, the Bron-Kerbosch algorithm was run on each component individually. Unfortunately, one of the resulting 517 connected components of Small still had 28,620 vertices, which would require 781MB of RAM and lead to thrashing on our test machine. Thus, memory consumption remained a barrier for all mutual best-hit graphs except Tiny, whose Turán graphs can be enumerated in under 1 second by the Bron-Kerbosch algorithm since Tiny can be divided into 1,320 connected components, the largest of which has only 43 vertices and requires only 1.8kb to store.

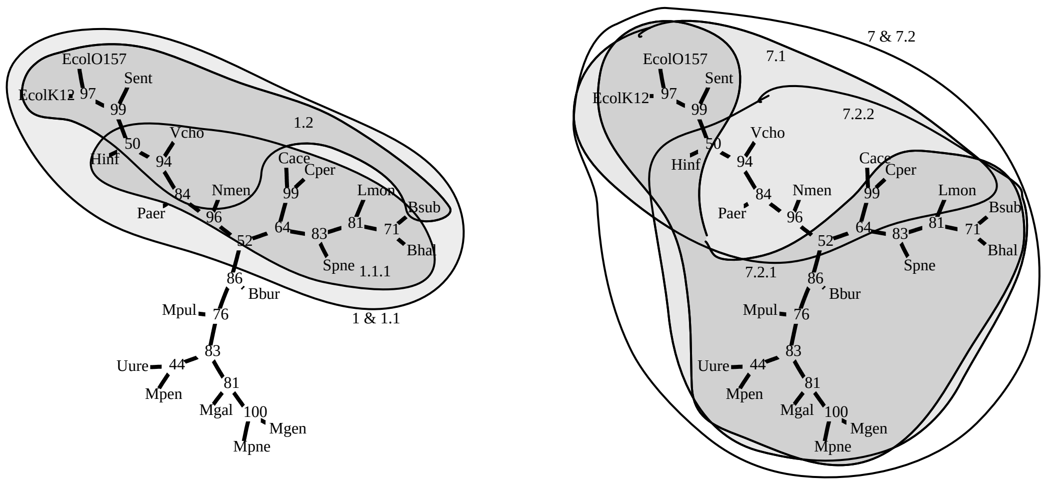

FIGURE 4.2. Two of the COGs hierarchies from FIGURE 4.1 imposed on a phylogenetic tree of the 20 genomes of SMALL.

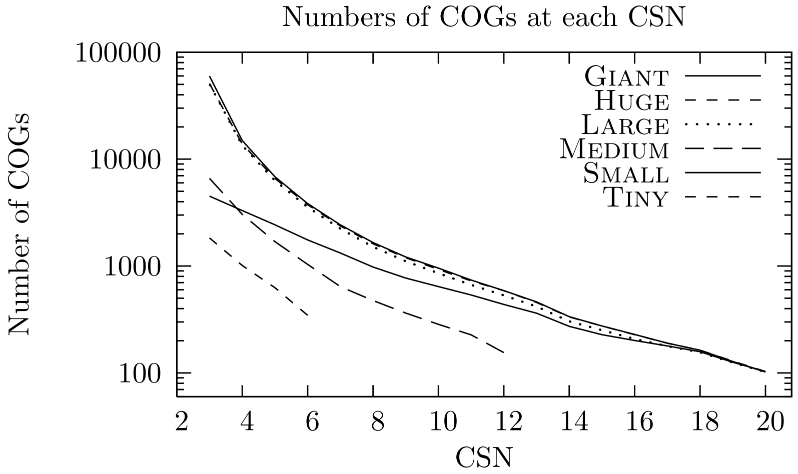

FIGURE 4.3. This logarithmic plot of the number of COGs at stringency (CSN) levels 3 through 20 in 4 mutual best-hit graphs shows that, while some COGs of lower stringencies split to form multiple COGs of higher stringencies, most simply drop out as the stringency is raised; both phenomena can be seen in FIGURE 4.1. All the data sets except SMALL stay in the same relative order as the CSN is increased. SMALL is unique because it contains only annotated gene products, while the other data sets contain a type of proto-gene called *open reading frames* (ORFs). Biologists are more likely to study highly conserved orthologos, so the genes involved in high stringency COGs are more likely to be annotated.

MAXIMAL-TURANS depends on the presence of relatively few maximal Turán graphs in mutual best-hit graphs. To test the robustness of this algorithm, we modified the mutual best-hit graph SMALL by adding or removing 2.5%, 5%, or 10% of the edges at random. The number of Turán graphs increases, as expected, but the running times are acceptable.

