Maximising the Size of Non-Redundant Protein Datasets Using Graph Theory

Bull, Simon C. and Muldoon, Mark R. and Doig, Andrew J.

2012

MIMS EPrint: **2012.68**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

Maximising the Size of Non-Redundant Protein Datasets Using Graph Theory

Simon C. Bull[1], Mark R. Muldoon[2], Andrew J. Doig[1*]

[1]Manchester Institute of Biotechnology, Faculty of Life Sciences, The University of Manchester, 131 Princess Street, Manchester M1 7DN, UK, [2]School of Mathematics, Alan Turing Building, University of Manchester, Manchester M13 9PL, UK

Abstract

Analysis of protein data sets often requires prior removal of redundancy, so that data is not biased by having multiple copies of similar proteins. This is usually achieved by pairwise comparison of sequences, followed by purging so that no two pairs have similarities above a chosen threshold. From a starting set, such as the PDB or a genome, one should remove as few sequences as possible, to give the largest possible non-redundant set for subsequent analysis. Protein redundancy can be represented as a graph, with proteins as nodes connected by undirected edges, if they have a pairwise similarity above the chosen threshold. The problem is then equivalent to finding the maximum independent set (MIS), where as few nodes are removed as possible to remove all edges. We tested seven MIS algorithms, three of which are new. We applied the methods to the PDB, subsets of the PDB, various genomes and the BHOLSIB benchmark datasets. For PDB subsets of up to 1000 proteins, we could compare to the exact MIS, found by the Cliquer algorithm.

The best algorithm was the new method, Leaf. This works by adding clique members that have no edges to nodes outside the clique to the MIS, starting with the smallest cliques. For PDB subsets of up to 1000 members, it usually finds the MIS and is fast enough to apply to data sets of tens of thousands of proteins. It gives sets that are around 10% larger than the commonly used PISCES algorithm, that are of identical quality. We therefore suggest that Leaf should be the method of choice for generating non-redundant protein data sets, though it is ineffective on dense graphs, such as the BHOLSIB benchmarks. The Leaf algorithm and sets from genomes and the PDB are available at: http://www.bioinf.manchester.ac.uk/leaf/.

* Email : andrew.doig@manchester.ac.uk

Supplementary Information

SI Document 1 Detailed Explanations of New Algorithms

SI Document 2 Examples of New Algorithms in Action

SI Document 3 Comparisons of Leaf and GLP algorithms to PISCES

SI Document 4 Algorithm Performances on Model Organisms

SI Document 5 Results from Culling PDB Data Sets

Introduction

Redundancy in datasets of proteins can be defined as multiple copies of similar proteins. Redundancy is a barrier to the effective use of the dataset for multiple reasons, most simply size. Redundant sequences in a dataset can prove detrimental to the discovery of novel relations between the proteins, as multiple copies of similar proteins can bias any conclusions drawn from using that set. Machine learning classifiers trained on redundant training sets will tend to over-fit and be of less value when applied to novel data. A pre-processing step is therefore often used to generate a non-redundant dataset consisting solely of representative proteins from the original redundant set.

Similarity between proteins is most usefully determined by comparing sequences, since structures are unavailable for most proteins and evolutionary relationships are difficult to quantify. Alignment based approaches to calculating sequence identity are either global or local methods, with local more useful when the two sequences may only share isolated regions of similarity, or when scanning a protein database with little to no *a priori* knowledge about the similarity between the database sequences and the query sequence (Barton 1996). The predominant heuristics for finding local alignments in proteins are BLAST (Altschul et al. 1990) or PSI-BLAST, which is more sensitive to weak sequence similarities in many cases (Altschul et al. 1997). BLAST is used by the protein redundancy removal application BlastCuller (Liu et al. 2009), while PISCES (Wang and Dunbrack 2003) makes use of PSI-BLAST to calculate the pairwise sequence identities.

Our intention here is to maximise the size of the non-redundant dataset. We test both novel and previously published methods that use graph theory. We show that it is possible to use novel graph theoretic methods to increase the size of non-redundant sets, while maintaining identical quality criteria for inclusion of proteins within the set. We find that our novel method, Leaf, generates the largest sets. We apply Leaf to generate non-redundant sets from the PDB, using various sequence similarity and structure quality parameters, and several genomes. Our webpage gives these sets, as well as a facility for users to generate their own non-redundant sets using Leaf.

Methods

Solving the Problem of Redundancy through Graph Theory

Sequence similarity relationships between proteins can be shown as a graph: A protein similarity graph *G(V, E)* denotes an undirected graph with vertices *V={1, 2, ... n}* and edges $E = \{\{i, j\} : i, j \in V\}$. Each protein in the redundant dataset is represented by a vertex. There is an edge between vertices *i* and *j* if the sequence identity of the proteins that *i* and *j* represent is greater than the similarity threshold, here taken to be an upper limit for acceptable mean percentage sequence identity. If the vertex which represents a protein has no edges incident to it, the pairwise sequence identity between that protein and every other protein in the dataset is below the similarity threshold. By representing the dataset and sequence similarities as a graph, it is possible to utilise graph theory to help optimise the generation of the non-redundant dataset.

A non-redundant dataset can be represented by a protein similarity graph that contains no edges. A non-redundant dataset can therefore be generated by removing vertices, and all edges incident to them, from the protein similarity graph until there are no edges remaining. The proteins that correspond to the vertices remaining in the graph will be the non-redundant dataset.

Our goal is to remove nodes, and incident edges, in such a way that the remaining vertices constitute the largest possible set of vertices that have no edges between them. The problem of finding this optimal set is known as the *maximum independent set* (MIS) problem, or equivalently the *stable set* problem. In graph theory, an *independent set* of a graph *G(V,E)* is a set of vertices $I \subseteq V$ such that $\forall i, j \in I : \{i, j\} \notin E$. An independent set *I* can be considered to be a *maximal independent set* if the addition of any vertex $v \in V$ that is not in *I* means that *I* no longer maintains the properties of an independent set. An MIS is a maximal independent set that contains the largest possible number of vertices. Finding the MIS is known to be an NP-complete problem, one where there is no known computationally efficient method for discovering the solution. Approximation based algorithms to find the MIS are thus often used instead.

Graph Definitions

In order to fully describe the properties of the developed algorithms, definitions of properties of the graphs is necessary: The *neighbourhood* of a vertex *v*, the vertices that share an edge with *v*, in an undirected graph *G(V,E)*, which contains no loops, can be defined as $neighbourhood(v) = \{i : \{i, v\} \in E\}$. In a protein similarity graph, the neighbourhood of *v* represents all the proteins that have a sequence which is too similar to the protein that *v* represents. The neighbourhood can also be defined for a set of vertices. If *s* is a set of vertices from *G*, then $neighbourhood(s) = \{i : v \in s, \{i, v\} \in E\}$. The *degree* of a vertex *v* in *G* can be defined as $\deg ree(v) = \# neighbourhood(v)$. The *support* of a vertex *v* in *G* can be defined as $\sum_{i \in neighbourhood(v)} \deg ree(i)$.

A *clique* $Q \subseteq V$ is a subset of the vertices of *G* such that $\forall i, j \in Q : \{i, j\} \in E$. A *maximum clique* of *G* is a largest possible subset of the vertices in *G* for which the clique property is satisfied. A *vertex cover* $C \subseteq V$ of *G* is a subset of vertices such that every edge in *G* is incident to at least one vertex in *C*. A *minimum vertex cover* of a graph *G* is a vertex cover *C* with the smallest possible number of vertices in it. Graph *components* are subgraphs that are not connected to each other. Finally, the *complement* of a graph *G(V,E)* is a second graph *H(V,E')* with the same vertex set, but a complementary edge set. That is, two vertices *i* and *j* are adjacent in *H* if and only if they are *not* adjacent in *G*. A maximum independent set in *G* is thus a maximum clique in *G*'s complement *H*.

PISCES

The benchmark for all the algorithms developed and tested here is PISCES, as it is very widely used (Wang and Dunbrack 2003). PISCES works by listing proteins in order of length. Redundancy is removed by: finding the protein highest up the list that is not marked as kept or removed, and marking it as being kept. For all proteins that have been determined to be too similar to this protein, mark them as being removed. Once the bottom of the list is reached all proteins that have been marked as being kept will be the non-redundant dataset. By only considering proteins higher up the list for inclusion, i.e. proteins with longer sequences, it is possible to miss the opportunity to increase the size of the non-redundant dataset. The returned set will also be biased to include long sequences. Here we evaluate algorithms that use graph theory to maximise the size of the non-redundant dataset while maintaining identical criteria for inclusion (e.g. no two proteins with more than 20% pairwise sequence identity).

New Algorithms

Two possible graph representations were used for the new algorithms. The first is an *adjacency matrix*. In this representation, an $n \times n$ matrix *M* is constructed, where *n* is the number of vertices in the graph. If there is an edge in between two vertices *i* and *j* in the graph, then $M_{i,j} = M_{j,i} = 1$. If no edge is present between the two vertices, then $M_{i,j} = M_{j,i} = 0$. In the *adjacency list* representation, there is one entry in the adjacency list for each vertex in the graph. The list records for each vertex *i* in the graph the vertices in $neighbourhood(i)$. Space is saved over the adjacency matrix representation when the graph is sparse as information is only stored about the presence of edges.

The density of the protein similarity graph of the entire human proteome was calculated for sequence identity thresholds of 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80% and 90%. The highest density was found for the 10% sequence identity threshold, but this was still only 0.01 on a scale where 0 indicates no edges in the graph and 1 indicates that the graph is complete (i.e. all members of *G* form a single clique). For the remaining percentage thresholds, the density was always below 0.005. Protein similarity graphs are therefore sparse, since the probability that any two proteins have a high pairwise sequence identity is <1%. An adjacency list representation is therefore utilised.

