# Diverse Protein Discovery in Large-Scale Datasets: A Machine Learning Perspective

**Jhadziah Pienaar**

Department of Computer Science
Aberystwyth University
Wales

September
2023

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science

Degree: MSc Data Science
Module: CSM9060
Supervisor: Dr Amanda Clare

**Abstract**

Proteins are indispensable to a multitude of cellular processes ranging from catalyzing metabolic reactions, transportation of molecules, structural support and numerous other functions [1]–[4].

As the fields of Bioinformatics, Genetics and Cell biology have advanced, so too have sequencing technologies, resulting in an unprecedented increase in volume of protein sequencing data. Processing this increased data volume has become increasingly difficult. One way bioinformaticians have attempted to tackle this issue is with protein embeddings which capture the functional, biochemical, and structural characteristics to provide a more computationally efficient representation of the protein.

This work was aimed at finding methods to identify the most distant proteins and most diverse subsets of proteins from large protein databases in a scalable and efficient way using a dataset of protein embeddings from SwissProt, data mining techniques and metaheuristics.

The problem of finding the most distant and diverse subset of proteins in a large dataset was formulated as a MaxSum maximum diversity problem and four metaheuristic algorithms were implemented and evaluated against a brute force MaxSum baseline.

Two of the implemented methods, alternative tabu search (IATS) and a version of the traditional tabu search ITSv1 were shown to be robust at returning subsets with the most distant proteins. While ITSv2 and a memetic tabu search TSME appeared to be better at finding diverse subsets. In the limited time available for this project, it was demonstrated that these metaheuristic approaches are able to provide optimal results.

## Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work, I understand and agree to abide by the University's regulations governing these issues.

**Name:** Jhadziah Pienaar
**Date:** 28/09/23

## Consent to share this work

- By including my name below, I hereby agree to this thesis being made available to other students and academic staff of the Department of Computer Science, Aberystwyth University.

**Name:** Jhadziah Pienaar
**Date:** 28/09/23

**Acknowledgment**

Thank you to Cory Thomas for teaching and helping me with the supercomputing cluster, guiding me with the code parallelization, GPU acceleration and submitting jobs to the computing cluster. Thank you to my supervisor Amanda for the support and help throughout the project and to my mother Suzana for taking the time to read over the final product.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background and Motivation

### 1.1.1 Background

Proteins are indispensable to a multitude of cellular processes ranging from catalyzing metabolic reactions, transportation of molecules, structural support and numerous other functions [1]–[4]. The diversity in roles proteins play is as vast as the variety of proteins themselves.

The analysis of protein data can help us to understand organisms, and to identify intricate relationships between proteins. It helps in gaining insight into the structure and function of proteins and can lead to potential biomedical discoveries that are significant in understanding and treating disease as well as improving patient outcomes.

As the fields of Bioinformatics, Genetics and Cell biology have advanced, so too have sequencing technologies, resulting in an unprecedented increase in the volume of genetic and protein sequencing data. With the increase in data volume, computational cost has risen and processing has become increasingly difficult. Advanced data mining techniques have therefore become an essential tool to extract meaningful information from these large-scale datasets. Additionally, finding ways to efficiently search through large protein datasets can contribute to the wider Bioinformatics community by introducing tools and methodologies that can be applied to other large datasets.

### 1.1.2 Embeddings

Embeddings are a relatively recent development in the field of machine learning and natural language processing (NLP) They are semantic representations of high dimensional data in a lower dimensional vector space, where each dimension is a representation of a feature or attribute of the original data [5]–[7]. Embeddings can therefore be seen as a way of mapping discreate variables as continuous vectors where the more similar the data is in the vector space, the more similar it is in the original data [7]. The higher the dimensions of the embeddings, the better the accuracy of the representation of the data because more features have been captured by the embedding. However, this means increased computational requirements for processing the data [5][6].

Embeddings are created by Neural networks or deep learning models [7]. The process of learning embeddings involves training a neural network to map each data point to its corresponding vector representation in the lower dimensional space. During the training process, the embeddings are adjusted so that similar data points have similar embeddings, and dissimilar data points have dissimilar embeddings. This is achieved through backpropagation, where the model updates the weights of the embedding matrix based on the loss function [6][8].

### 1.1.3 Protein embeddings

Embeddings have been adopted in other machine learning and deep learning applications beyond NLP. In bioinformatics, and Computational biology, protein embeddings capture the functional, biochemical, and structural characteristics of proteins, providing a more computationally efficient representation of proteins. Even though the protein embeddings are a lower dimensional representation of the original protein data from which they are derived, these vectors still tend to be very large, therefore, efficient algorithms that can locate similar or dissimilar proteins in a large dataset are very desirable [8].

### 1.1.4 Motivations

Finding the most diverse or similar proteins has many benefits. For example, in examining highly diverse proteins there is the potential to discover otherwise unknown functionality as well as different evolutionary paths. Identifying the most diverse proteins may also lead to identifying proteins in particular diseases and may open possibilities to different therapeutic targets. Having methods of finding the most diverse proteins may also benefit disease diagnosis by finding different

proteins in biological samples. Furthermore, diverse proteins may have applications in enzyme engineering, vaccine development, synthetic biology, agriculture and future bioinformatics and algorithm development.

## 1.2 This Project

The work in this dissertation project was focused on finding the subset of diverse proteins from a large dataset of protein embeddings [9] in an efficient and scalable way using machine learning algorithms and metaheuristics. This work was based on the methods detailed in [10]. These machine learning algorithms have the potential to efficiently identify aspects of large datasets that would otherwise be too computationally expensive to attempt and are bound to increase the rate at which important discoveries in the field are made. Not only does this work have the potential to lead to discoveries in this field but can also be useful in other diversity problems such as establishing viable systems in ecology, creating diverse working teams, product design and the placement of undesired facilities, among other diversity problems [10] based on large and complex datasets.

### 1.2.1 The algorithms

The algorithms implemented for this work include two variations of the Iterative tabu search (ITSv1 and ITSv2), an alternative iterative tabu search (IATS) and a tabu search based memetic algorithm (TSME). They were compared and assessed against the baseline brute force max sum calculation of protein distance in vector space.

# Chapter 2

# Aims and Objectives

This project had two main aims. The first is to assess meta-heuristic algorithms' (IATS, ITSv1, ITSv2, TSME) ability to find the subset of most diverse protein embeddings in the entire Swiss-Prot dataset and secondly to assess and compare these algorithms' ability to return diverse subsets of embeddings.

**The objectives are as follows:**

1. Implement a brute force calculation of the Euclidean distance MaxSum for all protein embeddings.
2. Assess the scalability of the brute force calculation.
3. Investigate and research suitable heuristic algorithms to solve the maximum diversity problem (MDP).
4. Implement heuristic and metaheuristic methods to find efficient ways of finding diverse subsets of proteins in the embedding space.
5. Evaluate the metaheuristic algorithms ITS, IATS, TSME on their precision, speed and scalability.
6. Visualise the results.
7. Compare the subsets of proteins found.

# Chapter 3

# Literature review

The work in this dissertation is based on finding the most diverse proteins in an embedding space. This can be formulated as a Maximum Diversity Problem (MDP). Although MDP has been successfully applied in Bioinformatic settings, for example, for selecting diverse subsets of unannotated enzymes, identifying small sub-populations of cells, identify the minimum potential energy of macromolecules, it is a very new area of research with limited applications overall [11]–[13]. At the time of writing, there appeared to be little to no applications of the MDP problem to identify the most diverse proteins using embeddings. This literature review will briefly introduce the Maximum Diversity Problem, discuss its development in more detail, and will then describe the strategies adopted by researchers to find efficient and scalable near optimum solutions to the diversity problem.

## 3.1 The Maximum Diversity Problem

The maximum diversity problem (MDP) is a well-known combinatorial optimisation problem that involves finding a subset of items from a larger sample that maximises the dissimilarity between the selected items in the subset [10]. This problem is often encountered in fields such as computer science, data mining, operational research and has many applications to both real world and simulated problems. For example, solving problems in genetics, the placement and location of facilities, decoding macromolecules, resource allocation, amongst others [10]–[12].

Early work on the MDP problem began in the late 70s where it developed out of the earlier work on the p-dispersion problem from the operations research and facility location fields. The p-dispersion problem is also a combinatorial optimisation problem, like the MDP, it involves the selection of a subset of locations

from a given set of locations, with the aim of maximising or minimising a given objective, though often related to dispersion or coverage. This research later led to the development of the first MaxMin (MMDP) and MaxSum (MDP) models in the late 80s and early 90s [10], [14]–[16].

These traditional models were shown to be NP-hard by a variety of researchers. This means that there is no known polynomial-time algorithm to solve the problem optimally and therefore various approximation algorithms, heuristics and meta heuristics have been proposed to both extend and find efficient and scalable near optimum solutions to the diversity problem for a variety of settings [10].

Some of the proposed meta heuristics include but are not limited to greedy algorithms, greedy randomised adaptive search procedure (GRASP) methodologies, genetic algorithms (GA), local and global search algorithms [10], [17], [18]. Additionally, alternative MDP models have also been proposed and extended, such as the MinDiff, MaxMean dispersion problem, Max-cap/dis, sliced chain max sum among others [10], [11], [19]. However, much of the research on MDP had been done on small datasets of 2000 elements or less [20].

### 3.1.1 MaxSum MDP

In the context of the MDP the max sum is used to maximise the sum of distances between the elements in a subset, subject to the specific criteria or constraints. The objective of the MDP is thus to achieve selection of diverse items that maximise this sum [10]. The MaxSum model solves this problem by maximising the objective function over the variables. A dissimilarity metric such as Manhattan, Euclidean, Cosine similarity or Hamming distance quantifies the dissimilarity between the items. The constraints can be anything from the number of selected items in a subset to the time taken for the subset to be selected. The objective function, variables and constraints can be tailored and customised to suit the specific applications of the max sum MDP. Formally the Max sum can be expressed as:

$$\text{Maximize} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \text{distance}(x_i, x_j) \tag{3.1}$$

where:

$n$ is the number of embeddings,

$x_i$ is the embedding for the $i$-th protein,

distance$(x_i, x_j)$ is the Euclidean or other distance between embeddings $x_i$ and $x_j$.

The MaxSum MDP is particularly computationally expensive (NP-Hard) for large solution spaces because a brute force calculation of the MaxSum requires the calculation of a distance metric between all items in the solution space followed by the summation of these measures before selection of k items with the maximum distance. Therefore, as the solution space increases in magnitude, so does the time and computational resources required. As a result, many different approaches for optimising the problem have been investigated. The majority of which involve meta-heuristics.

## 3.2 Meta-Heuristics for the Maximum Diversity Problem

Metaheuristics are higher level algorithms that are designed to optimise problems that are too computationally expensive and NP hard [21]. They aim to find the best near optimal solution in a 'small enough' time frame and therefore do not guarantee the global optimum solution to the problem [22]. Metaheuristics fall into a few categories including local searches, constructive algorithms, population based and hybrid metaheuristics [22]. These methods will be discussed below:

### 3.2.1 Iterative Greedy search algorithms

One metaheuristic approach to improving the efficiency of search algorithms is to implement an Iterative greedy (IG) algorithm [20]. Iterative greedy algorithms create a series of solutions through an iterative process in which, during each iteration a greedy constructive heuristic is applied in two phases; a construction and destruction phase where solution components are either added or removed, respectively, to create a new solution [20]. The new solution is then compared to previous solutions and the objective function, or another acceptance criterion is used to assess whether it should replace the previous solution [20]. In [20] an IG algorithm was compared to the best performing Variable Neighbourhood search (VNS) model of the time. Comparisions were made across a variety of sample sizes (2500-5000 instances) and it was demonstrated to be a high performance algorithm that was comparable to the other state-of-the-art algorithms [20].

### 3.2.2 Greedy Randomised Adaptive Search Procedures

Several approaches based on greedy randomised adaptive search procedures (GRASP) methodologies have also been proposed for solving the MDP [10]. These approaches range from hybrid methods combining GRASP with path rekindling and with various other local searches [10], [23].