	Vertices	**Edges**	**Cliques**	**Turáns**	**Runtime**
Tiny	35,354	36,183	4.08×10^3	3,495	1 sec
Small	46,619	159,081	1.77×10^4	15,877	6 sec
Medium	182,676	297,551	7.64×10^6	20,666	7 sec
Large	561,249	900,056	1.43×10^5	125,052	29 sec
Huge	575,435	1,171,655	3.59×10^{11}	132,144	42 sec
Giant	752,688	1,729,907	3.59×10^{11}	200,107	61 sec

Table 1. By varying the settings for BLAST on 20 bacterial genomes, we produced 6 mutual best-hit graphs with varying numbers of edges (pairs of best-fit genes) and vertices (gene products represented in a best-fit pair). We list the number of maximal cliques and maximal Turán graphs for comparison. For the larger mutual best-hit graphs, there are 5 orders of magnitude fewer maximal Turán graphs than maximal cliques—in fact, there are fewer maximal Turán graphs than edges.

	Edges	**Cliques**	**Turáns**	**Runtime**
Original (Small)	159,081	17,736	15,877	6 sec
Edges Added	+ 2.5%	24,060	20,312	10 sec
	+ 5.0%	28,951	23,400	11 sec
	+10.0%	39,077	29,643	14 sec
Edges Removed	− 2.5%	23,856	21,592	11 sec
	− 5.0%	31,444	28,745	18 sec
	−10.0%	43,141	40,620	41 sec

Table 2. Testing robustness under adding or deleting edges from the mutual best-hit graph Small.

5. Conclusions

We have described an algorithm for computing COGs of stringency m, as defined by Montague and Hutchinson [13]: namely that m-cliques of potential orthologs are identified, and merged as long as they overlap in $m - 1$ vertices. They did this to find highly-conserved orthologs in the herpesvirus, and also to avoid the merger of COGs through chance sequence similarity, which is important in comparing several small genomes.

The problem of finding stringent COGs is NP-hard, so one cannot expect an efficient algorithm for finding COGs in the general case. By exploiting the nature of the data sets we are interested in, however, we obtain an algorithm which is fast enough in practice. Specifically, the feature of mutual best-hit graphs our algorithm relies on is the presence of relatively few Turán graphs. The result is an algorithm that is several orders of magnitude faster than a brute-force approach.

Since the time- and memory-consuming phase of our algorithm is enumerating the maximal Turán graphs, we can support any specified overlap number between 2 and $m - 1$, inclusive. Increasing clique size reduces the set of COGs to those that are highly conserved orthologs. Increasing the overlap number makes it harder for COGs to merge accidentally. Further research is needed to explore the best combination of these values.

The source code, mutual best-hit graphs, and COG output for our algorithm are available at `http://www.cs.unc.edu/~cfalls/cogs.html`.

Acknowledgement. The first and last authors gratefully acknowledge support from NSF through grant 0076984. The second acknowledges support through a subcontract to Clyde A. Hutchison III at UNC Chapel Hill from the Berkeley Structural Genomics Center (GM62412, Sung-Ho Kim P.I.)

References

[1] E. A. Akkoyunlu. The enumeration of maximal cliques of large graphs. *Siam J. Comput.*, pages 1–6, 1973.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol. 215:403-410.*, 1990.

[3] J. G. Augustson and J. Minker. An analysis of some graph theoretical cluster techniques. *J. Assoc. Comput. Mach.*, pages 571–588, 1970.

[4] C. Berge. *The Theory of Graphs and Its Applications.* John Wiley, 1962.

[5] Bela Bollobos. *Modern Graph Theory.* Springer Verlag, 1998.

[6] Coen Bron and Joep Kerbosch. Finding all cliques of an undirected graph – algorithm 457. *Comm. ACM*, pages 575–577, 1973.

[7] D. J. Brooks, J. R. Fresco, A. M. Lesk, and M. Singh. Evolution of amino acid frequencies in proteins over deep time: Inferred order of introduction of amino acids into the genetic code. *Mol. Bio. Evolution*, 19(10):1645–1655, October 2002.

[8] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[9] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions: A survey. In *Proc. of MFCS*, volume 2136 of *Lecture Notes in Computer Science*, pages 37–57. Springer, 2001.

[10] J. K. Harris, S. T. Kelley, G. B. Spiegelman, and N. R. Pace. The genetic core of the universal ancestor. *Genomics Res.*, 13(3):407–412, March 2003.