The protein similarity graph is processed before the algorithms are run, by removing all isolated nodes from the graph and adding them to the independent set, as they must all be members of the MIS.

First, we outline three novel algorithms to find an MIS. More detailed explanations, together with pseudocode, are given in SI Document 1.

Leaf

The Leaf algorithm works by identifying cliques in the graph that satisfy the criterion of having at least one vertex which is not connected to any vertex outside of the clique. One of the (potentially many) vertices in the clique with no connections outside of the clique is arbitrarily chosen to be kept in the independent set being formed. The algorithm starts by searching for cliques of two vertices which satisfy the criterion. If a clique is found, then one of the nodes in the clique is kept, and the other removed. If no clique is found that satisfies the criterion, then a clique of three vertices is searched for. This process of increasing the number of vertices in the clique being searched for is continued until either a clique is found, or there can be no possible clique in the graph that satisfies the criterion. After a clique has been found and one of its vertices has been incorporated into the growing independent set, the process of searching for a clique begins again with searching for a clique of two vertices. There is no clique in the graph that satisfies the criterion if the size of the clique being searched for is over a certain threshold size. This threshold size is determined dynamically, and is equivalent to the number of neighbours of the highest degree vertex in the graph. Although this upper bound could be tightened through more careful analysis of the graph, searching for a tight upper bound involves finding the size of the maximum clique in the graph. If no clique satisfying the criterion can be found, then the NeighbourCull algorithm is used to determine which vertex to remove. As this method removes the most connected vertex, the upper bound of the size of the clique being searched for will decrease.

NeighbourCull

The NeighbourCull algorithm is based on the goal of removing a vertex which has the highest connectivity (i.e. the most neighbours), but is minimally connected to the vertices not in its neighbourhood. The algorithm works by identifying the vertices with the most neighbours. If there is only one vertex with the most neighbours, then this vertex is removed. When multiple vertices have the most neighbours, the tie is broken by examining the neighbours of the neighbours of the original vertex (i.e. all vertices reachable by traversing two edges). The set of vertices reachable by traversing two edges is determined, and the vertex with the smallest set of vertices reachable in this manner is removed. If there are still multiple vertices which cannot be decided between, then the vertex to remove is chosen arbitrarily from amongst the remaining possibilities.

FIS

The third new algorithm works by first initialising a maximal independent set, and then permuting it in an attempt to increase its size. The algorithm's first step is to determine the initial vertex from which the maximal independent set will be generated. This is the vertex with the fewest neighbours, with ties broken arbitrarily. From this initial vertex, the set is permuted using the *addnodes* sub-function. This takes as its arguments the current independent set, and the set of all the vertices in the graph. This function works by first determining if there are any vertices that are not adjacent to the current independent set. If there are no non-adjacent vertices, then the current independent set is returned. If there are vertices which are not adjacent to the current independent set, then the independent set can be extended by adding a new vertex. This is done by finding the non-adjacent vertex which, when added to the independent set, causes the fewest vertices that are currently not adjacent to the independent set to become adjacent. The function *swapnodes* is used to see if the size of the independent set can be increased by making small alterations to the vertices in the set. The vertices that are not in the independent set are tested one at a time to see how many vertices from the independent set they are adjacent to. If a vertex *i* that is not in the independent set is adjacent to only one vertex *j* that is, then *i* and *j* can be swapped without invalidating the properties of a maximal independent set. The new independent set resulting from this swap is passed to *addnodes* to see if it can be extended by the addition of any non-adjacent vertices.

SI Document 2 shows illustrative examples of Leaf, NeighbourCull and FIS in action.

Existing Algorithms

Three algorithms from the literature were chosen to be tested alongside the three new algorithms:

GLP is a state of the art heuristic for approximating the maximum clique that works by finding an initial clique starting from a random initial vertex in the graph, and then improving this initial clique using local search operations (Grosso et al. 2008). Algorithm 1 from this paper, along with restart rule 2, is used here. An implementation of the algorithm was written in Python. As the GLP algorithm makes use of rules for restarting, it is possible that the algorithm will execute for a substantial length of time on larger graphs. For this reason, a parameter is used with the GLP algorithm that limits the number of vertices that can be added to and removed from the clique being generated. There are two problems with using the algorithm as it stands for the generation of non-redundant datasets. The first is that the size of the maximum clique of the graphs being used was known in the tests done in the GLP paper (Grosso et al. 2008). This meant that the algorithm could be stopped if the clique being generated ever reached this size. Unfortunately, the protein similarity graphs being used here have unknown MISs, and therefore the algorithm cannot be stopped early in the same way. Secondly, the value of the parameter cannot easily be set to prevent the algorithm running for excessive lengths of time. In order to prevent this, GLP was adapted to allow a time limit to be placed on the running of the algorithm and the largest maximal independent set found up to that point is returned.

The final two algorithms work based on the neighbourhood of the individual vertices in the graph. The VSA algorithm of Balaji *et al*. (Balaji et al. 2010) finds an approximation to the MIS by calculating an approximation to the minimum vertex cover. The algorithm works by beginning with an empty vertex cover, and progressively increasing the number of vertices in the vertex cover by adding the vertex with the highest support. If two vertices have the largest value for the support, then the vertex with the higher number of neighbours is added. This is repeated until there are no vertices that are not either in the vertex cover or adjacent to a vertex in the vertex cover. This algorithm was re-implemented in Python and extended to incorporate a limit on the length of execution.

The final algorithm used was BlastCuller (Liu et al. 2009). Unlike GLP and VSA, this algorithm is designed to produce non-redundant protein datasets. BlastCuller generates a non-redundant dataset by approximating the MIS of the protein similarity graph. The algorithm works by initialising the result as an empty set, and then adding to it all the isolated vertices. Following this, the vertex with the most neighbours is deleted, and any newly isolated vertices are added to the result set. This process is repeated until all vertices have been either removed, or added to the result set. BlastCuller was re-implemented in Python for the tests here, in order to enable a time limit on the exaction to be incorporated.

Cliquer

Although there are no known efficient algorithms to compute the MIS of an arbitrary graph, it is nonetheless possible to find the size of the MIS exactly using so-called branch-and-bound algorithms, which have worst-case running times that are exponential in the number of vertices. These algorithms typically combine a brute-force search (list all possible subsets of the vertex set and ask whether each is an independent set, keeping track of the largest set seen so far) with a clever upper bound that allows one to prove statements such as “any independent set that includes vertices 1, 25, 1548 and 21973 contains at most 53 other vertices” and so eliminate whole families of subsets without having to enumerate and check each member.

To obtain exact answers against which to check our algorithms we used the Cliquer library (Östergård 2002; Niskanen and Östergård 2003) to find a maximum clique in the complement of the protein similarity graph. Cliquer works by successively computing the size c_i of the maximum clique in the subgraph that contains only the vertices in the set $S_i = \{v_1, v_2, \ldots, v_i\}$ and any edges running between them. It’s clear that either $c_{i+1} = c_i$ or $c_{i+1} = c_i + 1$, with the latter holding only when there is a maximum clique in S_{i+1} that includes the vertex v_{i+1}: this is the key observation behind the upper bound that speeds Cliquer’s search. The algorithm’s running time depends on the order in which the vertices are listed and we used Cliquer’s default ordering strategy, which proceeds in two stages. Initially the vertices are arranged in order of decreasing degree; one then uses the greedy colouring algorithm recursively to choose large sets of non-adjacent vertices. The final vertex ordering lists the vertices in order of increasing color-index (as assigned by the greedy colouring stage) and, within each colour-group, in order of decreasing degree.

Experimental Design

The algorithms were compared in terms of the number of proteins removed from the original redundant dataset of the human proteome (downloaded from http://www.uniprot.org/downloads on December 10th, 2010), and the time taken to finish. Pairwise sequence identities between all possible pairs of the 20251 human proteins were calculated using PSI-BLAST version 2.2.25. PISCES was used to perform the BLASTing, and to process the resulting alignment information. From this alignment file, it was possible to determine which proteins had a percentage pairwise sequence identity over any specified threshold.

Random datasets of 100, 250, 500, 1000, 2000 and 5000 proteins were generated by sampling from the 20251 human proteins downloaded from UniProt. 50 datasets were generated randomly of each size. Taking the 2000 protein datasets as an example, the process for generating datasets was as follows:

1. Select 2000 different proteins from the 20,251 possible proteins.
2. Extract any alignments from the alignment file where the 2000 proteins were either the query or the hit in the PSI-BLAST output.
3. Form an alignment file from the subset of entries selected in step 2, and a FASTA format file of the proteins selected in step 1.
4. Repeat steps 1-3 until 50 datasets have been generated.

This method of generating datasets ensures that the same protein is not present multiple times in any one dataset, but may be present in more than one dataset of any given size.

Individual datasets were tested as follows:

1. Generate an adjacency list for each percentage threshold (10%, 20%, 30%, 40%, 50%, 60%, 70%, 80% and 90%).
2. Run PISCES on the dataset using each of the nine percentage thresholds.
3. Run each of the algorithms being tested on each of the nine adjacency lists.

The time limit for each algorithm was set to be the longer of either ten times the running time of the Leaf algorithm, or the time that PISCES took, whichever is the longer. The model organism data used for the comparison of the algorithms, was downloaded from UniProt on November 3rd 2011. For each proteome, only reviewed proteins in the complete proteome were downloaded. The taxonomy ID for the proteomes was: 9606 for *h.sapiens*, 10090 for *mus musculus*, 83333 for *E. coli*, 559292 for *S. cerevisiae* and 3702 for *Arabidopsis thaliana*.