GRASP is a hybrid constructive metaheuristic [22] that is an iterative process consisting of a constructive phase where the neighbourhood is explored by replacing one element in the solution with another that is not in the solution (the best from the neighbourhood), resulting in the best solution over 'n' iterations [23]. The generation of an initial solution is done by estimating the gain of individual solutions' element using a greedy function, adding the element to a list called restricted candidate list (RCL), with their estimated value, and randomly choosing an element to include in the solution [23]. The local search is used to improve the generated solution because the solution may not be locally optimal [23].

The effectiveness of three GRASP methods were compared in [23], Gosh's proposed GRASP method for MDP [24], K largest distance (KLD) algorithm and KLD2 which mainly differ in how the RCL list is constructed [23]. The comparison showed that the three methods found the same solutions in 39/80 tests [23]. KLD and KLD2 found better solution to Gosh's algorithm overall, with better solutions being found 31, 25 and 7 times respectively [23]. To further improve the effectiveness of GRASP methodologies path rekindling was added [23], [25].

### 3.2.3 Tabu Search and Iterative Tabu search

An additional popular metaheuristic that has been implemented for solving MDP is Iterative TABU search (ITS) [10], [26], [27]. Iterative TABU search is an extension of the standard TABU local search metaheuristic algorithm that incorporates iterative exploration to move from one solution to the next [18], [26], [28].

One of the main components of TABU search is the use of both short- and long-term memory to inform future search moves to find the optimum solution [26]. TABU search works by randomly selecting an initial solution, comparing the initial solution to other solutions in the neighbourhood and storing the solution in a tabu list [26], [28]. The next iteration of the tabu search then finds a solution in a new neighbourhood, finds the best solution in the neighbourhood, appends to the list and then compares to the previous entered solutions from previous neighbourhoods.

The tabu list stores both the 'best' or nearly better solutions in the 'recency-based short-term memory (RSM)' list and the frequency at which the 'better' solutions are viewed are stored in the 'frequency-based long-term memory (FRM)' list [26]. Therefore, as the iterations continue, older solutions with the lowest frequency are replaced [26]. Furthermore, the tabu list is used to intensify and diversify the search by intensifying the search around areas in the solution space where good solutions have been found in the past and diversifying to search the remaining solution space to avoid getting stuck in the local optima [26]. A stopping criterion, such as number of iterations, is also part of the algorithm and terminates the search.

TABU search was shown to be able to efficiently find high quality solutions with larger solution spaces n=2000 [27]. TABU search was also found to be more robust compared to the GRASP methodology in [29]. Iterative tabu search has been found to be one of the best performing metaheuristics for solving MDP in the past (2000-2010) [10], [30].

### 3.2.4 Variable Neighborhood Search

One of the other best performing metaheuristics of the same period for solving the MDP problem is variable neighbourhood search (VNS), in particular B-VNS [10], [30]. VNS is an optimisation technique designed for combinatorial optimisation problems that focuses on neighbourhood change to escape local minima [31], [32]. VNS aims to find the best solution by changing neighbourhoods and conducting local searches within them based on the following assumptions [31]–[33]:

- The local optimum of one neighbourhood may not be the optimum of another neighbourhood or globally in the solution space.
- A global optimum is a local optimum respective to all other neighbourhoods.
- That in many problems the majority of local optima are close to one another.

VNS consists of three phases that are repeated until the designated stopping criteria is met [33]. The three stages include a shaking procedure, improvement procedure and neighbourhood change [33]. The shaking procedure is where a current solution is generated in a predefined neighbourhood [31], [33]. The neighbourhood change procedure guides the VNS exploration throughout the solution space and is also where the decision on whether the current solution should replace the previous neighbourhood's solution [31]–[33]. The improvement procedure includes local

searches within neighbourhoods to find the local optimum and variable neighbourhood descent which exploits the third assumption previously mentioned [33]. In particular, the variable neighbourhood descent sequentially explores neighbourhoods to improve the current solution [33]. For both parts of the improvement procedure a variety of methods can be used.

### 3.2.5 Genetic and Memetic algorithms

Other methods for solving the MDP include memetic algorithms [10]. To understand memetic algorithms, their predecessor, the genetic algorithm must first be explained. Genetic algorithms, as the name suggest, are based on a simplified version of Darwinian evolution that works by using an evolutionary process to maximise or minimise a given function [34]–[36]. The objective function in the case of the maximum diversity problem could be the MinDiff, MaxMean or MaxSum of a given distance metric. During each iteration the algorithm goes through stages of initial population generation, fitness score evaluation, solution selection, reproduction/cross-over and then mutation to find the optimum or near optimum solution for the solution space [34]–[36].

A population is a fixed sized, randomly selected subset of the entire solution space and contains many possible solutions or 'chromosomes' [35], [36]. The selection phase is characterised by the selection of two 'parent' solutions, chosen based on their fitness scores from the population. Solutions with higher fitness scores have a greater likelihood of being selected [34], [35]. The crossover phase takes the two parent solutions and crosses them over so that the new children have a mix of the two parent solutions' components or 'genes', whereas the mutation phase will randomly change a component of the current solution and is determined by the mutation rate parameter [34]–[36]. Although genetic algorithms can simultaneously explore different regions of the solution space they can get stuck in the local optima and then prematurely converge which results in sub-optimal solutions and areas of the solution space that may yield better solutions not being explored [34], [37], [38].

Memetic algorithms are an extension of genetic algorithms that implement a local search technique to mitigate the likelihood of premature convergence and to exploit the best features of both evolutionary and local search methods [34]. The main difference in memetic algorithm implementation compared to the traditional genetic algorithm is the addition of a local search algorithm after the combination phase [26]. The aim of the local search is to further improve the individual quality

of the offspring solutions by taking the current offspring solution and comparing it to other solutions in the neighbourhood and returning the best solution from the neighbourhood [39], [40]. The final step in the memetic algorithm is updating the population, which is determined by a decision on whether the solution should come from the current population or whether the population should be replaced [39], [40]. Genetic algorithms can be considered population-based meta-heuristics, however, memetic algorithms reflect a more hybrid approach [22].

A comparison of the performance of memetic and genetic algorithms at solving different combinatorial NP-hard optimisation problems known as the 'cryptanalysis of simplified data encryption standard' found that their memetic algorithm slightly outperformed the genetic algorithm in both reducing the variability of the results and run time especially in larger solution spaces [34]. Additionally memetic algorithms have been shown to be a framework suitable for a range of different optimisation problems including the MDP [40], [41].

### 3.2.6 Hybrid Memetic algorithms

Hybrid memetic algorithms have shown success at improving the scalability and efficiency of solving the MDP [10], [41], [42]. The two hybrid memetic algorithms that are considered the current best metaheuristic models for the MaxSum MDP overall, are opposition-based memetic algorithm (OBMA) and a TABU search memetic algorithm (TSME) [10]. To improve memetic algorithms in the context of MDP Opposition-based memetic algorithms combine opposition-based learning (OBL) with the traditional memetic algorithm framework [42]. OBL is a machine learning approach based on the following assumptions [43]:

- When looking for solution x of a given problem, we typically make an estimate Xr.
- Xr can be a random point, solution etc.
- Xr can be close or far from the optimal solution because without prior knowledge we cannot be sure that Xr is the best guess.
- In the worst-case scenario, the optimal solution may be the opposite of Xr.
- Thus the opposite of Xr is worth exploring.

Therefore OBL works by finding the both f(Xr) and the opposite value of Xr , f(Xo) in every iteration and then comparing the two to find out which one is more optimal based on the chosen evaluation function and retains the more optimal

value [43]. The base idea of OBMA is to double the search space and improve the speed at which the fittest member is found by finding both the chromosome and anti-chromosome, both of which could be a poor or good fit for the solution, comparing their optimality to increase the likelihood that the best chromosome is found in a shorter time [42], [43].

There is conflicting evidence on which of the two hybrid memetic algorithms is a better model overall. A recent review on meta-heuristics for solving the MDP has shown that the TSME has a slightly improved standard deviation compared to OBMA, however, OBMA performs best overall [10]. Additionally, OBMA has been shown to outperform in all instances when competing with some of the state-of-the-art algorithms such as ITS, scatter search, VNS and a memetic algorithm (MAMDP) on 40 instances of 4 datasets consisting of 2000-5000 items [42]. However, this study did not compare OBMA to TSME and therefore it is still unclear whether it would have outperformed TSME in these instances. On the other hand, a TSME was shown to outperform an opposition-based memetic algorithm at solving the MDP problem at 20 benchmark instances on average [41].

## 3.3 Other Metaheuristics

Additional metaheuristics that could be applied to the MDP are swarm intelligence metaheuristics such as Ant colony optimisation (ACO), particle swarm optimisation (PSO) and other local search algorithms such as simulated annealing.

### 3.3.1 Ant Colony Optimisation

ACO is based on the foraging behaviour of ants where the shortest possible path to food is highlighted by the number of pheromones left on the ground and as more ants take the same path the number of pheromones increases [44][45]. In ACO the potential solutions are generated randomly, weighted with 'pheromones' based on their optimality as assessed by an evaluation function and repeated until the optimum solution is found, or a termination condition is met [44], [45]. Therefore, ACO is considered a constructive metaheuristic algorithm [22]. ACO has been shown to be particularly good for combinatorial optimisation problems [44], [45].

### 3.3.2 Particle Swarm optimisation

PSO is based on the swarming behaviour of birds when looking for food, particularly, how the whole flock benefits from the discoveries of individual birds [46].

In the context of PSO, potential solutions are represented as groups of particles [46], [47]. Each particle has attributes including a position, velocity and fitness value, while also keeping track of its best position and fitness value [48]. Particles adjust their position within the solution space to find their own personal best solutions, and through collaboration by sharing their personal bests. Collectively, they identify the global best solution by sharing their personal bests [47]. PSO is easily parallelized and is an efficient global solution making it best suited to discrete optimisation problems [49].

### 3.3.3 Simulated annealing

Simulated annealing is a very well-known local search metaheuristic that is based on the annealing process of crystalline solids [22]. It is also an example of a stochastic hill climbing local search meaning that as the algorithm runs in search of the best solution (global optimum), it will continue to make moves until a better solution is found, otherwise, it keeps the current solution [50], [51]. Stochastic hill climbing means that the movements to a new solution are random [51]. The algorithm simulates the behaviour of atoms in heated crystalline solids. At a high temperature, the atoms shift unpredictably. As the system cools, impurities are eliminated. Similarly, the algorithm modifies the initial solution by introducing controlled random changes within set constraints. If a new solution/position proves superior, it replaces the existing one. Over time, the magnitude of change reduced until a global solution is found [50], [52], [53].

Simulated annealing differs from other hill climbing local search algorithms because it sometimes accepts a worse solution [50]. Simulated annealing was a popular choice for solving maximum diversity problem during 1980-2000, however its applicability has been surpassed by the emergence of more effective metaheuristics [10].

## 3.4 Discussion

We have seen that the problem of finding the most diverse proteins in the embedding space can be formulated as a Maximum Diversity Problem (MDP). However, because the MDP is NP-Hard, to improve the scalability and efficiency of solving the MDP, a variety of strategies have been proposed by researchers. These strategies include numerous metaheuristics and Memetic approaches, of the papers reviewed here, the best performing algorithms are VNS, ITS, TSME and OBME. It's important to note that all metaheuristics do not ensure the global optimum

solution and are designed to find the most effective solution in less time with lower computational costs than would otherwise be possible.

It is also important to note that the experimentation with these algorithms, as reported by most of these papers, has only been carried out on small datasets (maximum 2000-5000 items). The datasets of protein embeddings used in this work is two orders of magnitude larger, with 569,508 instances. Furthermore, the embeddings themselves possess high dimensionality. The implications of the high dimensionality and the difference in dataset size can potentially be significant to the performance of any of the above-mentioned algorithms run on this dataset. This is because handling dataset of this size is more complex, the scale of the size poses memory and computational challenges that are sure to affect performance. Algorithms that perform well on small datasets might not scale optimally to handle the increased volume and dimensionality of the data. The behaviour and efficiency of the algorithms could also differ substantially. Most importantly, significantly more processing and memory capacity will be required to execute these algorithms on this dataset. This means that without sufficient computing capacity, running these algorithms may not be feasible.