[11] E. Lerat, V. Daubin, and N. A. Moran. From gene trees to organismal phylogeny in prokaryotes: the case of the gamma-proteobacteria. *PLoS Biol 1*, pages 1, E19, 2003.

[12] P. M. Marcus. Derivation of maximal compatibles using boolean algebra. *IBM J. Res. Develop.*, pages 537–538, 1964.

[13] Michael G. Montague and Clyde A. Hutchison III. Gene content phylogeny of herpesviruses. *PNAS*, 97(10):5334–5339, May 2000.

[14] D. Morrison. PATRICIA – practical algorithm to retrieve information coded in alphanumeric. *Journal of ACM*, 15(4):514–534, October 1968.

[15] G. D. Mulligan and D. G. Corneil. Corrections to bierstone's algorithm for generating cliques. *J. Assoc. Comput. Mach.*, pages 244–247, 1972.

[16] R. E. Osteen. Clique detection algorithms based on line addition and line removal. *Siam J. Appl. Math.*, pages 126–135, 1974.

[17] M. C. Paull and S. H. Unger. Minimizing the number of states in incompletely specified sequential switching functions. *IRE Trans. Electronic Computers*, pages 356–367, 1959.

[18] RL Tatusov, EV Koonin, and DJ Lipman. A genomic perspective on protein families. *Science*, 278:631–637, 1997.

[19] Roman L. Tatusov, Michael Y. Galperin, Darren A. Natale, and Eugene V. Koonin. The COG database: a tool for genome-scale analysis of protein functions and evolution. *Nucleic Acids Research*, 28(1):33–36, 2000.

[20] Roman L. Tatusov, Darren A. Natale, Igor V. Garkavtsev, Tatiana A. Tatusova, Uma T. Shankavaram, Bachoti S. Rao, Boris Kiryutin, Michael Y. Galperin, Natalie D. Fedorova, and Eugene V. Koonin. The COG database: new developments in phylogenetic classification of proteins from complete genomes. *Nucleic Acids Research*, 29(1):32–28, 2001.

[21] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *Siam J. Comput.*, 6(3), September 1977.

[22] P. Turán. On an extremal problem in graph theory (in Hungarian). *Mat. Fiz. Lapok*, 48:436–452, 1941.

Appendix

In this appendix, we include the pseudo-code for our procedures, and a sketch of the running time analysis.

Theorem 5.1. *Given a graph $G = (V, E)$ containing the set of maximal Turán graphs Γ, phase 2 runs in $\mathcal{O}\left(n\mu + \mu^2\right)$ time, where $n = |V|$ and $\mu = |\Gamma|$.*

Proof. We first analyze Shared-Species-Build-Hashtable, then Shared-Species-Build-Graph.

Lines 4 through 9 of Shared-Species-Build-Hashtable execute in constant time, assuming the hashtable is properly implemented and that bit vectors can be initialized in constant time. The loops on lines 2 and 3 will cause an execution of the body of the loop for every pair $(v \in V,\, m \in \Gamma)$ such that $v \in m$. There cannot be more than $\mathcal{O}(n\mu)$ such pairs.

Lines 2, 4, and 5 of Shared-Species-Build-Graph execute in constant time, but line 3 takes $\mathcal{O}(s)$ time, where $s = |\mathcal{S}|$—the length of the bit vector. The loop on line 1 will cause an execution of the body of the loop for every pair $(m_1 \in \Gamma,\, m_2 \in \Gamma)$ such that some vertex $v \in m_1 \bigcap m_2$. There are $\mathcal{O}\left(\mu^2\right)$ such pairs. □

Theorem 5.2. *If G has n vertices in $s = |\mathcal{S}|$ species and μ maximal Turán graphs, phase 3 executes in $\mathcal{O}\left(s \cdot \left(\mu^2 + n\mu\right)\right)$ time.*

Proof. Line 1 of Output-Maximal-Turans loops s times. Since G has μ maximal Turán graphs, $G_{\mathcal{T}}$ has μ vertices. Therefore, the call to Depth-First-Search on line 2 searches a graph with μ vertices and at most $\binom{\mu}{2} < \mu^2$ edges, and will run in $\mathcal{O}\left(\mu^2 + \mu\right) \subseteq \mathcal{O}\left(\mu^2\right)$ time. Lines 3 and 4 report the COGs, which requires time proportional to the sum of the sizes of the COGs. At any one stringency level, there are at most μ COGs, since each maximal Turán graph is in at most one COG. Each COG consists of at most n vertices, so lines 3 and 4 run in $\mathcal{O}(n\mu)$ time. All together, Output-Maximal-Turans runs in $\mathcal{O}\left(s \cdot \left(\mu^2 + n\mu\right)\right)$ time. □

We can summarize the worst-case complexity in terms of the numbers of vertices n, species s, and maximal Turán graphs μ, but the practical results of the next section are more important.