The algorithms were also tested using a MIS benchmark suite of graphs, BHOSLIB (http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm). This benchmarking serves to test the ability of the algorithms to find a MIS in general, rather than simply in protein similarity graphs. Additionally, due to the difficulty of the problems in the BHOSLIB benchmarks suite, this should give an idea of how the algorithms designed for the simpler protein similarity graphs fare on more complex graphs in comparison to state of the art heuristics.

Results

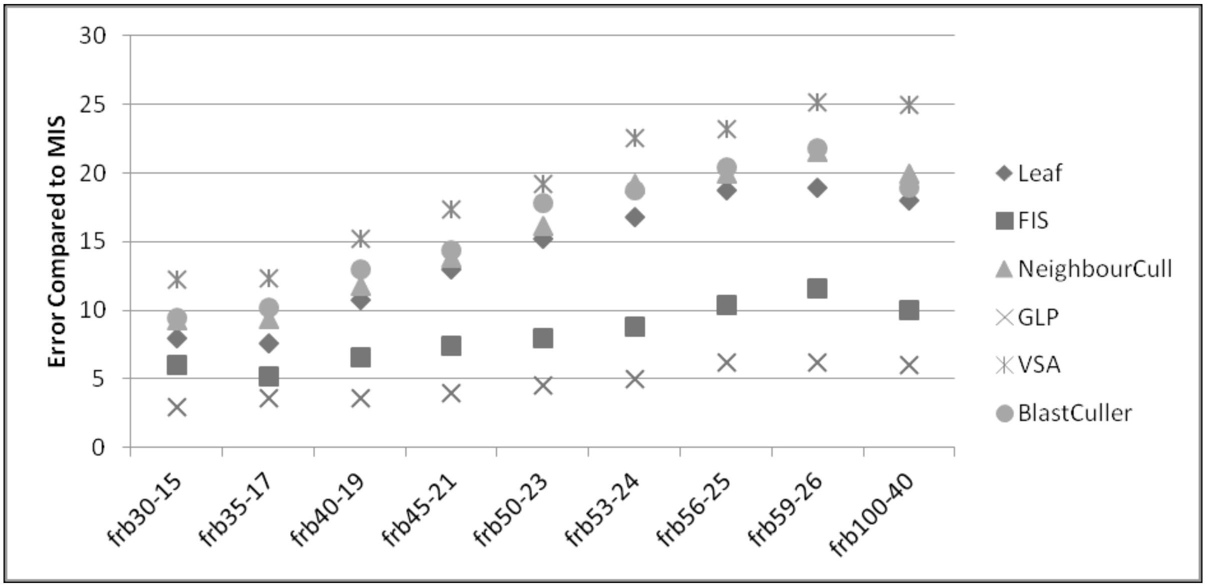
Subsets of the human proteome

The quality of each algorithm tested is measured as the number of proteins removed from the starting set, where the smaller the number removed, the better (Table 1).

GLP

When the execution time of the GLP algorithm was limited to ten times that of the Leaf algorithm, GLP performed more poorly than Leaf and often worse than PISCES (SI Document 3). This is mainly due to GLP terminating before it has had a chance to build up a maximal independent set in the protein similarity graph.

In order to determine whether running GLP for a longer length of time will increase the size of the non-redundant dataset generated, we extended the time limit for the execution of GLP to 500 times the Leaf execution time and studied a subset of the 5000 protein datasets (Figure 1). The improvement over PISCES was lower for GLP than for Leaf at all sequence identity thresholds. At all but 10%, 40% and 50% sequence identity, the results for GLP were worse than those of PISCES.



Comparison of Leaf and GLP to PISCES for 5000 protein data sets, when GLP is terminated after executing for 500 times as long as Leaf.

Leaf, FIS, NeighbourCull, VSA and BlastCuller

At sequence identities greater than 50%, there is little improvement over PISCES for any of the algorithms (Table 1). Gains are small at the higher sequence identities because the protein similarity graphs themselves contain only a few proteins. For example, using a 90% sequence identity threshold with datasets of 1000 proteins generates protein similarity graphs with a mean of 9.68 proteins, and the mean number of nodes in each component is 2.68. The small size of the components leaves very little room for an improvement in the size of the non-redundant dataset.

For sequence identity thresholds below 60%, the improvement over PISCES achieved by all five algorithms is more substantial, (Table 1). The pattern of improvement changes depending on the sequence identity threshold used. One trend that is noticeable across all sequence identity thresholds is the increasing difference between the five algorithms as the datasets increase in size. This is likely to be due to the increased size and complexity of the protein similarity graphs of larger datasets.

The order of success of the algorithms is the same for almost every combination of dataset size and sequence identity threshold, with Leaf showing the most improvement followed by FIS, NeighbourCull, BlastCuller and finally VSA. The three algorithms that work solely by identifying the vertex that is most connected by some measure show the smallest improvement over PISCES.

Comparisons to the Maximum Independent Set

The Cliquer algorithm computes the exact size of a maximum independent set, which is the perfect solution to our problem of finding the largest possible non-redundant protein data set. Unfortunately, it is so slow that it is only possible to find the MIS for starting sets of 1000 proteins or fewer with this method. We ran starting sets of 5000 proteins and none reached a solution after 6 months of processing on a Condor(Thain et al. 2005) distributed computing pool. Jobs submitted to this pool run mainly on inactive, recent-model desktop machines in student computing clusters and, during the academic term, get around 8-10 hours of uninterrupted processor time per day. Nevertheless, we can compare the approximate methods used here to the exact solution for sets of 100, 250, 500 and 1000 proteins. Table 2 shows these comparisons. We see that the perfect solution is often achieved. For example, Leaf finds the MIS every time for the 250 protein subsets. Even for the 1000 proteins subsets, Leaf misses the MIS in only a few cases, shown, for example, by the mean error for the 20% cut off being only 0.1 proteins. This gives reassurance that we have found highly accurate algorithms that can reach, or get close to, the MIS in a short time. For example, with the 1000 protein sets at a 20% cut off, the Cliquer algorithm takes on average 4130 seconds to find the MIS for each set, while the Leaf method needs only 42ms and nearly always finds the MIS.

Human Proteome

The results of running the algorithms on the entire human proteome (20251 proteins) are in Table 3. Leaf again outperformed the other algorithms in most cases.

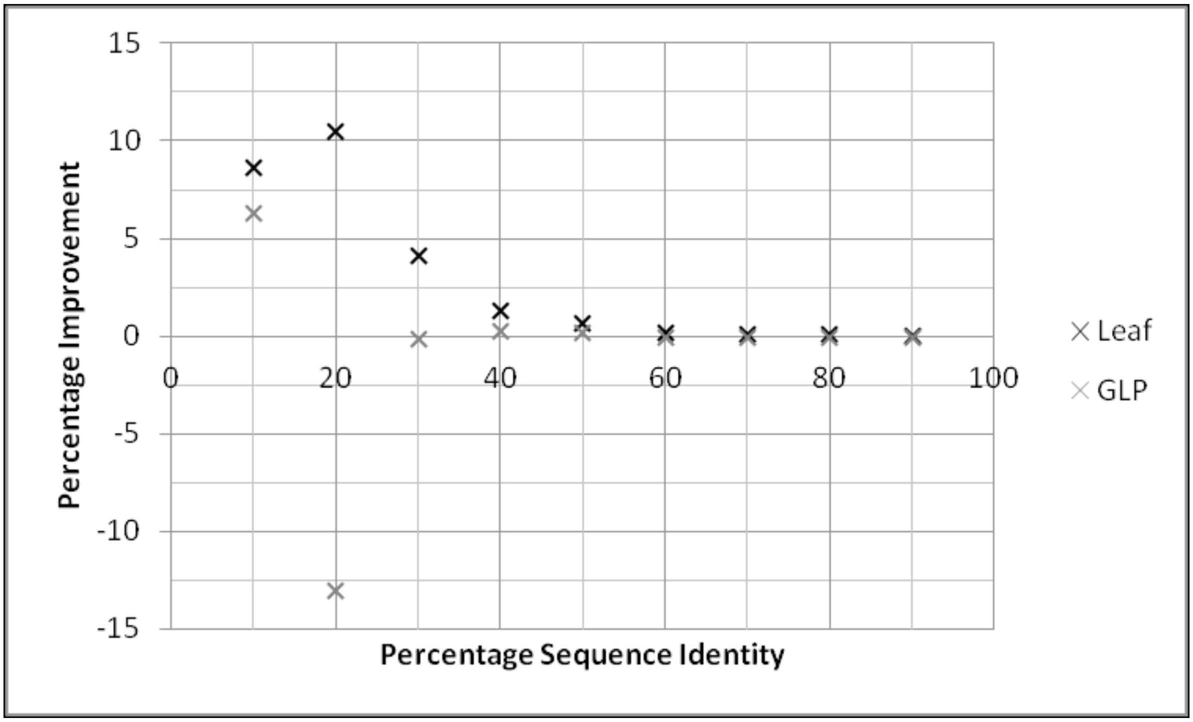
GLP was originally used in the test on this datasets, but the size of the representation of the protein similarity graph proved to be problematic. For example, the largest connected component of the protein similarity graph at 10% sequence identity contains 16,966 vertices, and has a mean degree of 215. The complement of this graph will therefore have the same number of vertices, but a mean degree of 16,751. In order to record all the connections between vertices in the graph, approximately 284 million connections need to be recorded. The size of the graph representation will cause the algorithm to be substantially slower, making the time required to generate a non-redundant dataset prohibitive. Similar issues may explain the poor results of GLP on the random subsets of the human proteome.

Model Organisms

We applied MIS algorithms to the *mus musculus, E. coli, Arabidopsis thalania* and *s. cerevisiae* proteomes, in order to evaluate its performance on diverse proteomes and to generate potentially useful data sets for groups studying these organisms. SI Document 4 summarises their performances and shows that again Leaf consistently gives the largest culled sets.