In terms of algorithm selection for experimentation, OBME was not thought to be suitable for this problem because it was difficult to formulate what the 'opposite' protein embedding or solution should be. Because these embeddings are a lower dimensional representation of the actual high dimensional nature of the protein data, what the embedding attributes actually represent, is unknown. Perhaps given more time, a formulation might be found but given the time constraints for this project, this algorithm was not selected.

The VNS algorithm was also not selected for implementation because the dynamic neighbourhood changing adds a layer of complexity compared to the ITS that may impact more on the computational and memory requirements. Bearing all of this in mind, the metaheuristics implemented for experimentation on this diversity problem and dataset were the ITS, TSME because of their relative simplicity to implement.

# Chapter 4

# Method

The experimental approach taken with the selected algorithms for solving the MDP on this dataset, was to first implement a brute force max sum function to output all protein embeddings sorted by their distance. This output then served as the baseline for comparison against the metaheuristic and memetic approaches.

This section provides details about the experimental settings, evaluation and validation, data, computational resources, and the implemented algorithms.

### 4.0.1 SwissProt Embedding Dataset

The protein embeddings used in this work are a representation of the functional and structural properties that have been encoded from the amino acid sequences [43]. The dataset is made available as a 1.5GB H5 file that contains open-source protein embedding data curated by UniProt and can be found on the UniProt web portal [42]. It consists of 569,508, protein embeddings that are 1024-dimensional. The H5 file stores the data as key, value pairs, where the key is the protein label, and the value is the 1024-dimensional embedding array. No cleaning of data was required except for one protein embedding with NaNs that was omitted.
An example of the input data can be seen in A.1.

### 4.0.2 Computational Resources

The meta-heuristic algorithms were run on an Acer predator 500 laptop with a 6-core intel(R) Core (TM) i7-8750H CPU @ 2.20GHz, GeForce RTX 2070 GPU and 32gb of DDR4 ram. Due to time and computational constraints the brute force calculation was run on a CPU cluster with 48 cores, 96 threads and 600gb of memory. Despite these high specifications, the estimated completion time for the brute force to return the top 1000 most distant proteins was still 1300 hours.

Therefore, the brute force had to be parallelized. After parallelization the full brute force returning all max sum Euclidean distances took 48 hours.

### 4.0.3 Packages

All code was developed on a windows system running Python 3.8 in accordance to PEP 8. **The packages used are as follows:**

- Pandas
- Numpy
- DataClassess
- Random
- H5py
- Matplotlib
- Seaborn
- Plotly
- Scipy spatial distance
- Sklearn for PCA
- Tqdm
- MPLcursors
- Time
- Umap
- Cuda
- Multiprocessing
- Ast
- Math
- collections for defultdict
- typing for list
- warnings

## 4.1 Data Exploration

Before developing the brute force and metaheuristic algorithms, the data was briefly explored. The shape and length of the extracted data was observed, Nans were extracted. Additionally, some experimentation with distance metrics was also carried out, this aided in the selection of which distance metric to use to calculate the distance between two embeddings. Three distance metrics were compared on a small subset of 100 embeddings and repeated 10 times. The distance metrics evaluated were Manhattan, Euclidean and Cosine similarity. On the small random samples, the Manhattan and Euclidean returned the same solutions whereas the cosine similarity returned different solutions. Therefore, the chosen distance metric was the Euclidean distance which measures the straight-line distance between two embeddings. It was selected because it is intuitive and simple.

Calculating the Euclidean distance was also experimented with. The first involved directly calculating the Euclidean distance by manually calculating the square root of the one embedding squared minus the other embedding squared and then summing all the distances together. The second used Numpy's Linear algebra norm to calculate the Euclidean distance which was summed after all iterations for one embedding. Both were equivalent and resulted in the same values.

```
# Calculate the Euclidean distance using numpy
distance = np.linalg.norm(embedding - other_embedding)
```

### 4.1.1 Visualising the embedding space

To attempt to visualise the embedding space both UMAP and PCA reduction techniques were used. Its important to note that this is only a 3D projection of the data and may not accurately reflect the what the embedding space actually looks like.

UMAP offers projections to lower dimensional spaces whilst attempting to preserve both the global structure and local neighbourhoods, however, due to its stochastic nature the results can vary between runs [54]–[56]. UMAP achieves this by utilising techniques such as creating an initial fuzzy topological representation based on the nearest neighbour descent algorithm of each point in the higher dimensional space, optimising the low dimensional embedding using stochastic gradient decent and using spectral embedding techniques [56].

PCA is one of the older and more widely used reduction techniques. PCA offers projection to lower dimensional spaces that can be directly related back to the original variables of the data and is more consistent than UMAP due to its linear nature [57], [58]. PCA aims to retain as much information as possible by transforming the data in the direction of the greatest variance however it may lose local structure for the very same reasons [58].
Figures 4.1 shows the PCA of the entire embedding space and figure 4.3 shows the UMAP interpretation. Figure 4.2 shows the PCA excluding all non enzymes and unknowns.

In order to make the plots more informative and easier to view, the protein embeddings were coloured according to enzyme classification number. Only the first number was taken from the enzyme classification.

**The Enzyme classification first numbers mean the following[59]:**

- EC 1: Oxidoreductases - enzymes that catalyze oxidation/reduction reactions.
- EC 2: Transferases - enzymes that transfer a functional group from one molecule to another.
- EC 3: Hydrolases - enzymes that catalyze the hydrolysis of various bonds.
- EC 4: Lyases - enzymes that cleave molecules without hydrolysis or oxidation.
- EC 5: Isomerases - enzymes that catalyze isomerization changes within a molecule.
- EC 6: Ligases - enzymes that join two molecules together, (often requires energy in the form of ATP.)
- EC 7: Translocases - catalyze the movement of ions or molecules across membranes.

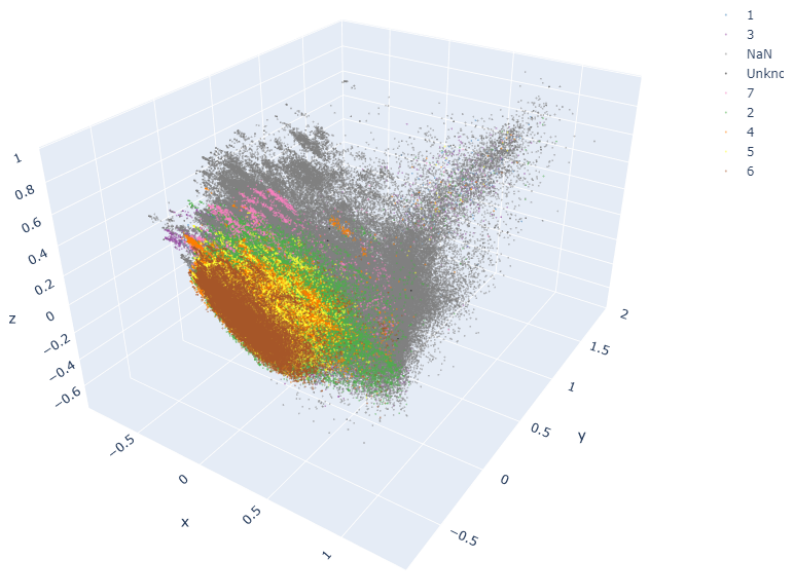


Figure 4.1: PCA plot of the entire embedding space in 3D coloured by Enzyme Classification number (EC). EC 1 = blue, EC 2 = Green, EC 3 = Purple, EC 4 = Orange, EC 5 = Yellow, EC 6 = Brown, EC 7 = Pink

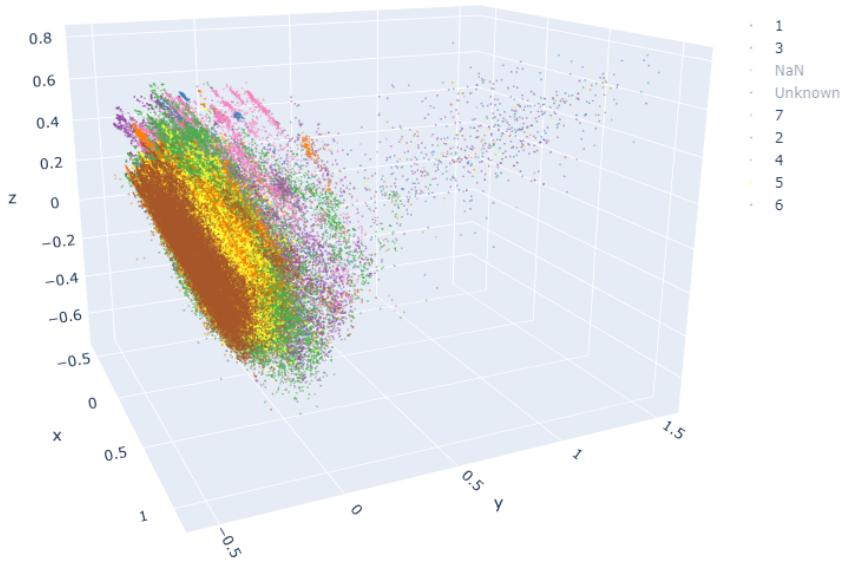


Figure 4.2: PCA plot of the entire embedding space in 3D coloured by Enzyme Classification number (EC) excluding all non enzymes and unknowns. EC 1 = blue, EC 2 = Green, EC 3 = Purple, EC 4 = Orange, EC 5 = Yellow, EC 6 = Brown, EC 7 = Pink

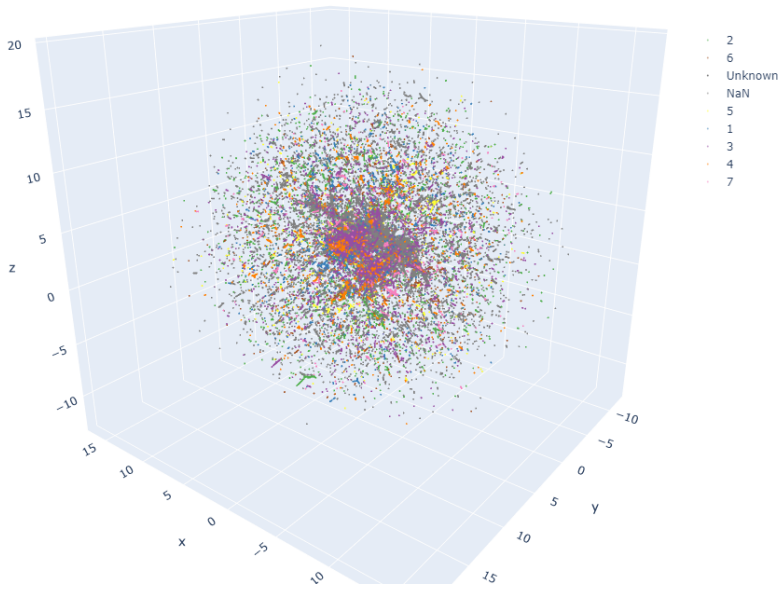


Figure 4.3: UMAP plot of the entire embedding space in 3D coloured by Enzyme Classification number (EC). EC 1 = blue, EC 2 = Green, EC 3 = Purple, EC 4 = Orange, EC 5 = Yellow, EC 6 = Brown, EC 7 = Pink

## 4.2 The Brute Force MaxSum

The brute force calculation takes one embedding and calculates the pairwise Euclidean distance between itself and every other embedding in the dataset and then sums these distances. Therefore, the higher the MaxSum distance an embedding has, the further away in vector space the embedding is to every other embedding in the dataset and hence, the more dissimilar it is.

### 4.2.1 Scalability testing

The brute force calculation was assessed for scalability on a set of samples from 0 to 10,000. Two separate runs using the brute force to calculate the max sum were assessed and can be seen in Figure 4.2.1. As sample size increases so does the computational time for the brute force calculation to run. This increase implies an almost cubic relationship between the dataset size and the increase in time.