Conclusion 5.3. COGs of all stringencies can be computed in $\mathcal{O}\left(n^3\mu + \mu^2\right)$ or $\mathcal{O}\left(s \cdot (n + \mu)^2\right)$ time.

Algorithm 1 MAXIMAL-TURANS

Input: An undirected graph $G = (V, E)$ in which vertices represent genes and edges represent best-hit pairs and the species *species* $[v] \in \mathcal{S}$ of each gene $v \in V$.

Output: When MAXIMAL-TURANS completes, $\mathcal{M}[v]$ will store, for each vertex $v \in V[G]$, all the maximal Turán graphs containing v.

```
MAXIMAL-TURANS($G$)
for each edge $(v_1, v_2) \in E[G]$ ▷ Grow Turán graphs from each edge.
    do max-turans ← turans-of-vertex[$v_1$] $\bigcap$ turans-of-vertex[$v_2$]
       feasible ← Adj[$v_1$] $\bigcap$ Adj[$v_2$]
       cur-turan ← $(v_1, v_2)$
       BRANCH(feasible, max-turans, cur-turan)

BRANCH(feasible, max-turans, cur-turan)
vertices-to-avoid ← choose a vertex set (Turán graph) from max-turans
new-feasible ← feasible
for each $v \in$ feasible ▷ Try all feasible vertices.
    do if $v \in$ vertices-to-avoid
          then continue ▷ Skip vertices in turan-to-exit.
       results ← BOUND($v$, new-feasible, max-turans, cur-turan)
       if results $\neq \emptyset$ and vertices-to-avoid $= \emptyset$
          then vertices-to-avoid ← choose a vertex set from results
       max-turans ← max-turans $\bigcap$ results
       new-feasible ← new-feasible $-v$ ▷ $v$ is no longer feasible.
       return max-turans

BOUND(vertex, feasible, max-turans, cur-turan)
max-turans ← max-turans $\bigcap$ $\mathcal{M}$[vertex]
cur-turan ← vertex $\bigcup$ cur-turan
feasible ← feasible $\bigcap$ Adj[vertex]
if feasible $= \emptyset$ ▷ Is this a dead-end?
   then if max-turans $= \emptyset$ ▷ Is this a maximal dead-end?
           then for each vertex $\in$ cur-turan ▷ Store maximal Turán graph
                     do $\mathcal{M}$[vertex] ← $\mathcal{M}$[vertex] $\bigcup$ cur-turan
                return max-turan
           else return $\emptyset$ ▷ Non-maximal dead-end—just backtrack
   else BRANCH(feasible, max-turans, cur-turan)
```

Algorithm 2 SHARED-SPECIES-BUILD-HASHTABLE

Input: For each vertex $v \in V[G]$, the set of maximal Turán graphs $\mathcal{M}[v]$ that contain v.

Output: Reports, for each pair of maximal Turán graphs, the species in which they share vertices.

Data Structures: A hashtable whose keys are pairs of maximal Turán graphs and values are bit vectors with bits corresponding to species.

```
SHARED-SPECIES-BUILD-HASHTABLE($\mathcal{M}$)
hashtable ← HASHTABLE-CREATE()
for each vertex $v \in V[G]$
    do for each pair $(mt_1, mt_2)$ of maximal Turán graphs $\in \mathcal{M}[v]$
        do key ← $(mt_1, mt_2)$
           if HASHTABLE-HASKEY(hashtable,key)
              then bitv ← HASHTABLE-GET(hashtable,key)
              else bitv ← bit vector of length $|\mathcal{S}|$ with all 0s
           bitv[SPECIES($v$)] ← 1
           HASHTABLE-REPLACE(hashtable,key,bitv)
return hashtable
```

Algorithm 3 SHARED-SPECIES-BUILD-GRAPH

Input: A hashtable storing, for each pair of maximal Turán graphs, the set of species in which they share vertices.

Output: An undirected graph $G_{\mathcal{T}}$ with weighted vertices and edges. A vertex of weight w corresponds to a maximal Turán graph spanning w species (i.e. the Turán graph consists of maximal w-cliques). An edge with weight w indicates that the two Turán graphs share vertices in w species (i.e. their intersection consists of w-cliques).