PDB

Non-redundant sets of protein crystal structures are often used to study protein structure. PDB files can be culled not just on the maximum pairwise sequence identity, but also structure quality, as measured by minimum resolution and R-factor. We used the Leaf algorithm to compare with PISCES, using a range of these parameters (SI Document 5). For sets with low sequence identities (20% – 25%), Leaf returns data sets that are around 10% larger than from PISCES.



Algorithm comparison for the BHOSLIB Benchmarks

BHOSLIB Benchmark

The results of running Leaf, FIS, GLP, VSA and BlastCuller on the BHOLSIB benchmark datasets can be seen in Figure 2, where the mean difference between the number of vertices returned by the algorithms and the true MIS is shown..

GLP consistently outperforms the other algorithms on the benchmark datasets, unlike the protein datasets. For the other algorithms, the structure of the graphs is not suitable for the simple methods used to generate the independent sets. For example, the Leaf method is unlikely to be able to use the fact that a clique has a vertex which is not connected to any vertices not in the clique, as these vertices are uncommon in this test graphs. This will cause Leaf to behave very similarly to NeighbourCull, as it falls back on the removal of the vertex with the most neighbours. Hence, the results for Leaf and NeighbourCull are very similar.

Leaf Protein Culling Server

We have implemented the Leaf method to provide datasets (http://www.bioinf.manchester.ac.uk/leaf/). The website uses Leaf to cull subsets of the PDB or submitted user sequences. Pre-computed sets of nonredundant PDB chains can also be downloaded, along with the source code and data files needed to run the culling on a local machine. Preculled PDB datasets are available with various sequence identity cut offs, resolutions and R-value limits. Culled proteomes are available for *h. sapiens, E. coli, Arabidopsis thaliana, S. cerevisiae* and *mus musculus*.

Discussion

When comparing algorithms, the one that clearly underperforms is GLP. This algorithm substantially underperforms when compared to Leaf, and occasionally when compared to PISCES. Even when the time was increased to 500 times that of Leaf, the datasets returned by GLP were still smaller than those returned by Leaf. GLP has to work on very large graphs as it uses the complement of the protein similarity graph. While this problem can be overcome by using large amounts of memory, the time needed to produce a suitable result on larger graphs is far too large. For these reasons it is undesirable to use GLP, at present, for starting sets of this nature.

The Leaf algorithm consistently outperformed the other algorithms on both the datasets of random proteins, and the datasets of biological importance. It is not a general solution to the MIS problem, however, as its relatively poor performance on the BHOSLIB Benchmark data sets, suggest that it is only appears suitable for sparse graphs. In conclusion, the Leaf algorithm is the most suitable for finding non-redundant protein datasets of maximal size.

FUNDING

Simon Bull thanks the BBSRC (UK) for a PhD studentship.

ACKNOWLEDGEMENTS

We thank Penny Doig and Mark Kambites, who helped with developing NeighbourCull. Our Cliquer computations ran on the High Throughput Computing facility of the Faculty of Engineering and Physical Sciences at The University of Manchester.

AUTHOR CONTRIBUTIONS

SB, MM and AD designed the project, analysed results and wrote the paper. MM implemented the Cliquer algorithm. SB performed the remaining work.

Conflict of Interest: none declared.

REFERENCES

Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) Basic local alignment search tool. J Mol Biol 215: 403-410.

Altschul SF, Madden TL, Schäffer AA, Zhang J, Zhang Z et al. (1997) Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. Nuc Acids Res 25: 3389-3402.

Balaji S, Swaminathan V, Kannan K (2010) A simple algorithm to optimize maximum independent set. Adv Modeling Optimization 12: 107-118.

Barton GJ (1996) Protein sequence alignment and database scanning. In: Sternberg MJE, editor. Protein Structure Prediction. Oxford: Oxford University Press. pp. 31-64.

Grosso A, Locatelli M, Pullan W (2008) Simple ingredients leading to very efficient heuristics for the maximum clique problem. J Heuristics 14: 587-612.

Li W, Jaroszewski L, Godzik A (2001) Clustering of highly homologous sequences to reduce the size of large protein databases. Bioinformatics 17: 282-283.

Liu P, Zeng Z, Qian Z, Feng K, Cai Y. A graph theoretic algorithm for removing redundant protein sequences; 2009; Beijing. IEEE. pp. 1-3.

Niskanen S, Östergård PRJ (2003) Cliquer User's Guide, Version 1.0. Communications Laboratory, Helsinki University of Technology.

Östergård PRJ (2002) A fast algorithm for the maximum clique problem. Discrete Applied Mathematics 120(1-3): 197-207.

Thain D, Tannenbaum T, Livny M (2005) Distributed computing in practice: the Condor experience. Concurrency - Practice and Experience 17(2-4): 323-356.

Wang GL, Dunbrack RL (2003) PISCES: a protein sequence culling server. Bioinformatics 19: 1589-1591.

TABLES

Table 1. Mean Number Kept from Datasets of 100, 250, 500, 1000, 2000 and 5000 Proteins

100 Proteins

Cut Off	PISCES	Leaf	FIS	NeighbourCull	VSA	BlastCuller
10%	79.3	81.3	81.3	81.3	80.9	81.2
20%	89.6	90.2	90.2	90.2	90.1	90.1
30%	95.8	95.9	95.9	95.9	95.9	95.9
40%	98.2	98.2	98.2	98.2	98.2	98.2
50%	99.3	99.3	99.3	99.3	99.3	99.3
60%	99.7	99.7	99.7	99.7	99.7	99.7
70%	99.8	99.8	99.8	99.8	99.8	99.8
80%	99.9	99.9	99.9	99.9	99.9	99.9
90%	99.9	99.9	99.9	99.9	99.9	99.9

250 Proteins

Cut Off	PISCES	Leaf	FIS	NeighbourCull	VSA	BlastCuller
10%	174.5	183.4	183.4	183.2	182.0	183.0
20%	204.9	208.8	208.7	208.7	207.9	208.5
30%	231.7	232.4	232.4	232.4	232.3	232.4
40%	241.2	241.9	241.9	241.9	241.8	241.9
50%	246.9	246.9	246.9	246.9	246.9	246.9
60%	248.3	248.4	248.4	248.4	248.4	248.4
70%	248.9	249.0	249.0	249.0	249.0	249.0
80%	249.2	249.2	249.2	249.2	249.2	249.2
90%	249.6	249.6	249.6	249.6	249.6	249.6

500 Proteins

Cut Off	PISCES	Leaf	FIS	NeighbourCull	VSA	BlastCuller
10%	311.2	329.8	329.5	329.1	326.3	328.6
20%	371.9	384.3	384.1	384.0	380.4	383.4
30%	442.2	445.1	445.0	445.0	444.5	444.8
40%	472.3	474.5	474.5	474.5	474.1	474.4
50%	488.5	489.0	489.0	489.0	488.9	489.0
60%	493.4	493.4	493.4	493.4	493.4	493.4
70%	496.0	496.0	496.0	496.0	496.0	496.0
80%	497.2	497.2	497.2	497.2	497.2	497.2
90%	498.4	498.4	498.4	498.4	498.4	498.4

1000 Proteins

Cut Off	PISCES	Leaf	FIS	NeighbourCull	VSA	BlastCuller
10%	554.4	591.1	589.6	589.1	582.7	587.9
20%	668.0	699.6	698.7	698.0	688.2	695.8
30%	840.2	849.2	849.0	849.0	846.9	848.6
40%	917.4	922.0	921.9	921.9	920.6	921.6
50%	958.4	960.3	960.3	960.3	960.2	960.3
60%	976.6	977.0	977.0	977.0	977.0	977.0
70%	985.5	985.6	985.6	985.6	985.6	985.6
80%	990.4	990.6	990.6	990.6	990.6	990.6
90%	994.8	994.9	994.9	994.9	994.9	994.9

2000 Proteins

Cut Off	PISCES	Leaf	FIS	NeighbourCull	VSA	BlastCuller
10%	969.1	1040.0	1036.9	1036.1	1022.9	1033.4
20%	1158.2	1240.1	1236.2	1234.9	1210.1	1229.9
30%	1544.7	1575.5	1574.8	1574.6	1566.2	1573.0
40%	1754.6	1768.7	1768.2	1768.4	1765.6	1767.9
50%	1864.1	1870.1	1870.1	1870.1	1869.3	1870.1
60%	1920.1	1922.1	1922.1	1922.1	1922.0	1922.1
70%	1947.9	1948.9	1948.9	1948.9	1948.8	1948.9
80%	1980.6	1980.9	1980.9	1980.9	1980.9	1980.9
90%	1980.6	1981.0	1981.0	1981.0	1981.0	1981.0

5000 Proteins

Cut Off	PISCES	Leaf	FIS	NeighbourCull	VSA	BlastCuller
10%	1940.6	2107.7	2098.1	2096.6	2063.9	2090.5
20%	2284.0	2520.0	2504.7	2503.3	2439.7	2491.8
30%	3293.0	3423.1	3419.2	3417.7	3380.8	3410.7
40%	3997.3	4052.1	4050.5	4050.6	4040.9	4048.5
50%	4385.4	4417.8	4417.8	4417.5	4413.0	4416.9
60%	4622.7	4634.1	4634.0	4634.0	4632.7	4633.7
70%	4756.5	4762.5	4762.5	4762.5	4762.2	4762.4
80%	4905.6	4908.5	4908.5	4908.5	4908.4	4908.5
90%	4905.8	4908.7	4908.7	4908.6	4908.6	4908.7