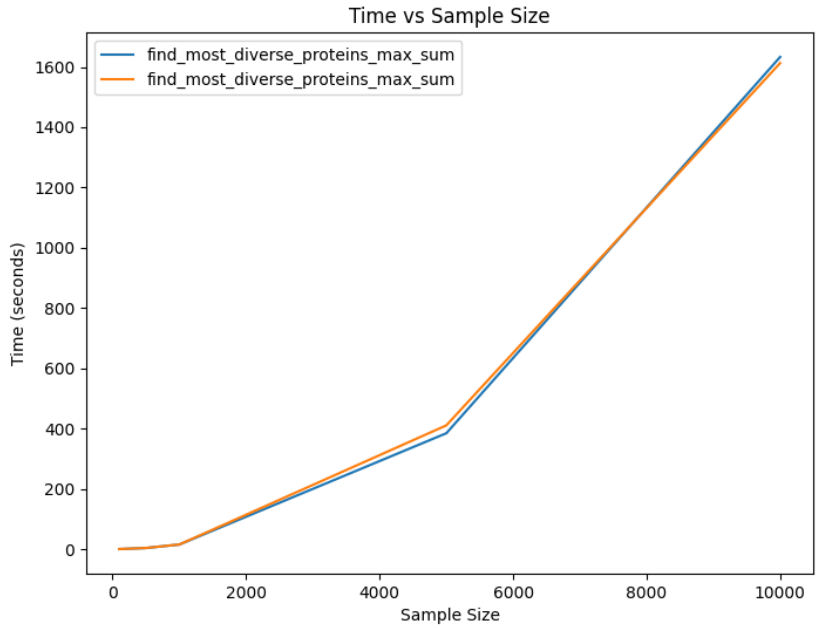


Figure 4.4: Plot showing the scalability of the brute force calculation of the pairwise euclidean distance over a set number of samples

### 4.2.2 Parallelisation and GPU acceleration

To speed up computation time for the brute force calculation, both GPU acceleration and parallelisation was explored. In the end, GPU assisted computation was not necessary as the parallelisation was efficient enough.

For parallelisation, the brute force code was altered so that the max sum function operated on chunks of the data as opposed to the full set. In other words, it still calculated the MaxSum pairwise Euclidean distance for each embedding against the full dataset, but it now did this with smaller chunks of embeddings.

The main multiprocessing algorithm works as follows: It initially converts the input dictionary to a list of tuples and then splits the tuples into chunks by dividing the dataset by the number of cores available. The Python multiprocessing library was used to create a pool of worker processes based on the number of available CPU cores by using 'mp.cpu_count()'. The MaxSum function to calculate the distances is then applied asynchronously using 'pool.apply_async' to each chunk of embeddings. All the results are then collected into a list of lists and the worker processes are closed. Finally, the results lists are flattened into a single list and then sorted in descending order and saved to a CSV file. The CSV only contains the protein (embedding) label and the max sum distance value in descending order.

The GPU accelerated version worked the same, except that the Euclidean distance calculation was separated from the max sum and uses the jit annotation to indicate that this function is to be accelerated via a CUDA capable GPU.

The parallelisation, GPU acceleration and sending of the job to the supercomputing cluster was done with the help and guidance of Cory Thomas.

## 4.3 The MaxSum Tabu Search Algorithms

Two types of the tabu search were investigated. The first approach is based on a more traditional tabu search and the second was an alternative tabu search.

From here onwards the tabu searches will be referred as ITSvX (iterative tabu search) and IATS (iterative alternative tabu search), respectively.

### 4.3.1 Common Key features

Both approaches take the embeddings as key value pairs (a dictionary) where the protein labels are keys, and their embeddings arrays are values.

Both algorithms' fitness metric is essentially a brute force calculation of the pairwise Max Sum Euclidean distance between embeddings of the given neighbourhood or subset of the embedding space. The aim of this fitness metric is to determine the overall distance of the selected proteins. To do this a list of protein labels representing the selected proteins is fed into the fitness function and for each protein, it iterates through the rest of the other proteins in the sublist and calculates the pairwise Euclidean distance between embeddings before calculating the sum of the pairwise distances. The x individual embeddings with the highest fitness (distance) will be returned as the current best solution from the neighbourhood search. Therefore, the fitness is a key metric to evaluate the quality of the solution generated by the ITS or IATS algorithm.

However, there is a difference in where the fitness calculation is implemented in ITS versus the IATS. In the ITS this is done within the ranking function and the fitness function just checks whether the current fitness value is greater than the current best and if so, it iteratively updates the fitness and the best subset and adds this to the tabu list. Whereas in the alternative TABU the fitness is calculated in the fitness function and the ranking just sorts the subset ready for

output The pseudo code for the fitness metric is shown below:

```
def max_sum_fitness(protein_labels):
    Initialize total_distance to 0

    For i from 0 to length of protein_labels - 1:
        Retrieve the embedding emb1 for protein_labels[i]
        For j from i + 1 to len of protein_labels:
            Retrieve embedding emb2 for protein_labels[j]
            Calculate distance between emb1 and emb2
            Add distance to total_distance

    Return total_distance
```

### 4.3.2 The Alternative Tabu Search

IATS (class name MaxSumTabuSearch) is an example of a modified traditional Tabu search algorithm and consists of a class names 'MaxSumTabuSearch' with the following parameters:

- 'Num_proteins': the desired number of most diverse proteins to return as output.
- 'max_iterations': total number of iterations for the TABU search.
- 'tabu_tenure': - how many iterations a solution remains in the tabu list.

The IATS model differs in a few ways from the typical traditional tabu search. Most importantly, IATS performs a form of global search by taking random samples from the entire dataset as pseudo neighbourhoods. The main idea here is that by breaking up the global solution space into random smaller samples it will increase the efficiency of the search.

IATS starts by randomly choosing a random 'best solution' (list of proteins) from the entire embedding space which later solutions will be compared to. This solution may be a good starting point but may equally be a very poor start point due to the random nature of the selection process. Then the iterative local searches begin with the random selection of pseudo neighbourhoods (list of proteins), and subsequent fitness evaluation of all embeddings within these neighbourhoods. Once the local neighbourhood has been explored, then a comparison is made between the current best global solution and the current best local solution. Once all iterations

are complete the best solutions ( a list of the most distant proteins) are returned. Additionally, it makes use a tabu list and tabu tenure to exclude already found solutions for a number of iterations to help reduce the chances of being stuck in a local optimum.

### 4.3.3 The Iterative Tabu searches

ITS (class name TradMaxSumTabuSearchVx) are an example of a traditional Tabu search algorithm with the following tunable parameters:

- 'Num_proteins': The desired number of most diverse proteins to return as output.
- 'max_iterations': total number of iterations for the TABU search
- 'local_iterations': the total number of local iterations for the local search
- 'local_sample_size': The number of embedding selected during the local search

The ITS search begins in the same way as the IATS by selecting a random subset to evaluate the local search against. These algorithms perform the local search on neighbourhoods that are generated randomly from the global subset. Similar to the IATS, the idea was to further break up the samples to improve efficiency of the search for the most distant and thus dissimilar protein. Where they differ from the IATS is in the neighbourhood generation.

Two variations of neighbourhood generation were explored. The first method (class name TradmaxSumTabuSearchV1) creates neighbourhoods selecting an initial random solution (list of proteins) from the global search space. It then generates a neighbourhood that is a list of lists (possible solutions), all the same fixed size. So given that for example, 10 proteins are selected as the initial local solution, the neighbourhood generation will create a list of 10 lists where each subsist is a slightly mutated copy of the initial list (One element is swapped out with a random selected protein from the global space).

The second method (class name TradmaxSumTabuSearchV2) doesn't create a new neighbourhood at each local iteration, instead it selects from the global space a random neighbourhood that is 5% of the total embeddings and explores this space in the local search and at each iteration, selecting a potential solution at random from the local space.

During the local searches, the fitness, i.e., the diversity, of each embeddings solution is calculated in the same way as the IATS, those lists with the highest fitness value are returned as the best solutions of that local search. The neighbourhood best is then compared to the current global best solution and replaces it if the fitness values is higher than the current best. Once all global iterations have been completed, the best solutions are ranked based on their diversities and the list of the most diverse proteins are returned as output.

TradMaxSumTabuSearchV2 should efficiently explore all possible solutions from the global neighbourhood and by using the two-tier approach (multi start global and local searched) should offer a better balance between exploration of the local spaces as well as more diverse areas of the global solution space, finding the most optimal solution overall.

During development the performance of the tabu searches were debugged and evaluated on small subsets of 100 – 5000 instances of the dataset and compared to a brute force calculation of this same subset for validation. For these runs, the brute force function output a dataFrame with all the sample proteins ranked by their max sum, therefore the index represented its diversity and ranking. A secondary function compared the outputted lists of 10 protein embeddings from the two tabu searches and returned whether they were in the brute force list and at what index (rank) the embedding label was.

### 4.3.4 The Memetic Tabu Search

The next approach to locating the most diverse proteins in a scalable and efficient way was the implementation of a memetic algorithm with a Tabu local search and genetic algorithm. The class MemeticGLs has the following tunable parameters:

- Dna_size: How many proteins in a solution
- Max_epochs: Maximum number of iterations
- Local_iterations:For local refinement, number of iterations
- Population size: How many individuals in a population
- Retain_percent: How many best individuals to carry over to the new population.

The memetic algorithm was implemented with a genetic algorithm that creates a new population of individuals at each epoch. The individuals in the population

each represent a solution (a list of potentially diverse proteins), their DNA being the list of proteins. A population is initially generated randomly from a subset that is selected from the global embedding space; this subset is the local space to explore. At each epoch, the population is evaluated for fitness, put through a selection process, and refined or forgotten.

Selection involves a number of strategies for carrying over the best individuals (solutions) to the next generation. The best individuals are selected for inclusion in the next generation without any changes, they are also selected for cross over with each other to explore the local space. Once crossed over, the newely created individuals are also added to the new generation. The next best set of individuals are selected for mutation. Here proteins are randomly swapped out of each individual's DNA. These mutations are randomly selected from the global space to add some diversity. The mutated individuals are then also added to the new generation. The next set of individuals lower down the rankings are selected for refinement, which is a set of mutations, this time selected from the local space. Because these mutations are taken from the local space, they are checked against the tabu list to ensure that no duplications are allowed in the new generation. These individuals are also added to the new generation. All the individuals with low fitness are forgotten.

Finally, the rest of the new population are created from a new randomly selected local space. The fitness of each individual in the population is calculated by the max sum of the pairwise Euclidean distances between proteins in the DNA. After all epochs have been completed the best individual is returned as well as the list of the ranked individuals.

## 4.4 Experimental Evaluation and Validation

### 4.4.1 Output data of the tested algorithms

Initially, before the brute force results on the complete dataset were computed, a smaller brute force ground truth was calculated on a subset of the data so that testing and validation could be carried out and so that some base parameters could be established for each algorithm.

These tests compared smaller sub-samples of the data. The tests did not return the actual most distant proteins from the entire dataset, instead of the most distant proteins from the subset were returned. This allowed the performance of the

algorithms to be tested against the brute force subset many times efficiently.

Once some base parameters were established and the output of the functions was as expected, the algorithms were run on the full dataset and the outputs compared against the full brute force outputs to determine the optimal parameter for each. Once the optimal parameters were found a further set of tests were run to identify if the returned proteins followed a pattern.

The brute force calculation on the full dataset was then used as the baseline validation for the experimentation. As previously mentioned, the output of the brute force calculation was a CSV file with the protein label and corresponding max sum Euclidean distances in descending order. Therefore, the index position of the protein label in the files serves as the pseudo ranking with the first protein label in the csv file being the most diverse protein. The output of the metaheuristic algorithms was then matched to the protein label and index from the brute force csv to find out how well the algorithm performed at returning the most diverse proteins.

All algorithms return a list of protein labels like this :

['Q9XDP0', 'Q9V5Z7', 'Q9TUX7', 'Q9SXS8', 'Q9PKV0', 'Q9NS84', 'Q9JHK7', 'Q9I9P7', 'Q9GPJ1', 'Q9FFX1', 'Q9F9L1', 'Q9BZM1', 'Q96BJ8', 'Q92847', 'Q8YUT2', 'Q8ERT7', 'Q8EMI3', 'Q89FP2', 'Q80TS5', 'Q7YR44']

Before entering the comparison with baseline which outputs the ranks like this:

- Protein Q9XDP0 is in the validation set at index 202351.
- Protein Q9V5Z7 is in the validation set at index 365895.
- Protein Q9TUX7 is in the validation set at index 27485.
- Protein Q9SXS8 is in the validation set at index 32168.
- Protein Q9SXS8 is in the validation set at index 32168.