Data Structures: The graph is stored as an adjacency list.

```
SHARED-SPECIES-BUILD-GRAPH(hashtable)
for each $((mt_1, mt_2) \mapsto bitv) \in hashtable$ ▷ meaning: <key> ↦ <value>
    do edge ← $(mt_1, mt_2)$
       edge-weight ← number of 1s in bitv
       $E \leftarrow (edge, edge\text{-}weight) \bigcup E$
       $V \leftarrow \{mt_1, mt_2\} \bigcup V$
return $G_{\mathcal{T}} = (V, E)$
```

Algorithm 4 OUTPUT-MAXIMAL-TURANS

Input: An undirected graph with weighted vertices and edges—the output of SHARED-SPECIES-BUILD-GRAPH.

Output: Let $\mathcal{S}$ be the set of species and $s = |\mathcal{S}|$. For each level, $3 < m < s$, the collection of sets of vertices that represent the COGs at level m.

```
OUTPUT-MAXIMAL-TURANS($G_{\mathcal{T}}$)
for each stringency $m \in 3 \ldots s$
    do connected-components ← DEPTH-FIRST-SEARCH($G_{\mathcal{T}}, m, m - 1$)
       for each component ∈ connected-components
           do Report component as a COG of stringency $m$
```

						Ecol														
	B	B	B	C	C	K	O	H	L	M	M	M	N	P	S	S	U	V	M	M
	h	s	b	a	p	1	1	i	m	g	p	p	m	a	e	p	u	c	p	g
	a	u	u	c	e	2	5	n	o	e	n	u	e	e	n	n	r	h	e	a
	l	b	r	e	r		7	n	n	n	e	l	n	r	t	e	e	o	n	l
1.2		x				x	x						x	x	x			x		
1	x	x		x	x	x	x	x	x				x	x	x	x		x		
1.1	x	x		x	x	x	x	x	x				x	x	x	x		x		
1.1.1	x			x	x			x	x				x		x			x		
1.1.2	x	x		x	x	x	x	x	x					x	x	x				
2.1	x			x				x						x				x		
2	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2.2	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2.2.1	x		x		x	x	x	x	x	x	x	x	x		x	x	x		x	x
2.2.2		x	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x
3.2				x	x	x	x	x	x				x	x	x			x		
3	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x
3.1	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x
3.1.1	x	x				x	x			x	x	x			x	x			x	x
3.1.2	x	x		x	x		x	x	x	x	x	x	x	x		x		x	x	x
4.2	x	x		x	x	x	x	x					x	x	x	x		x		
4	x	x		x	x	x	x	x	x				x	x	x	x		x	x	
4.1				x				x	x				x	x				x	x	
4.1.1				x				x					x	x				x		
4.1.2								x	x				x					x	x	
5.2		x				x	x							x						
5	x	x				x	x						x	x	x	x				
5.1	x	x				x	x						x	x	x	x				
5.1.1	x	x											x		x	x				
5.1.2	x					x	x						x	x	x	x				
6.2	x	x	x	x	x			x	x	x	x	x	x	x		x	x		x	x
6	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
6.1	x	x		x	x	x	x	x	x	x	x		x	x	x	x	x	x		
6.1.1						x	x	x		x	x				x		x	x		
6.1.2	x	x		x	x	x	x	x	x				x	x	x	x		x		
7.1				x		x	x	x					x	x	x			x		
7	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x
7.2	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x
7.2.1	x	x			x	x	x		x	x	x	x			x	x			x	x
7.2.2	x	x		x	x			x	x	x	x	x	x	x		x		x	x	x
8.1	x	x	x	x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x
8	x	x	x	x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x
8.2	x	x		x	x	x	x	x	x	x	x		x	x	x	x		x		
8.2.1						x	x			x	x		x	x	x					
8.2.2	x	x		x	x	x	x	x	x				x	x	x	x		x		
	B	B	B	C	C	K	O	H	L	M	M	M	N	P	S	S	U	V	M	M
	h	s	b	a	p	1	1	i	m	g	p	p	m	a	e	p	u	c	p	g
	a	u	u	c	e	2	5	n	o	e	n	u	e	e	n	n	r	h	e	a
	l	b	r	e	r		7	n	n	n	e	l	n	r	t	e	e	o	n	l

Craig Falls, CB 3175 Sitterson Hall, Chapel Hill, NC 27516 USA
E-mail address: cfalls@cs.unc.edu
URL: http://www.cs.unc.edu/~cfalls/

E-mail address: bradford_powell@unc.edu

E-mail address: snoeyink@cs.unc.edu