Table 2. Comparisons to exact algorithm

100 Protein Subsets

Cut Off	Exact Kept	NeighbourCull Error	FIS Error	Leaf Error	VSA Error	BlastCuller Error	GLP Error	PISCES Error
20%	90.2	0	0	0	0.1	0.1	0	0.5
30%	95.7	0	0	0	0	0	0	0.1
40%	98.1	0	0	0	0	0	0	0
50%	99.2	0	0	0	0	0	0	0
60%	99.6	0	0	0	0	0	0	0
70%	99.7	0	0	0	0	0	0	0
80%	99.8	0	0	0	0	0	0	0
90%	99.9	0	0	0	0	0	0	0

250 Protein Subsets

Cut Off	Exact Kept	NeighbourCull Error	FIS Error	Leaf Error	VSA Error	BlastCuller Error	GLP Error	PISCES Error
20%	209.71	0.04	0.04	0	0.86	0.14	0.04	3.75
30%	232.64	0	0	0	0.11	0	0	0.79
40%	242.21	0	0.04	0	0.07	0.04	0	0.57
50%	247.00	0	0	0	0.04	0	0	0.07
60%	248.25	0	0	0	0	0	0	0.04
70%	248.82	0	0	0	0	0	0	0.04
80%	249.04	0	0	0	0	0	0	0.04
90	249.50	0	0	0	0	0	0	0.04

500 Protein Subsets

Cut Off	Exact Kept	NeighbourCull Error	FIS Error	Leaf Error	VSA Error	BlastCuller Error	GLP Error	PISCES Error
20%	385.34	0.3	0.3	0.07	4.0	1.00	4.7	11.5
30%	445.9	0.2	0.1	0	0.6	0.4	0.2	2.7
40%	476.1	0	0	0	0.4	0.07	0.7	2.3
50%	489.9	0	0	0	0.07	0	0.3	0.3
60%	494.0	0	0	0	0	0	0	0.03
70%	496.2	0	0	0	0	0	0	0
80%	497.3	0	0	0	0	0	0	0
90%	498.4	0	0	0	0	0	0	0

1000 Protein Subsets

Cut Off	Exact Kept	NeighbourCull Error	FIS Error	Leaf Error	VSA Error	BlastCuller Error	GLP Error	PISCES Error
20%	699.7	1.7	1.0	0.1	11.5	3.9	40.3	31.7
30%	849.2	0.2	0.2	0	2.3	0.6	174	9.0
40%	922.0	0.2	0.2	0.06	1.4	0.4	2.4	4.6
50%	960.3	0.02	0	0	0.2	0.02	1.2	1.9
60%	977.0	0	0	0	0	0	0	0.4
70%	985.6	0	0	0	0	0	0	0.2
80%	990.6	0	0	0	0	0	0	0.1
90%	994.9	0	0	0	0	0	0	0.1

Table 3. Number of Proteins Kept from Human Proteome

Cut Off	PISCES	Leaf	FIS	NeighbourCull	VSA	BlastCuller
10%	5073	5636	5585	5600	5487	5578
20%	5700	6643	6572	6580	6365	6541
30%	9007	9856	9796	9800	9594	9762
40%	12422	12843	12832	12829	12746	12811
50%	14927	15169	15167	15164	15129	15154
60%	16771	16887	16884	16886	16874	16884
70%	17969	18036	18036	18036	18030	18033
80%	18763	18801	18801	18801	18798	18801
90%	19366	19389	19388	19389	19388	19388

Supplementary Information 1 – Detailed Explanations of New Algorithms

Maximising the Size of Non-Redundant Protein Data Sets Using Graph Theory

Simon C. Bull, Mark R. Muldoon and Andrew J. Doig

NeighbourCull

The NeighbourCull algorithm involves repeatedly removing a vertex which has the highest connectivity (i.e. the most neighbours), but is minimally connected to the vertices not in its neighbourhood. An outline of the procedure is sketched in Algorithm 1. First the set of vertices that are not in the maximal independent set is initialised (line 1). Following this, the loop (lines 2-14) that determines which vertices to exclude from the maximal independent set is entered. The first step is to find those vertices that are remain in the graph and still have neighbours (line 3). If there are no vertices with neighbours, and hence no edges in the protein similarity graph, a maximal independent set has been found and the algorithm can exit (lines 4 and 5). If some edges remain, then we find those vertices that have the most neighbours (line 7). If there is only a single vertex that has the maximum number of neighbours, it is marked as not being in the maximal independent set (lines 8-10). However, if there are multiple vertices with the maximum number of neighbours then we do further analysis to decide which one to remove (lines 12-14).

In cases where more than one vertex has maximal degree, the one to remove is determined by two applications of the $nei\boldsymbol{g}\boldsymbol{h}bour\boldsymbol{h}ood$ function. We compute the size of the *extended neighbourhood*

$$nei\boldsymbol{g}\boldsymbol{h}bour\boldsymbol{h}ood(v) \bigcup nei\boldsymbol{g}\boldsymbol{h}bour\boldsymbol{h}ood\big(nei\boldsymbol{g}\boldsymbol{h}bour\boldsymbol{h}ood(v)\big)$$

and remove a vertex whose extended neighbourhood is smallest, resolving any remaining ties by an arbitrary choice.

```
1.  $Removed := \emptyset$
2.  While $True$
3.    $nodesWit\boldsymbol{h}Neig\boldsymbol{h}bours \coloneqq \{i \dashv | i \in V: \#neighbourhood(i) > 0\}$
4.    If $nodesWit\boldsymbol{h}Neig\boldsymbol{h}bours = \emptyset$
5.      Return $Removed$
6.    Else
7.      $max = \{i \in V \dashv | \forall j \in V: \#neighbourhood(i) \geq \#neighbourhood(j)\}$
8.      If $\#max = 1$
9.        $Removed \coloneqq Removed + \mathrm{max}$□
10.       <Update the adjacency list to reflect the removal of $\mathrm{max}$□>
11.     Else
12.       <Select $n \in \mathrm{max}$□ such that $n$ has the smallest extended neigbourhood>
13.       $Removed \coloneqq Removed + n$
```

```
14.         <Update the adjacency list to reflect the removal of $n$ >
```

Algorithm 1: Pseudocode for the NeighbourCull algorithm

Leaf

The Leaf algorithm works by identifying cliques in the graph that satisfy the criterion of having at least one vertex which is not connected to any vertex outside of the clique. An outline of the algorithm can be seen in Algorithm 2. First the set of vertices that are not in the maximal independent set is initialised (line 1). Next a loop is entered (lines 2-18), which is only exited once there are no edges remaining in the graph (lines 4 and 5). If there are edges remaining, then the next step is to select a vertex to add to the independent set, or one to remove from graph. First the variable $nClique$ is initialised (line 6). This is the size of the neighbourhood that all vertices in the clique being searched for must possess. A loop is entered to search for sequentially larger cliques (lines 7-14). If a clique is found where there is at least one vertex v in the clique that does not share any edges with a vertex not in the clique, then v is to be added to the independent set being formed (lines 8-12). If no clique of a given size is found, then the size of clique being searched for is incremented (lines 13 and 14). If the loop in lines 7-14 terminates without finding a clique, then the NeighbourCull method is used to determine a vertex to delete (lines 15-18).

```
1.  $Removed := \emptyset$
2.  While $True$
3.    $max = \{i \in V \dashv | \forall j \in V: \#neighbourhood(i) \geq \#neighbourhood(j)\}$
4.    If $\#neighbours(max) = 0$
5.      Return $Removed$
6.    $nClique \coloneqq 1$
7.    While
8.      If there is a clique $C$ of $nClique + 1$ vertices that satisfies the criterion for Leaf
9.        $toKeep \coloneqq i$ where $i \in C$
10.       $Removed \coloneqq Removed \cup \{\forall j \in C: j \neq i\}$
11.       <Update the adjacency list to reflect the removal of vertices in $C$ that are not $i$ >
12.       Exit the inner while loop
13.     Else
14.       $nClique \coloneqq nClique + 1$
15.   If
16.     Use NeighbourCull to determine the vertex $v$ to remove
17.     $Removed \coloneqq Removed + v$
18.     <Update the adjacency list to reflect the removal of $v$ >
```

Algorithm 2: Pseudocode for the Leaf algorithm,

FIS

The third new algorithm works by first initialising a maximal independent set in a greedy manner, and then attempting to permute this maximal independent set in an attempt to increase its size. An outline of the algorithm, including its two sub-functions $addnodes$ and $swapnodes$, can be seen in Algorithm 3. The algorithm's first step is to determine the initial vertex from which the maximal independent set will be generated. This is done in line 1, and is chosen to be the vertex with the fewest neighbours, with ties broken arbitrarily. From this initial vertex a maximal independent set is generated (line 3), and following this the set is permuted in an attempt to increase its size (line 4). Once the set has been permuted, either the permuted independent set (line 6) or the non-permuted set (line 8) is returned based on which contains a greater number of vertices.