The smaller the index number (rank) the more distant the protein embedding is to all other embeddings in the dataset.

### 4.4.2 Testing and Parameter Tuning

The aim of the hyperparameter tuning was to find the best parameters that would allow the algorithms to return the best optimal solutions as efficiently as possible.

Once the algorithms were implemented and tested on a small subset of data, they were run on a range of parameters. These runs were automated so that the data could be collected on a continuous basis. This involved running the algorithms various times and iteratively changing the hyperparameters. For each run the following data was stored:

- Algorithm
- Total run time
- Hyperparameter
- Top 20 most diverse proteins of the outputted algorithm
- Each protein's rank in the brute force baseline
- The percentile of the protein in the brute force baseline

To assess overall performance, the average percentile and rank, as well as the number of outputted proteins in the top 95%, 90% and 80% percentile were recorded for each different set of parameters, for each of the tested algorithms.

Once the best hyper-parameters were selected, given the time constraints, a set of of three tests were done on each algorithm to compare their outputs against the outputs of the other algorithms. Additionally, this allowed each algorithm to be assessed for overall performance rather than a random success.

# Chapter 5

# Results

## 5.1 The Most Distant Proteins

Solving the MDP on the full dataset by brute force calculation took 48 hours once parallelized. The most distant protein embedding was P83570 which is a regulatory neuropeptide. Neuropeptides are protein molecules used in neurogenic communication and can have a regulatory affect in both neurotransmission and other tissues of the body [54]. This neuropeptide affects the muscle tissues around the distal oviduct of the common cuttlefish. The neuropeptide stops the movement of the oviduct to reduce the baseline tension and therefore likely plays a role in reproduction [55].

Overall, none of the top 25 proteins were enzymes. Additionally, there were three uncharacterized proteins (P85398, Q54VH6, Q54B23) and one T cell receptor (P0DPR3) from a Homo sapien. All other proteins in the top 25 were protamines of varying natures however the majority of which were sperm pro-tamine. Protamine are small, basic proteins that are found in the sperm cells of many animals and fish [53].

Most of the most distant proteins came from different organisms, however, three of the most distant proteins P02333, P02332 and P02334, came from Oncorhynchus mykiss (Rainbow trout) (Salmo gaird-neri). Moreover, all of the top 25 most distant proteins were short in length ranging between 2 and 64 amino acids (AA). In total 19 out of 25 of the most distant proteins play a role in Sperm DNA condensation, chromosome condensation or cell differentiation and 20 out of 25 play a role in reproduction overall.

Table 5.1 shows the 25 most distant protein embeddings in descending order.

| 25 Most Distant Proteins in the SwissProt Dataset | |
|---|---|
| Protein Label | MaxSum Value |
| P83570 | 3220439.768 |
| P83215 | 3155623.427 |
| P85398 | 3152218.385 |
| P0DPR3 | 3060873.025 |
| Q54VH6 | 3048006.435 |
| P02333 | 3039970.423 |
| P02332 | 3019114.805 |
| P14402 | 3013188.554 |
| P12819 | 3007287.256 |
| P42131 | 3006592.975 |
| Q54B23 | 2996462.042 |
| Q9GLQ1 | 2983895.787 |
| P83213 | 2976980.858 |
| P67843 | 2964674.243 |
| P67842 | 2964674.243 |
| P67841 | 2964674.243 |
| P02334 | 2953423.711 |
| Q9GLP9 | 2949705.499 |
| P42139 | 2941778.076 |
| P42138 | 2939935.233 |
| P67840 | 2937300.46 |
| P67839 | 2937300.46 |
| P67838 | 2937300.46 |
| P42152 | 2935501.707 |

Table 5.1: Table showing the 25 most distant protein embeddings in the SwissProt dataset as calculated by the brute force MaxSum.

The PCA plots of the top 1000, 5000 and 10,000 most distant proteins can be seen in figures 5.1, 5.2 and 5.3 respectively:

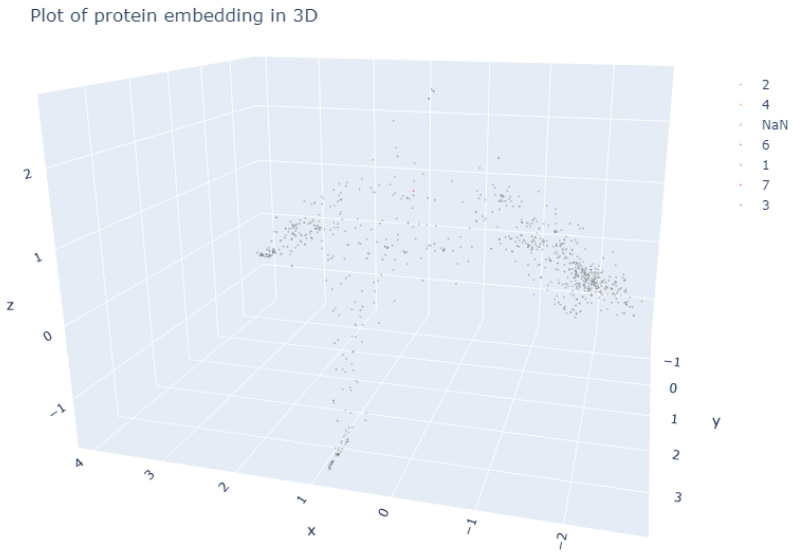


Figure 5.1: A PCA plot of the most distant 1000 protein embeddings in the entire SwissProt dataset, coloured by EC number

In the plot the most distant embeddings seem to be clustered in 3 corners and most sparse in the middle area of the embedding space. From this is seems that the more distant proteins may be similar to one another at least within these 3 principal components.

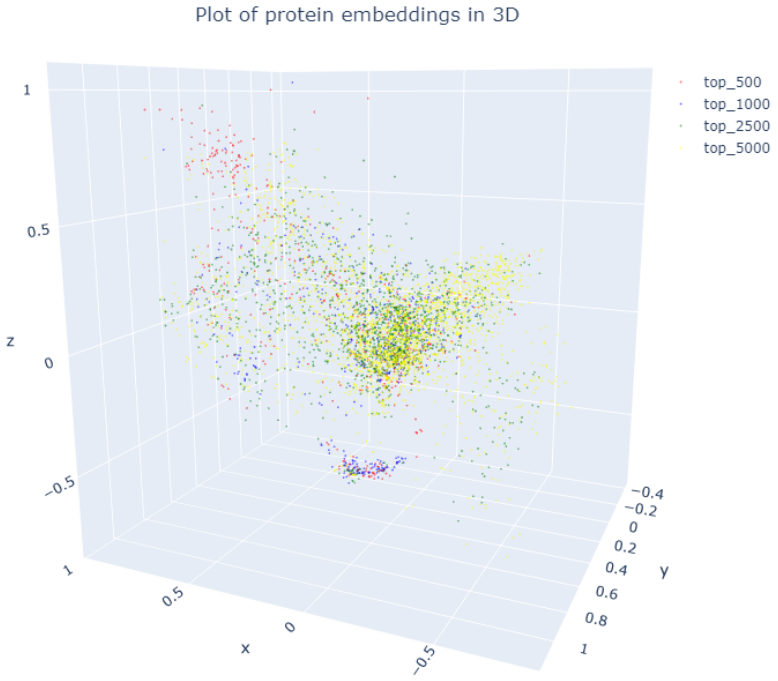


Figure 5.2: A PCA plot of the most distant 5000 protein embeddings in the entire SwissProt dataset, coloured by distance group

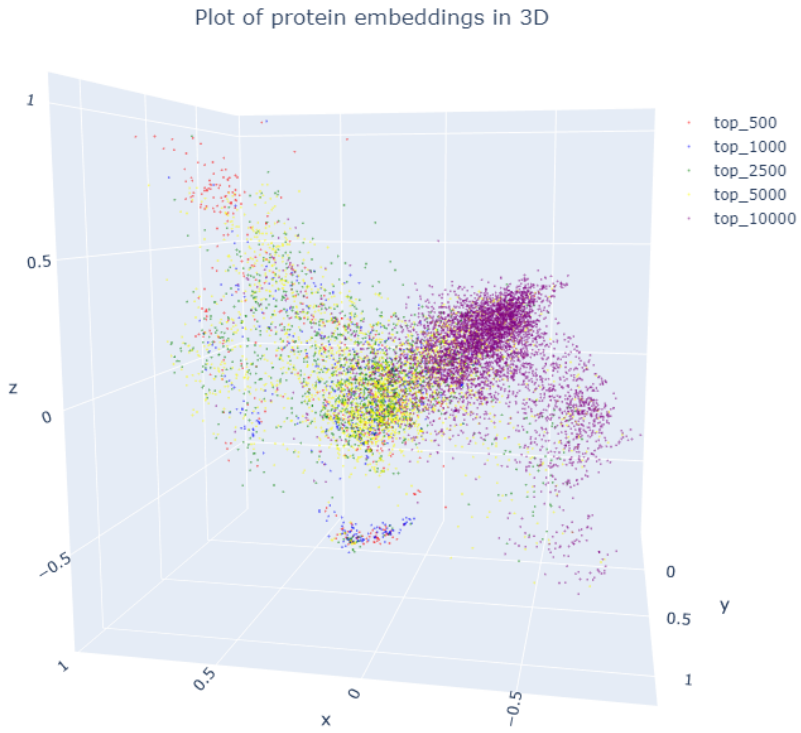


Figure 5.3: A PCA plot of the most distant 10,000 protein embeddings in the entire SwissProt dataset, Coloured by distance group

## 5.2 Algorithm Performance

The performance of the algorithms was assessed based on two main ideas, the first is based on how many of the most distant proteins outputted in a given subset of proteins are in the top 80th, 90th, 95th percentiles and the highest ranked brute force proteins. The second was based on how diverse the subset is based on the range of ranks, highest and lowest ranked protein, and the lowest and highest percentiles within one subset.

### 5.2.1 Performance on Small Subsets

All algorithms performed well on smaller set of samples of 101 – 10,000. The two best performing algorithms for returning the highest number of distant proteins in a subset was ITSv1 and IATS. The algorithms were run on their most efficient parameters rather than the parameters that elicit the best subset and performance. Regardless all algorithms outputted all 10 proteins in the top 95th percentile across all sample sizes.

ITSv1 and IATS outputted all proteins in the top 99.5th percentile of most distant

proteins across all sample sizes. However, for IATS, as the sample size increased over 1000 the number of proteins outputted in the top 99.9th percentile decreased as sample size increased. For ITSv1 however the number of proteins outputted in the top 99th percentile began to drop at 1000 samples but then stayed consistent to 5000 samples before decreasing as samples increased.

ITSv2 had less than half of outputted proteins in the top 99.9th percentile at 1000 samples which decreased to 0 by 5000 samples. As samples increased the number of proteins in the top 99.5th percentile and top 99th percentile decreased after 1000 and 5000 respectively. The TSME algorithm saw similar trends for the top 99th and top 99.5th percentile with decreases in outputted number of proteins at 1000 and 5000 samples respectively however there was no difference in the number of outputted proteins in the top 99.9th percentile after 1000 samples.