The majority of the work in the algorithm is done in the $addnodes$ sub-function. This takes as its arguments the current independent set, and the set of all the vertices in the graph. This function works by first determining if there are any vertices that are not adjacent to the current independent set (line 11). If there are no non-adjacent vertices, then the current independent set is returned (line 12 and 13). If there are vertices which are not adjacent to the independent set being formed, then the independent set can be extended by adding a new vertex (lines 15-17). This is done by finding the non-adjacent vertex which, when added to the independent set, causes the fewest vertices that are currently not adjacent to the independent set to become adjacent. The number of currently non-adjacent vertices that will become adjacent if a vertex i is added to the independent set Ind is determined to be $added_i = openN(i) - Ind$. Therefore the vertex j that is added to Ind is chosen such that $\forall v \in nonAdj{:}\, added_v \geq added_j$, where $nonAdj$ is the set of all vertices that are not adjacent to Ind .

```
1.  <Select the node $I$ such that $\#openN(I)$ is minimal>
2.  $Ind = I$
3.  $Ind_A = addnodes(Ind, V)$
4.  $Ind_S = swapnodes(Ind_A, V)$
5.  If $\# \llbracket Ind \rrbracket \downarrow A < \# \llbracket Ind \rrbracket \downarrow S$
6.      Return $Ind_S$
7.  Else
8.      Return $Ind_A$
```

$addnodes(Ind, V)$:

```
9.  $Start \coloneqq Ind$
10. While $True$
11.     $nonAdj = V - closedN(Start)$
12.     If $nonAdj = \emptyset$
13.         Return $Start$
```

```
14.   For $i$ in $nonAdj$
15.     If <$added$ is the smallest number found so far>
16.       $min = i$
17.   $Start \coloneqq Start + \min$

$swapnodes(Ind, V)$:

18. $Start \coloneqq Ind$ , $changed \coloneqq True$ , $maxSet = Ind$
19. While $changed$
20.   $changed \coloneqq False$
21.   For $i$ in $V - Start$
22.     $adj = closedN(i) \cap Start$
23.     If $\#adj = 1$
24.       $test$ := $Start - adj + i$
25.       $temp \coloneqq addnodes(test, V)$
26.       If $\#temp > \#maxSet$
27.         $maxSet \coloneqq temp$
28.         $Start \coloneqq temp$
29.         $changed \coloneqq True$
30. Return $maxSet$
```

Algorithm 3: Pseudocode for the FIS algorithm

The function $swapnodes$ is used at the end of the algorithm to see if the size of the independent set generated by line 3 can be increased by making small alterations to the vertices in the set. The vertices that are not in the independent set are tested one at a time to see how many vertices from the independent set they are adjacent to (lines 21-29). If a vertex i that is not in the independent set is adjacent to only one vertex j that is, then i and j can safely be swapped without invalidating the properties of a maximal independent set (line 24). The new independent set resulting from this swap is passed to $addnodes$ to see if it can be extended by the addition of any non-adjacent vertices (line 25). If the set returned by $addnodes$ contains more vertices than the largest maximal independent set previously found it is recorded as the current best maximal independent set (lines 26-29).

Supplementary Information 2 Examples of New Algorithms in Action

Maximising the Size of Non-Redundant Protein Data Sets Using Graph Theory

Simon C. Bull, Mark R. Muldoon and Andrew J. Doig

Example

The simplest method of fully understanding the new algorithms is through an example which demonstrates the differences between them. The graph in Figure 1 is one such graph, and will be used to illustrate the execution of the Leaf, NeighbourCull and FIS algorithms. For all three algorithms the alphabetic names of the vertices will be used to arbitrarily break any ties.

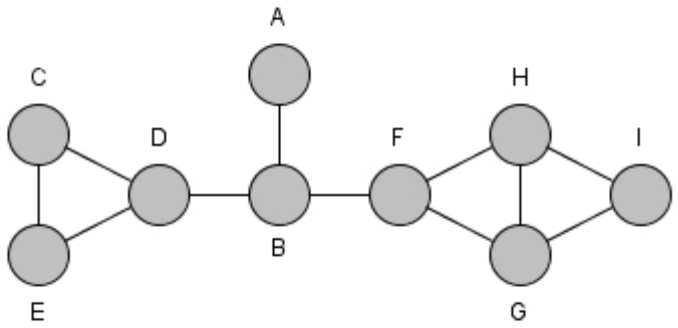


Figure 1: An example graph to demonstrate the differences between Leaf, NeighbourCull and FIS.

The execution of the Leaf algorithm on the graph in Figure 1 is as follows:

1. Select vertex A to keep. This is because vertices A and B comprise the only maximal clique of two vertices. Vertex B is not kept because it is connected to vertices that are not in the clique (Figure 2).
2. There are no more maximal cliques of two vertices, so cliques of three vertices are examined.
3. Select vertex C to keep. There are three maximal cliques of three vertices ($\{C, D, E\}$, $\{F, G, H\}$ and $\{G, H, I\}$), all of which contain at least one vertex that has no connection to a vertex not in the clique. Clique $\{C, D, E\}$ is arbitrarily chosen as the one to keep a vertex from. Vertex C is chosen arbitrarily from this clique. (Figure 2).
4. There are no maximal cliques of two vertices, so cliques of three vertices are examined.
5. Select vertex F to keep. Clique $\{F, G, H\}$ is arbitrarily chosen as the maximal 3-clique to keep a vertex from. Vertex F is the only vertex in the clique that has no connections to vertices not in the clique. Therefore vertex F is kept. (Figure 2).
6. Keep vertex I as it has no neighbours (Figure 2).
7. The final independent set is $\{A, C, F, I\}$.

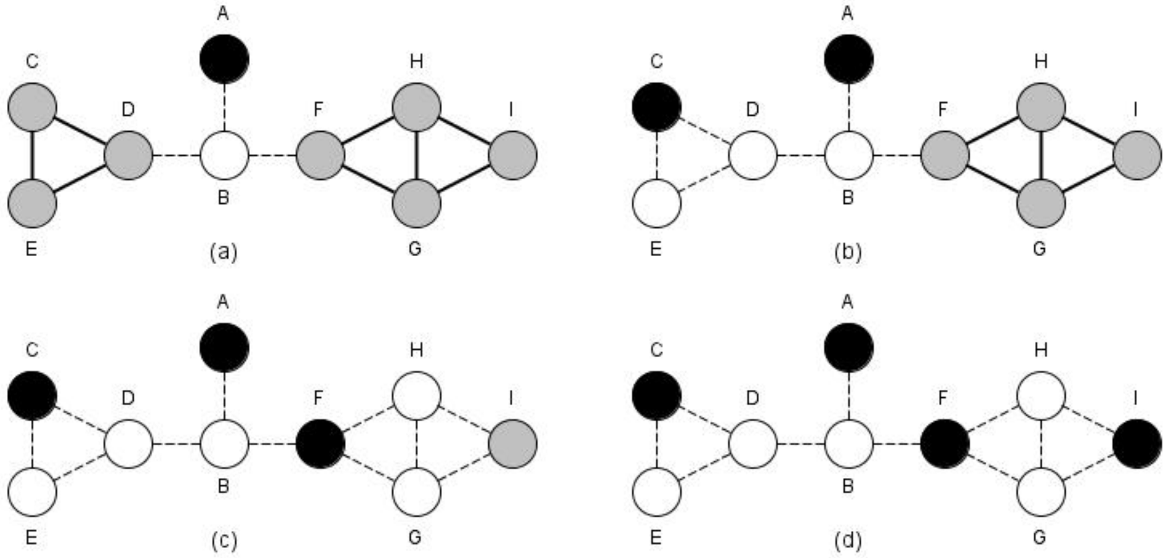


Figure 2: The progress of execution of the Leaf algorithm on the graph seen in Figure 1. Black vertices are in the independent set being generated, white vertices have been removed and grey vertices are those that are still to be decided upon. Dashed edges indicate edges that have been removed from the graph due to a vertex being removed. Each graph corresponds to the results of one the execution steps of the Leaf algorithm. (a) corresponds to step 1, (b) to step 3, (c) to step 5, (d) to step 6.

The execution of the NeighbourCull algorithm on the graph in Figure 1 is as follows:

1. Vertices B, D, F, G and H all have three neighbours, and no other vertex has more, so we need to look at the sizes of their extended neighbourhoods to choose a vertex for deletion. The relevant data are summarised in the table below where, in the column headings, $N(v)$ is an abbreviation for $neighbourhood(v)$ and $\#[N(v) \cup N(N(v))]$ is the size of the extended neighbourhood.

Vertex v	$N(v)$	$N(N(v))$	$\#N(v)$	$\#[N(v) \cup N(N(v))]$
B	$\{A, D, F\}$	$\{B, C, E, G, H\}$	3	8
F	$\{B, G, H\}$	$\{A, D, F, G, H, I\}$	3	7
D	$\{B, C, E\}$	$\{A, C, D, E, F\}$	3	6
G	$\{F, H, I\}$	$\{B, F, G, H, I\}$	3	5
H	$\{F, G, I\}$	$\{B, F, G, H, I\}$	3	5

   Vertices G and H have the smallest extended neighbourhoods, and so vertex G is arbitrarily chosen to be removed instead of H (Figure 3).
2. Vertices B and D now have the most neighbours of the remaining vertices, 3, while their extended neighbourhoods contain 7 and 6 vertices, respectively. Thus we remove vertex D.
3. Now vertices B, F and H all have two neighbours apiece, while their extended neighbourhoods contain either 4 (for B and H) or 5 (for F) vertices. We choose, arbitrarily, to remove vertex B.
4. Vertex H will be removed as it has the most neighbours (Figure 3).
5. Vertices C and E both have the most neighbours. Remove vertex C arbitraily (Figure 3).
6. The final independent set is $\{A, E, F, I\}$ (Figure 3).