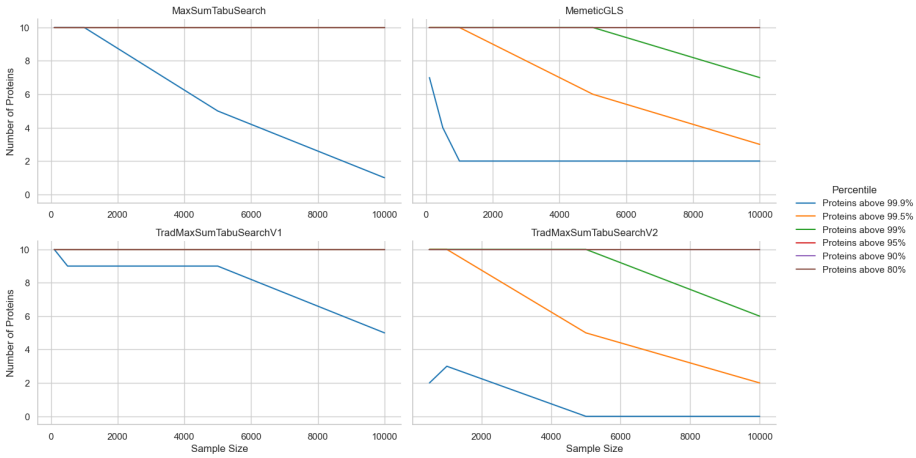


Figure 5.4: Plots showing the number of proteins in the sample in the top percentiles across sample sizes for all algorithms. IATS class name is MaxSumTabuSearch, ITSv1 and ITSv2 class name is TradMaxSumTabuSearchVX, and TSME is MemeticGLS.

## 5.3 Parameter Tuning and Testing

The parameter testing on the whole dataset showed that across all tested parameters the ITSv1 followed by IATS were the best overall algorithms at outputting the highest number of most distant proteins in the top percentiles of distant protein embeddings. The TSME algorithm showed mixed results but outperformed ITSv2 with many proteins in the top 80th percentile but less in the higher percentiles compared to IATS and ITSv1. ITSv2 also showed mixed results and had the worst performance in regard to both time and number of most distant proteins,

averaging only a quarter of the outputted proteins in the top percentiles over all parameters.

In contrast, across all tested parameters the best performing algorithms based on returning diverse subsets, as assessed by the range of ranks in the outputted subset, was TSME and ITSv2. TSME had ranges in ranks from 449,737 ranks and ITSv2 had ranges over 485,209 ranks in their outputted subsets over all parameter tests. IATS and ITSv1 did not have as diverse subsets with the ranges of ranks less than 80,043 and 114,006 ranks respectively. The relationship between range of ranks in the outputted subset over the number of iterations can be seen in Figure 5.5.

The tested parameters can be seen in A.6


Figure 5.5: Plots showing the relationship between range of ranks within an outputted subset and time. IATS class name is MaxSumTabuSearch, ITSv1 and ITSv2 class name is TradMaxSumTabuSearchVX, and TSME is MemeticGLS.

### 5.3.1 Key Insights From the Parameter Testing

Generally, for all algorithms as iterations increased, so did the computational complexity, resources and the time taken to search the solution space and return an output of proteins. However, there wasn't always an improvement in output accuracy with the increase of iterations. The number of proteins in each percentile over iterations can be seen in figure 5.6.

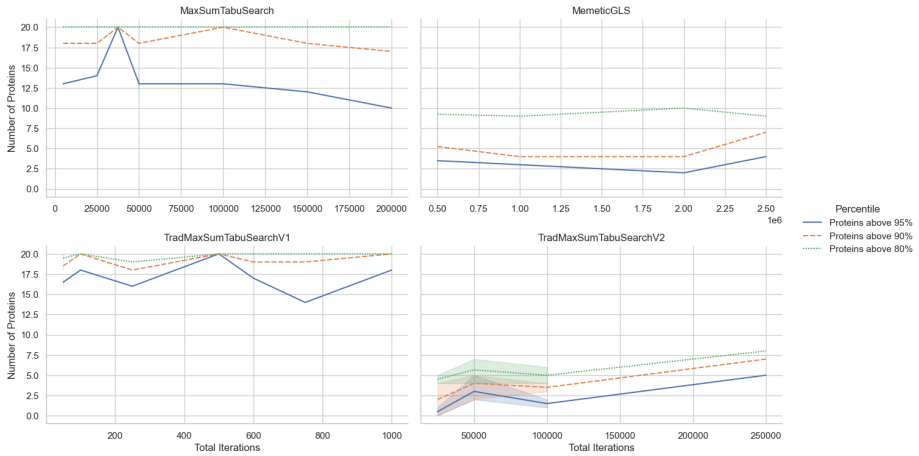


Figure 5.6: Plots showing the relationship between number of ranks within an outputted subset in the top percentiles and number of iterations. IATS class name is MaxSumTabuSearch, ITSv1 and ITSv2 class name is TradMaxSumTabuSearchVX, and TSME is MemeticGLS.

All algorithms except TSMv2 saw the number of proteins in the top 80th percentile stay relatively stable despite the number of iterations. For the IATS (MaxsumTabuSearch) algorithm the best subset was found between 25000 and 50000 (approximately 31250) iterations and the number of proteins in the top 95th and top 90th percentile would decrease after 100000 iterations. For ITSv1 (TradMaxSumSearchV1) the number of proteins in the top percentiles seemed to follow a pattern of increasing in number of proteins before decreasing and then repeating the pattern. The best performance was found at 500 iterations where all 20 proteins were in the top 80th percentile. After 500 iterations the performance of the ITSv1 algorithm decreased until 750 iterations and then, following the pattern seemed to be increasing in performance by 1000 iterations. It is unclear whether the performance would have continued to improve beyond this point. Moreover, its important to note that for ITSv1 in some tests a better single selected protein within the subset could be found in less iterations and time, for example in 1 hour and 250 iterations.

For the TSME (MemeticGLS) algorithm as iterations increased the number of proteins returned in the top 90th and top 95th percentile decreased until the 2,000,000 threshold before steadily increasing. However, at this same threshold the number of proteins in the 80th percentile began to decrease. For the ITSv2 algorithm, the number of proteins returned in all three percentiles increased after 100,000 iterations, with the best performance being found at 250,000 iterations.

Similarly, the performance as measured by number of proteins in the top percentiles also did not always improve with time. In figure 5.7 the number of proteins in the top 95th, 90th and 80th percentile over time can be seen.

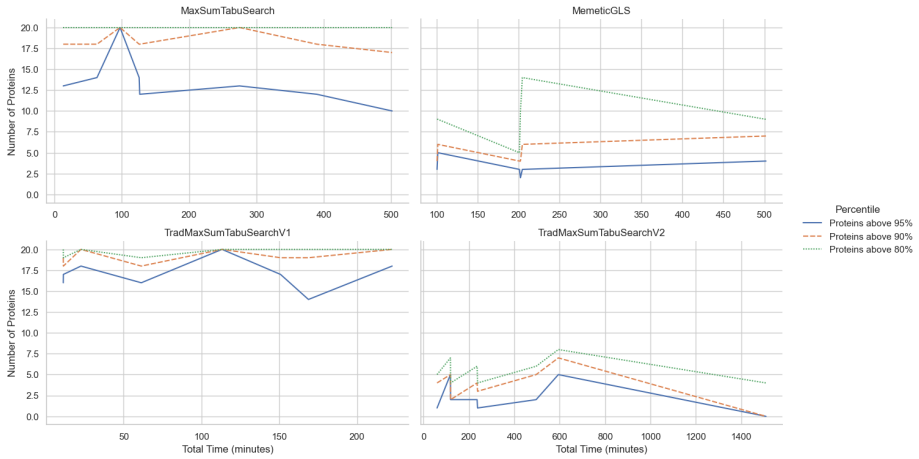


Figure 5.7: Plots showing the relationship between number of ranks within an outputted subset in the top percentiles and time. IATS class name is MaxSumTabuSearch, ITSv1 and ITSv2 class name is TradMaxSumTabuSearchVX, and TSME is MemeticGLS.

For IATS (MaxSumTabuSearch) a similar pattern to the number of iterations against performance can be seen. The best performing subset was found after running for approximately 100 minutes. After 300 minutes the performance of the algorithm decreases steadily. For ITSv1 the same pattern can be seen as in figure 5.6. For TBME (MemeticGLS) the best performance is seen just after 200 minutes and then decreases for the top 80th percentile whilst improving for the top 90th and top 95th percentile gradually over time.

For ITSv2 the best performance was around 600 minutes and then decreased after that, which contrasts what was seen in terms of performance over iterations. Therefore, there was also no improvement in the output proteins in the worst performing algorithm ITSv2 with longer run times and iterations (25 hours).

The results of the scalability tests of the four algorithms can be seen in figure 5.8:

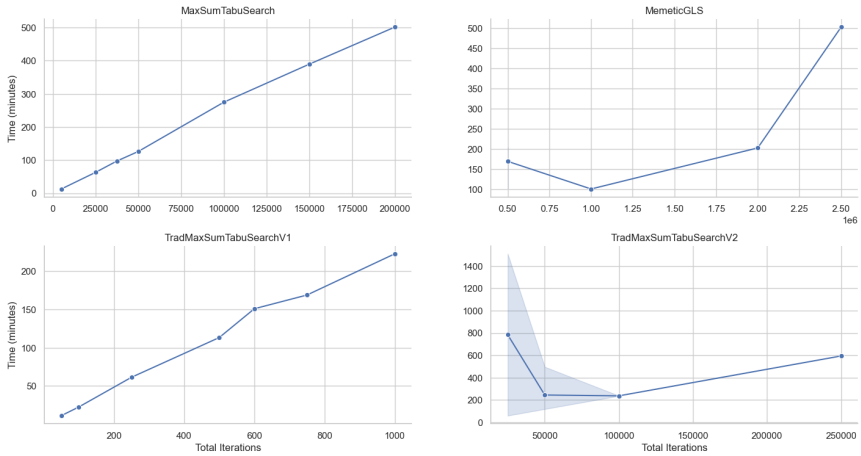


Figure 5.8: Plots showing the relationship between iterations and time for all algorithms on the full dataset. IATS class name is MaxSumTabuSearch, ITSv1 and ITSv2 class name is TradMaxSumTabuSearchVX, and TSME is MemeticGLS.

For the IATS (MaxsumTabuSearch) and ITSv1 (TradMaxSumSearchV1) the time taken had a positive correlation with iteration increase. However, for TSME (MemeticGLS) the amount of iterations only increase the time taken after 1,000,000 and 100,000 iterations respectively.

### 5.3.2 Alternative Iterative Tabu Search - IATS

Regardless of the parameters chosen all outputted proteins of the IATS algorithm were in the top 80th percentile, 17 or more were in or above the top 90th percentile and at least half of the outputted proteins were in the top 95th percentile overall. The best ranked protein outputted across all the tested parameters was 7th and the worst ranked 80,050th in the brute force baseline, both of which were still within the 80th percentile of the most distant proteins.

The best performing parameters for the IATS algorithm based on the test was 'num_proteins': 20, 'max_iterations': 750, 'local_iterations': 50, 'tabu_tenure': 10 had all 20 protein embeddings in the top 95th percentile of the baseline brute force calculation, took 97 minutes to run and had a rank range of 24,338 proteins. Although these parameters resulted in the best overall subset performance in terms of number of most distant proteins in each percentile, they did not result in the best single highest ranked protein across all tests.

The parameters that resulted in finding the single highest ranked protein were 'num_proteins': 20, 'max_iterations': 1000, 'local_iterations': 200, 'tabu_tenure': 10 which was also the parameters that yielded the most diverse subset with the

highest median rank 28,690, highest range of 80,043 ranks, lowest ranked protein (80,050) and had the longest run time 8.35 hours. Moreover, there was a lot less variance in the ranks returned with the highest variance being 23,865 ranks and the lowest being just 6753.93 ranks.

### 5.3.3 Traditional Iterative Tabu Search - ITSv1

ITSv1 outperformed all other algorithms at returning the most distant proteins in outputted subsets regardless of the parameters being tested. In all tests but two, all outputted proteins were in the top 80th percentile, 18 or more proteins were in the top 90th percentile and over 14 of 20 proteins were in the top 95th percentile across all parameters. ITSv1 was also able to pick out some of the proteins in the top 10 most distant proteins of the baseline brute force in 3 of the 8 tests despite differing parameters. The single most distant protein found by ITSv1 was ranked 2nd most distant in the brute force baseline with the following parameters 'num_proteins': 20, 'max_iterations': 250 and only ran for 1 hour and 1 minute. Across all parameters the least distant protein selected was still within the top 74th percentile of the most distant proteins of the brute force baseline.