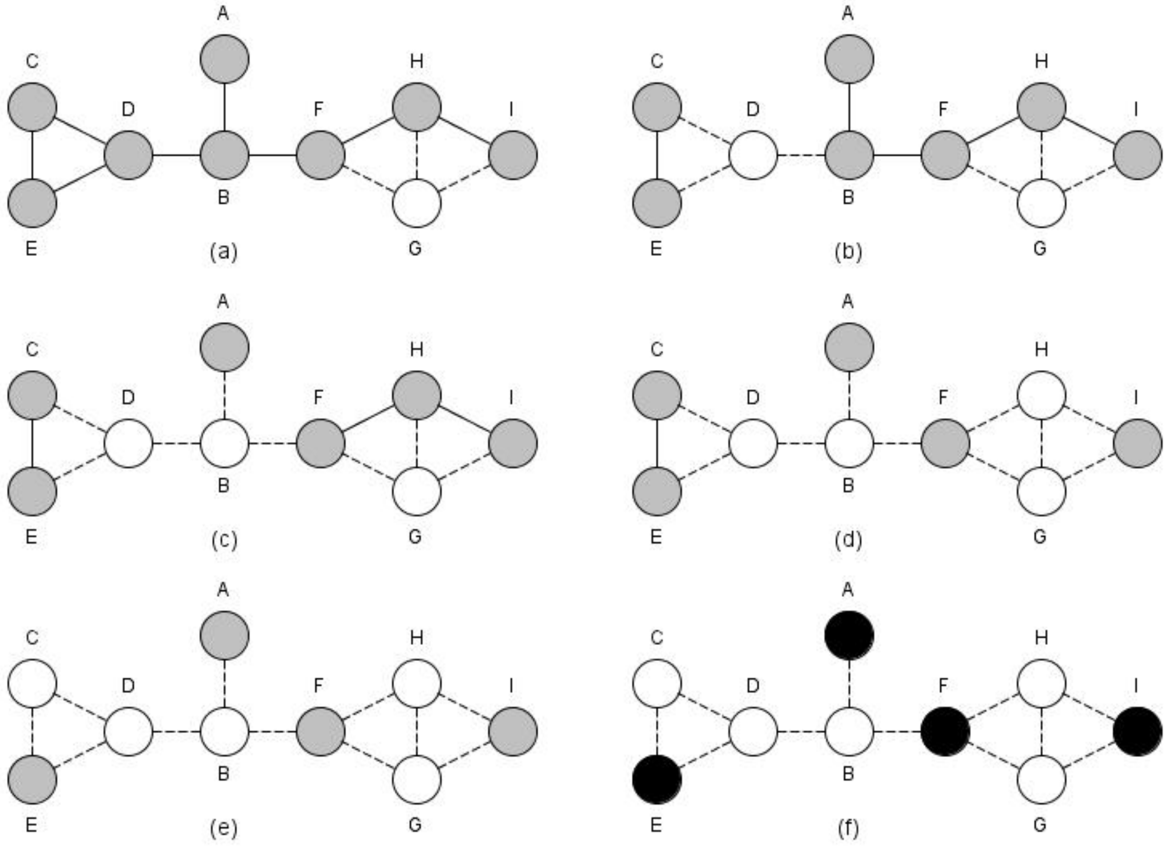


Figure 3: The progress of execution of the NeighbourCull algorithm on the graph seen in Figure 1. Black vertices are in the independent set being generated, white vertices have been removed and grey vertices are those that are still to be decided upon. Dashed edges indicate edges that have been removed from the graph due to a vertex being removed. Each graph corresponds to the results of one the execution steps of the NeighbourCull algorithm. (a) corresponds to step 2, (b) to step 4, (c) to step 6, (d) to step 7, (e) to step 8 and (f) to step 9.

The execution of the FIS algorithm on the graph in Figure 1 is as follows:

1. The initial vertex is set to A as it has the fewest neighbours (Figure 4).
2. $Ind = \{A\}$
3. Vertices C, D, E, F and I would all cause the fewest new vertices to become adjacent to Ind. Vertex C is added arbitrarily (Figure 4).
4. Vertices F and I would both cause the fewest new vertices to become adjacent to Ind. Vertex F is added arbitrarily (Figure 4).
5. Vertex I is added to Ind as it is the only vertex available to add (Figure 4).
6. Ind is $\{A, C, F, I\}$ after the function $addnodes$ completes.
7. The first vertex that is not in Ind, and is only adjacent to one vertex in Ind, is D. D is swapped with C, and $addnodes$ is called with $Ind = \{A, D, F, I\}$ (Figure 4).
8. No additional vertices can be added.
9. The size of Ind has not increased so $maxSet$ is still $\{A, C, F, I\}$.
10. The next vertex that is not in Ind, and is only adjacent to one vertex in Ind, is E. E is swapped with C, and $addnodes$ is called with $Ind = \{A, E, F, I\}$ (Figure 4).
11. No additional vertices can be added.
12. The size of Ind has not increased so $maxSet$ is still $\{A, C, F, I\}$.

13. No more vertices can be swapped so the final independent set is $\{A, C, F, I\}$.

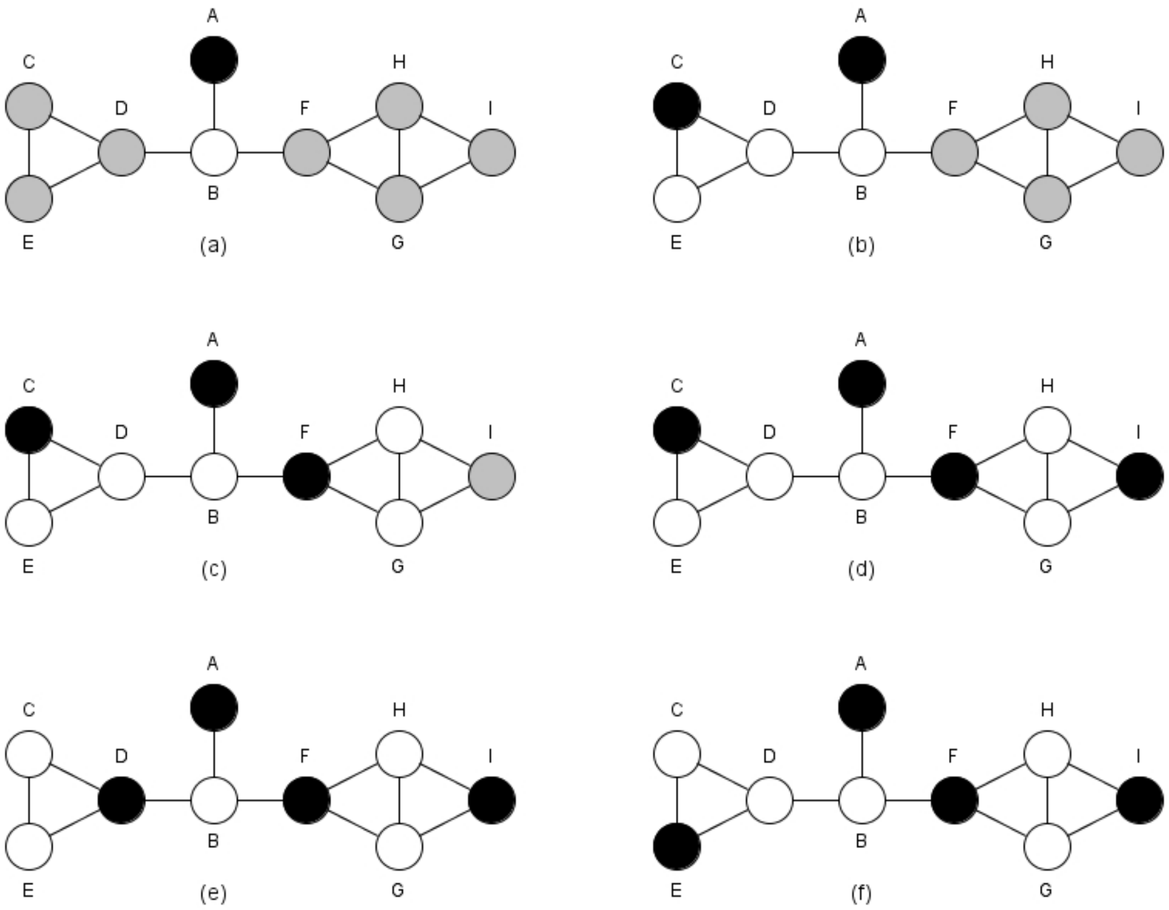
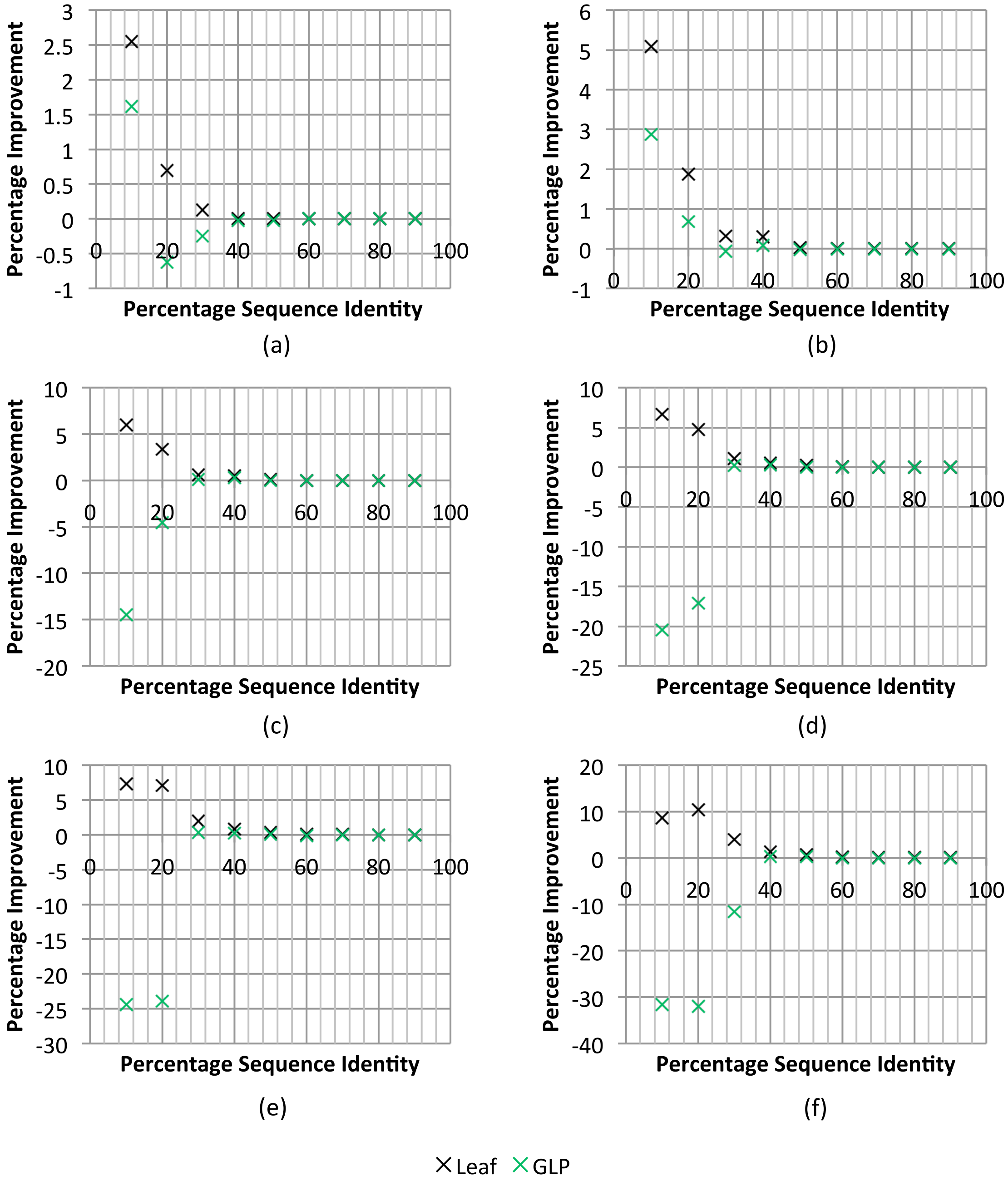


Figure 4: The progress of execution of the FIS algorithm on the graph seen in Figure 1. Black vertices are in the independent set being generated, white vertices are the vertices adjacent to the independent set and grey vertices are those that are still to be decided upon. Each graph corresponds to the results of one the execution steps of the FIS algorithm. (a) corresponds to step 1, (b) to step 3, (c) to step 4, (d) to step 5, (e) to step 7 and (f) to step 10.

Supplementary Information 3 Comparisons of Leaf and GLP algorithms to PISCES

Maximising the Size of Non-Redundant Protein Data Sets Using Graph Theory

Simon C. Bull, Mark R. Muldoon and Andrew J. Doig



The percentage improvement over PISCES shown by the Leaf and GLP algorithms. Graph (a) is for the datasets of 100 proteins, (b) for the datasets of 250 proteins, (c) for the datasets of 500 proteins, (d) for the datasets of 1000 proteins, (e) for the datasets of 2000 proteins and (f) for the datasets of 5000 proteins.

Sequence I	Mean Degr	Number of	Number of	Largest Con	Mean Comp	Mean Degr	PISCES		Leaf		FIS		NeighbourCull		VSA		BlastCuller	
							Time	Removed	Time	Removed	Time	Removed	Time	Removed	Time	Removed	Time	Removed
10	200.24191	18395	492	16972	37.388211	216.82218	55.878217	15178	2472.3061	14615	4950.7864	14666	577.32156	14651	1110.1182	14764	180.52198	14673
20	76.557668	18251	593	16383	30.777403	84.908136	46.546527	14551	1486.1758	13608	946.36902	13679	233.79955	13671	521.21604	13886	153.17603	13710
30	34.387357	15851	1436	11190	11.038301	46.860947	44.014081	11244	387.23044	10395	411.48527	10455	88.315359	10451	179.92516	10657	75.732308	10489
40	16.335196	11993	2538	3106	4.7253743	48.515132	43.083674	7829	21.287985	7408	26.618835	7419	6.0416175	7422	12.850922	7505	4.3409924	7440
50	6.0211939	8493	2403	680	3.5343321	23.973529	42.619561	5324	0.2455121	5082	0.6624101	5084	1.0683879	5087	0.5708436	5122	0.2349874	5097
60	4.5996076	5607	1753	232	3.1985168	11.887931	42.549825	3480	0.0702698	3364	0.0914535	3367	0.8343557	3365	0.1296872	3377	0.0653191	3367
70	4.5456516	3691	1192	74	3.0964765	73	42.7326	2282	0.0378185	2215	0.0271365	2215	0.7951804	2215	0.0708645	2221	0.0363025	2218
80	4.9652865	2391	800	73	2.98875	66.90411	42.756811	1488	0.0251	1450	0.016365	1450	0.1557584	1450	0.0483793	1453	0.0237436	1450
90	3.9198876	1423	476	54	2.9894958	12.62963	42.809867	885	0.028548	862	0.0101676	863	0.0412472	862	0.0216718	863	0.0128449	863

Sequence I	Mean Degr	Number of	Number of	Largest Con	Mean Comp	Mean Degr	PISCES		Leaf		FIS		NeighbourCull		VSA		BlastCuller	
							Time	Kept	Time	Kept	Time	Kept	Time	Kept	Time	Kept	Time	Kept
10	200.24191	18395	492	16972	37.388211	216.82218	55.878217	5073	2472.3061	5636	4950.7864	5585	577.32156	5600	1110.1182	5487	180.52198	5578
20	76.557668	18251	593	16383	30.777403	84.908136	46.546527	5700	1486.1758	6643	946.36902	6572	233.79955	6580	521.21604	6365	153.17603	6541
30	34.387357	15851	1436	11190	11.038301	46.860947	44.014081	9007	387.23044	9856	411.48527	9796	88.315359	9800	179.92516	9594	75.732308	9762
40	16.335196	11993	2538	3106	4.7253743	48.515132	43.083674	12422	21.287985	12843	26.618835	12832	6.0416175	12829	12.850922	12746	4.3409924	12811
50	6.0211939	8493	2403	680	3.5343321	23.973529	42.619561	14927	0.2455121	15169	0.6624101	15167	1.0683879	15164	0.5708436	15129	0.2349874	15154
60	4.5996076	5607	1753	232	3.1985168	11.887931	42.549825	16771	0.0702698	16887	0.0914535	16884	0.8343557	16886	0.1296872	16874	0.0653191	16884
70	4.5456516	3691	1192	74	3.0964765	73	42.7326	17969	0.0378185	18036	0.0271365	18036	0.7951804	18036	0.0708645	18030	0.0363025	18033
80	4.9652865	2391	800	73	2.98875	66.90411	42.756811	18763	0.0251	18801	0.016365	18801	0.1557584	18801	0.0483793	18798	0.0237436	18801
90	3.9198876	1423	476	54	2.9894958	12.62963	42.809867	19366	0.028548	19389	0.0101676	19388	0.0412472	19389	0.0216718	19388	0.0128449	19388

# Proteins	20251

Supplementary Information 5 Results from Culling PDB Data Sets

Maximising the Size of Non-Redundant Protein Data Sets Using Graph Theory

Simon C. Bull, Mark R. Muldoon and Andrew J. Doig

% Maximum Sequence Identity	Minimum Resolution	Maximum R-Factor	#Proteins from PISCES	#Proteins from Leaf	% Improvement
20	1.6	0.25	1886	2021	7.2
20	1.8	0.25	2954	3214	8.8
20	2.0	0.25	4030	4459	10.6
20	2.2	1.0	4640	5179	11.6
20	2.5	1.0	5346	5962	11.5
20	3.0	1.0	5922	6577	11.1
25	1.6	0.25	2276	2415	6.1
25	1.8	0.25	3677	3967	7.9
25	2.0	0.25	5089	5570	9.5
25	2.2	1.0	5910	6518	10.3
25	2.5	1.0	6822	7569	10.9
25	3.0	1.0	7525	8367	11.2
30	1.6	0.25	2676	2772	3.6
30	1.8	0.25	4469	4699	5.1
30	2.0	0.25	6360	6765	6.4
30	2.2	1.0	7492	7986	6.6
30	2.5	1.0	8713	9337	7.2
30	3.0	1.0	9615	10337	7.5
40	1.6	0.25	3182	3259	2.4
40	1.8	0.25	5604	5778	3.1
40	2.0	0.25	8337	8612	3.3
40	2.2	1.0	10029	10392	3.6
40	2.5	1.0	11872	12338	3.9
40	3.0	1.0	13219	13762	4.1
50	1.6	0.25	3524	3567	1.2
50	1.8	0.25	6308	6414	1.7
50	2.0	0.25	9571	9744	1.8
50	2.2	1.0	11657	11885	2.0
50	2.5	1.0	13897	14210	2.3
50	3.0	1.0	15584	15937	2.3
60	1.6	0.25	3704	3743	1.1
60	1.8	0.25	6737	6830	1.4
60	2.0	0.256	10336	10491	1.5
60	2.2	1.0	12679	12865	1.5
60	2.5	1.0	15249	15502	1.7
60	3.0	1.0	17221	17523	1.8
70	1.6	0.25	3845	3874	0.8
70	1.8	0.25	7069	7136	0.9
70	2.0	0.257	10927	11046	1.1

70	2.2	1.0	13469	13625	1.2
70	2.5	1.0	16282	16502	1.4
70	3.0	1.0	18463	18727	1.4
80	1.6	0.25	3979	4000	0.5
80	1.8	0.25	7341	7401	0.8
80	2.0	0.256	11416	11535	1.0
80	2.2	1.0	14128	14288	1.1
80	2.5	1.0	17154	17382	1.3
80	3.0	1.0	19535	19828	1.5
90	1.6	0.25	4112	4132	0.5
90	1.8	0.25	7650	7716	0.9
90	2.0	0.25	12009	12156	1.2
90	2.2	1.0	14950	15133	1.2
90	2.5	1.0	18250	18501	1.4
90	3.0	1.0	20916	21213	1.4