ITSv1 also had the least diverse subset with the smallest range of ranks within the outputted subsets being less than that of ITSv2 and TSME but not IATS. The smallest range was 21,352 ranks, and the largest range was 142409 ranks and the standard deviation of the ranks in the outputted subsets ranged between 6797.68 to 31,191.44 ranks.

The best performing parameters relating to the most distant proteins in a subset for ITSv1 was 'num_proteins': 20, 'max_iterations': 500 and the best parameters for the most diverse subset was 'num_proteins': 20, 'max_iterations': 50. Furthermore, ITSv1 was the fastest algorithm overall across all tested parameters with the longest test running for just 3.7 hours and the shortest ran for just 11 minutes.

### 5.3.4 Traditional Iterative Tabu Search - ITSv2

The performance of ITSV2 was mixed for outputting a subset with the most distant proteins and was the worst of all tested algorithms across all parameters. Regardless of the parameters tested less than half of the outputted proteins were in the top 80th percentile, less than a quarter in the top 90th percentile and very few (between 0 and 5) proteins in the top 95th percentile. The highest ranked single protein across all parameter tests was ranked at 837th place and the worst was 559183th place in the brute force baseline of the entire dataset. The best

performing parameters for returning a subset with the highest number of distant proteins for ITSv2 was 'num_proteins': 20, 'max_iterations': 500, 'local_iterations': 500, 'local_sample_sizes': 100 and took 9.89 hours to run, had 5 proteins in the top 95th percentile, 7 proteins in the top 90th percentile, 8 proteins in the top 80th percentile and a range in ranks of 559,235 ranks.

Overall, ITSv2 was one of the best models for returning a diverse subset with high ranges in ranks, large standard deviations from 136,553 to 214,396 ranks within an outputted subset and contained both higher percentile proteins and bottom percentile proteins across all tested parameters. The highest percentiles in a subset would range from the top 99th percentile to the top 89th percentile and the lowest percentiles would be between the top 84th and top 99.7th percentile. Compared to the other algorithms ITSv2 also had some of the longest run times ranging between 59 minutes and 25 hours on the parameters tested. The best parameters for returning the most diverse subset were 'num_proteins': 20, 'max_iterations': 500, 'local_iterations': 500, 'local_sample_sizes': 100.

### 5.3.5 Memetic Tabu Search - TSME

The performance of the TSME was also mixed regarding providing subset with the highest number of distant proteins. Approximately half the outputted proteins were present in the top 80th across all tested parameters. Approximately a quarter of the proteins were in the top 90th percentile and less than a quarter ( less than 5) were in the top 95th percentile. Although approximately half of the outputted proteins were in the top 80th percentile there were also proteins, in the outputted lists, between the 12th and 6th showing extremely diverse subsets. The TSME algorithm had the greatest range in ranks overall and therefore had some of the most diverse subsets across all parameters, with the smallest range of ranks being 449,737 and the highest being 567,212.

In terms of diverse subsets, it outperformed ITSv2 outputting proteins in both the highest percentiles (top 99th percentile) and the lowest percentiles (10th percentile) across the majority of parameters. Furthermore, the single most distant protein outputted across all runs was the 105th most distant and the least distant being the 567,580th in the baseline brute force of the entire dataset. TSME also had high standard deviation of ranks within a subset ranging between 147,586 and 211,179 ranks across all tested parameters. TSME had longer run times than the ITSv1 and IATS algorithms however it didn't have the longest times overall. Depending on parameters the algorithm would run for anywhere between 1.5 hours

and 8.3 hours.

The best performing parameters for returning a diverse subset was 'dna_size': 20, 'max_epochs': 5000, 'local_iterations': 500, 'population_size': 500, 'retain_percent': 0.05.

## 5.4 Precision Testing for Returning The Highest Number of Distant Proteins in a Subset

The two best performing algorithms for finding subsets with the most distant proteins, IATS and ITSv1, were run on their best parameters on the entire dataset three times to assess whether the high performance on the parameter testing is repeatable and to see how precise the algorithms are.

**IATS was run on the following parameters:**

- Max_iterations = 750
- Local_iterations = 50
- Tabu_tenure = 10

**ITSv1 was run on the following parameters:**

- Max_iterations = 500
- Tabu_tenure = 10

Both algorithms' outputted subset had all 20 proteins in the top 80th percentile. Of the two algorithms ITSv1 was the most accurate and best performing over the three instances. ITSv1 had the smallest range in ranks of outputted proteins compared to IATS in all instances. ITSv1 also had more proteins in the highest percentiles (higher than the top 95th) over all test instances. However, ITSv1 is slightly slower than IATS taking 113 minutes and 95 minutes to output a subset of proteins respectively.

### 5.4.1 Alternative Iterative Tabu Search - IATS

In the accuracy tests all outputted proteins were in the top 80th percentile of most distant protein in the brute force baseline. 20 proteins were in the top 90th percentile of most distant proteins in 1 of 3 of the total test instances and 17 of the proteins were in the top 90th percentile in 2 of 3 instances. Half of the proteins

outputted by IATS were in the top 95th percentile (12,10, 12 proteins over the 3 instances), a quarter of the outputted proteins were in the top 99th percentile, at least 3 but no more than 5 were in the top 99.5th percentile and no proteins were in the top 99.9th percentile of most distant proteins across all test instances. The median and average percentile that the outputted proteins were in across the three instances was the top 96th percentile. The highest ranked protein over the three test instances was ranked 237th and the lowest was ranked 70,944th in the brute force baseline.

### 5.4.2 Traditional Iterative Tabu Search - ITSv1

In the precision tests, all outputted proteins were in the top 80th percentile. All outputted proteins were in the top 90th percentile in 2 of 3 test instances. In the test instance where not all were in the top 90th percentile 19 of the 20 outputted proteins were. In all three test instances 18 of the 20 proteins were in or above the top 95th percentile and 10 of those were in the top 99th percentile. In 2 of 3 tests 9 of the 20 proteins were in the top 99.5th percentile and in every test one of the proteins were in the top 99.9th percentile of most distant proteins of the baseline brute force of the entire dataset. Moreover, the ITSv1 algorithm was able to find and output the most distant protein in the entire dataset (ranked 0) in one of the three tests and in the other two test outputted the second most diverse protein overall. The lowest ranked protein over the three runs was 62,048th . The ITSv1 algorithm also had much smaller range in ranks than IATS with the largest range being 62,048 ranks and the smallest range being 34,609 ranks. In all tests the least diverse protein was less than or equal to the top 89th percentile of most diverse proteins.

The summary table for the precision tests can be seen in Appendix A.2

## 5.5 Repeatability Tests for Returning Diverse Subsets

### 5.5.1 Best Algorithms

The two best performing algorithms for diverse subsets, TSME and ITSv2, were run on their best parameters on the entire dataset three times to assess whether the high performance on the parameter testing was repeatable and to see how precise the algorithms were.

**TSME was run on the following parameters:**

- Max_iterations = 500
- Local_iterations = 500
- Lolcal_sample_size = 100
- Tabu_tenure = 10

**ITSv2 was run on the following parameters:**

- Max_iterations = 5000
- Local_iterations = 500
- Population_size = 500
- Retain percentage = 0.05

The summary of the repeatability testing can be seen in A.3

### 5.5.2 Memetic Tabu Search - TSME

Over all three tests the TSME algorithm returned a different subset of proteins each time. However, each time the proteins had a large range of ranks, between 498,163 and 55,1147 ranks, and large standard deviation between 160,065 and 181,809.82 ranks. In 2 of the 3 tests the highest ranked protein was within the top 200 most distant proteins and in all 3 tests within the top 1000 most distant protein embeddings.

Looking at the output subset with the highest standard deviation we can see that the proteins come from a large range of organisms, serve different purposes, and vary in amino acid sequence length. Not one of the 20 outputted proteins in the subset were from the same organism, had the same amino acid length or served the same purpose. Some of the proteins came from viruses, others from bacteria or large multicellular organism such as pigs. 7 of the 20 proteins were enzymes.

The proteins and their functions can be seen in A.4.

### 5.5.3 Traditional Iterative Tabu Search - ITSv2

Over all three tests ITSv2 outputted the exact same subset of proteins each time. The standard deviation of this subset was 162,289 ranks and therefore had a higher

standard deviation than 1 of 3 if the subsets outputted by TSME. In this subset the highest ranked protein was 9696. The proteins rank ranged over 551,596 ranks.

The proteins of the output subset come from a range of organisms however in this subset there are proteins from the same organism. For example, 4 of 20 of the proteins are found in humans, 2 of 20 are from fruit flies and 2 of 20 are from Oceanobacillus iheyensis. None of the proteins have the same function. However, 5 of the 20 are enzymes.

The proteins and their functions can be seen in A.5.

# Chapter 6

# Discussion

The aim of this projects was to find subsets of the most diverse proteins and, if possible, find the subset of most diverse proteins in an embedding space that is both large in terms of the size of the dataset, but also complex in terms of its high dimensionality. Formulated as an MDP, this is an NP-Hard problem made even more difficult by the intrinsic nature of the dataset and the significant computational and memory resources required for its processing.

From the outset, it was understood that metaheuristic and memetic strategies would need to be applied to solve the problem but the practicalities of dealing with the data were not that well understood. For instance, it was surprising to find out that it would take 1300 hours to calculate the brute force max sum baseline on high specification computing resources. That is just over 54 days, which would not be tractable for this project given the timescale and the amount of preparatory work required to submit the task to the processing queue (the code would have to be correct first time because re-submission would not have been possible). Not only did it require high specification computational resources to be arranged, but it did ultimately require the code to be parallelized to reduce computation time. Otherwise, it would have impacted the success of the project. Other strategies also had to be adopted whereby only a small subset of the data was used for development and testing.

## 6.1 Algorithm Performance

Four algorithms were implemented and evaluated for their precision, efficiency and scalability. The four algorithms were two versions of the traditional iterative tabu search (ITSv1, Itsv2), iterative alternative tabu search (IATS), and the memetic algorithm (TSME). ITSv1 and IATS were very good at returning precise subsets

of the most distant proteins, which may be the most diverse subset of the entire dataset. However, a subset of the most distant doesn't necessarily guarantee a diverse subset overall. For example, the most distant proteins may be very different to the majority of protein embeddings, but they may still be close in space to each other and therefore similar to one another, meaning a subset of the most distant proteins may not necessarily be a very diverse subset. Overall, all algorithms were able to return some of the most distant proteins in the top percentiles in their outputted subset.

The algorithms that outputted the most 'diverse' subsets were TSME and ITSv2. Both TSME and ITSv2 had the greatest range of ranks and largest standard deviations within an outputted subset regardless of their parameters, meaning that there was far more variability in ranks within an output subset. The subsets output by TSME and ITv2 had proteins that were close in vector space to most proteins in the entire dataset and some that were very far away from others. Therefore, one might speculate that the outputted proteins were very different to one another. However, it is not guaranteed that the subsets outputted were the most diverse proteins or the most diverse subset in the entire dataset as a whole.

It is therefore unclear which algorithms performed the best overall without defining what exactly diversity means because depending on desired output, the algorithms may or may not be suitable. If the most diverse subset is determined by the most distant proteins, then ITSv1 and IATS may be the most suitable choice, however, if the goal is to return a subset where all the given proteins are very different to one another regardless of the entire dataset as a whole then TSME or ITSv2 would be more suitable because they return subsets where the range of ranks were large, i.e. they were able to output some of the most and least distant proteins in the dataset.

Regardless of the ambiguity in the definition of what constitutes a diverse subset, algorithms were successful in returning near optimal subsets efficiently.

### 6.1.1 Theories Regarding Performance

It is likely that one of the contributing factors for the success of the algorithms for both output goals was the multi start approach in the algorithms outer loops because it allows the algorithms to have multiple start locations, increasing the chances that the solution will escape local optima but also providing a large range of different proteins in the solution space. Additionally, the tabu list present in

all algorithms also helps to avoid getting stuck in a local optima by preventing the revisiting of previously explored solutions. It is likely that IATS was effective at returning the most distant proteins because it is the closest to the original brute force because it samples from the global space, but does so by breaking it into smaller random subsets which act as pseudo neighborhoods making the overall global search more efficient. It's also the most simplistic of the four algorithms.

ITSv1 was likely able to output subsets with the most distant proteins because during neighborhood generation only a single random change is made so when better solutions are found based on the fitness metric, more of the better proteins (those with greater distances) are kept in the solution.

However, one pitfall that all the algorithms faced, was because of the following phenomenon. All solution subsets are replaced wholesale if the subset fitness is the current best, even though it may be discarding proteins in the global list that are better solutions. For instance, if a new solution subset contains a single protein with a very high distance and a few with low distances, then it can potentially replace a better solution subset strictly based on its overall higher max sum value. For example, if we have a list of 5 proteins with the following distances [90, 10, 5, 10, 10] the max sum would be 125 compared to [50, 20, 30, 5, 10] max sum of 115. In this case, the first list would be selected as the better solution despite having most of its protein with lower distances than the second list. Ideally, what should have happened was that a new solution based on both lists should have been returned, [90, 50, 30, 20, 10]. This may explain why some algorithms seemed to output less proteins in the higher percentiles, and why the performance of the algorithms dropped off over longer run times and iterations. It also makes it hard to know whether the range is a good metric for assessing diversity.

It is important to note that if a new solution was formed from the two lists containing only the proteins with the highest MaxSum values that this in itself may not have been a diverse list and therefore would only be ideal if diversity is defined as a subset with the highest max sum. In the precision and repeatability tests the ITSv2 returned the same subset in all 3 tests. This may be an example of the algorithm finding a very diverse subset, but it may also be an example of the algorithm getting stuck in a local optimum.

## 6.2 Base assumptions

### 6.2.1 Distance and Diversity

In implementing the heuristic and metaheuristic methods to find efficient ways of finding diverse subsets of proteins in the embedding space, it was initially assumed that the most diverse subset meant finding the subset that contained the proteins with the largest distance from all other proteins. Given the discussion above, about how proteins with the large distances may still be close to each other in vector space and thus similar, which led a to a certain ambiguity in the definition of the true objective.

There is a wider question about how valid the assumption was and what diversity really means. Protein embeddings are lower dimensional representation of the original data, in this context, is a protein embedding with a large distance in vector space to all other protein embeddings, as calculated by the MaxSum, really diverse when it may be within a cluster of equally distant proteins?

### 6.2.2 Embeddings and Validity

In the same regard, another base assumption was that the protein embeddings are good representation of the proteins themselves and that dissimilarity of these proteins can be determined by their distance in vector space. However, without knowing what important structural and functional characteristics have been reduced to create the embeddings and whether or not these characteristics have been preserved during the dimensionality reduction process, it is unclear whether these proteins' representations are accurate and whether measuring dissimilarity via the distance metric is valid. Questions such as, what aspects are being emphasised by measuring the dissimilarity with a given distance metric? Could not be investigated further or validated in this project but are perhaps important when defining the objectives of any future problem to be solved with these approaches.

## 6.3 Limitations

### 6.3.1 Time Constraints and Testing

The results of this work should be taken with a pinch of salt. Due to the time constraints many of the tests were short and not robust. For example, only one test was conducted on each parameter which doesn't control for the possibility that the results may have been by chance, and it does not check for repeatability.

Therefore at least three runs should have been done with each parameter to validate that the effect of that parameter on performance.

Additionally, only a fraction of the possible parameter combinations was tested for each algorithm. In total all algorithms were only tested over $6 - 8$ combinations of parameters, with the algorithms with longer run times receiving less testing than those with shorter times. This was not an exhaustive coverage of the potential options for tuning to ensure the best performance of the algorithms, but also meant that the comparisons were not entirely fair. Ideally more than three runs on each parameter should have been conducted to ensure the work was repeatable and reliable. Therefore, the optimal parameters identified may not actually be most optimal for this task but also may not be suitable for another embedding dataset.

### 6.3.2 Computational Considerations

The tests and runs were limited to running on a relatively high spec Acer predator 500 laptop with a 6-core intel(R) Core (TM) i7-8750H CPU @ 2.20GHz, GeForce RTZ 2070 GPU and 32GB of DDR4 ram. During the runs, no more than half the memory was required and the computing resource demands were well within the system's capacity. Therefore, these results are only indicative of the algorithms' potential given an equivalent system, bearing in mind that a larger dataset or volume of data would again affect the results.

### 6.3.3 Chosen Metrics

It was also difficult to assess what is considered a diverse subset. In this work, that decision was based on two main criteria. The first was the MaxSum distance in vector space, which assumed that a subset with the highest MaxSum Euclidean distance would be the most diverse subset of the entire dataset, and the second is a diverse subset had a large standard deviation of and range in ranks between protein embeddings in the 'best' subset. The first may be true when extracting a single subset from an entire dataset. However, as previously discussed, this doesn't guarantee a diverse subset itself. As for the second assessment, the output subset may have a large variety of proteins at different rankings and thus be diverse but may not be the most diverse subset from the entire dataset. Similarly, proteins were ranked based on their MaxSum distance as calculated by the brute force calculation of the entire dataset, but this may not be a valid method to assess whether a subset is diverse.

Moreover, in this work the MaxSum distance was based on Euclidean distance however other distance metrics may be better suited for the problem and could yield different results. As previously discussed, it is also difficult to say which metric would be more suitable because we do not know what a diverse subset is, let alone what the most diverse subset is beforehand, therefore there is no accurate baseline to assess which metric is the most suitable in the first place.

From the early data exploration with the Cosine similarity, Manhattan and Euclidean distance, both Manhattan and Euclidean metrics outputted the same proteins. The Euclidean distance was chosen for its simplicity; however, this has no effect on its suitability for the task at hand. If a different distance metric was used different proteins may have been considered the most distant. Furthermore, other methods for solving the MDP, such as MinDiff or MaxMin, may have also been better suited, and likely would have returned different results.

### 6.3.4 Other Considerations

The only way to know for sure if the proteins are diverse is by looking them up in the SwisProt database, but this too has its limitations. This is a time-consuming task because of the sheer volume of data, the data may be incomplete and even if the proteins seem diverse at face value, they still may bare similarities that may be overlooked, so specialist domain knowledge is also required.

It is also important to remember that because protein embeddings were used, the outputted subset proteins may not actually be diverse to one another. Protein embeddings are a semantic representation of the proteins themselves and therefore we can only confidently say that the subset of protein embeddings selected are diverse, not necessarily that the proteins themselves are.

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

Proteins play a crucial role in the biological world and understanding them can benefit both society and our understanding of the world. Protein embeddings are a recent development in bioinformatics and are a tool used in our attempt to understand both the similarities and dissimilarities in proteins.

This work was aimed at finding methods to identify the most distant proteins and most diverse subsets of proteins from large protein databases in a scalable and efficient way using a dataset of protein embeddings, data mining techniques and metaheuristics.

The problem of finding the most distant and diverse subset of proteins in a large dataset was formulated as a MaxSum maximum diversity problem and four metaheuristic algorithms were implemented and evaluated against a brute force MaxSum baseline.

The aims of this project were to assess the ability of the tested metaheuristics at finding diverse subsets of protein embeddings within the database and to find the most diverse subsets of the dataset as a whole. Although it is hard to formulate what a diverse subset is let alone what the most diverse subset of the dataset is, to achieve these aims it was assumed that the most diverse subset in the entire dataset would have proteins with the greatest Euclidean distances from the other embeddings and that a diverse set would have a high standard deviation and range of ranked protein embeddings.

Two of the implemented methods, alternative tabu search (IATS) and a version

of the traditional tabu search ITSv1 were shown to be robust at returning subsets with the most distant proteins. While ITSv2 and a memetic tabu search TSME appeared to be better at finding diverse subsets. In the limited time available for this project, it was demonstrated that these metaheuristic approaches are able to provide optimal results, but further work is required to improve on the base implementations to take this research further.

This project has provided a variety of algorithms to find diverse subsets within the embedding space, tackle large embedding datasets and has provided a base for future work.

## 7.2 Algorithm improvements and Future Work

Given more time, the tabu search algorithms would have been improved by ensuring that the best solution was refined as opposed to replaced wholesale. The refinement process would create a new list by selecting the proteins from the local best subset, that would improve the global best subset and replacing the individual proteins in the global list with lower distances with those with higher distances from the local list.

There is also plenty of opportunity for future research into the algorithms search procedures. This could look like using global and local histories to analyze how much of the search space is explored during each iteration or over a single run of the algorithm. This research could aid the understanding of how these algorithms search the embedding space but could also lead to improvement of the search algorithms themselves.

Further research into the embeddings in the output subsets could be done by mapping the outputted protein labels to the SwissProt dataset which could also be informative. This could provide further information on dissimilarities between items in the subset but also where the dissimilarities are in the proteins themselves. Visualising the outputted protein subsets in terms of the greater PCA or UMAP plots might also aid in understanding where in the vector space they are located.

# Bibliography


[1] "Protein - amino acids, structure, function — britannica." (Aug. 1, 2023), [Online]. Available: https://www.britannica.com/science/protein (visited on 08/18/2023).

[2] E. Kruse, N. Uehlein, and R. Kaldenhoff, "The aquaporins," *Genome Biology*, vol. 7, no. 2, p. 206, Feb. 28, 2006, ISSN: 1474-760X. DOI: 10.1186/gb-2006-7-2-206. [Online]. Available: https://doi.org/10.1186/gb-2006-7-2-206 (visited on 08/18/2023).

[3] P. K. Robinson, "Enzymes: Principles and biotechnological applications," *Essays in Biochemistry*, vol. 59, pp. 1–41, Nov. 15, 2015, ISSN: 0071-1365. DOI: 10.1042/bse0590001. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4692135/ (visited on 08/18/2023).

[4] "Enzyme — definition, mechanisms, & nomenclature — britannica." (Jul. 4, 2023), [Online]. Available: https://www.britannica.com/science/enzyme (visited on 08/18/2023).

[5] "Embeddings — machine learning," Google for Developers. (), [Online]. Available: https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture (visited on 08/17/2023).

[6] johnmaeda. "LLM AI embeddings." (May 23, 2023), [Online]. Available: https://learn.microsoft.com/en-us/semantic-kernel/memories/embeddings (visited on 08/17/2023).

[7] W. Koehrsen. "Neural network embeddings explained," Medium. (Oct. 2, 2018), [Online]. Available: https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526 (visited on 08/17/2023).

[8] K. K. Yang, Z. Wu, C. N. Bedbrook, and F. H. Arnold, "Learned protein embeddings for machine learning," *Bioinformatics*, vol. 34, no. 15, pp. 2642–2648, Aug. 1, 2018, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty178. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6061698/ (visited on 09/15/2023).

[9] "Protein embeddings — UniProt help — UniProt." (), [Online]. Available: https://www.uniprot.org/help/embeddings (visited on 08/17/2023).

[10] R. Martí, A. Martínez-Gavara, S. Pérez-Peló, and J. Sánchez-Oro, "A review on discrete diversity and dispersion maximization from an OR perspective," *European Journal of Operational Research*, vol. 299, no. 3, pp. 795–813, Jun. 16, 2022, ISSN: 0377-2217. DOI: 10.1016/j.ejor.2021.07.044. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221721006548 (visited on 08/02/2023).

[11] T. Shinozaki, T. Iwaki, S. Du, M. Sekijima, and S. Furui, "Distance-based factor graph linearization and sampled max-sum algorithm for efficient 3d potential decoding of macromolecules," *IPSJ Transactions on Bioinformatics*, vol. 4, pp. 34–44, 2011. DOI: 10.2197/ipsjtbio.4.34.

[12] A. Gerniers, O. Bricard, and P. Dupont, "MicroCellClust: Mining rare and highly specific subpopulations from single-cell expression data," *Bioinformatics*, vol. 37, no. 19, pp. 3220–3227, Oct. 11, 2021, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab239. [Online]. Available: https://doi.org/10.1093/bioinformatics/btab239 (visited on 07/19/2023).

[13] C. Atallah, K. James, Z. Ou, *et al.*, "Automatic diverse subset selection from enzyme families by solving the maximum diversity problem," in *2022 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, Aug. 2022, pp. 1–9. DOI: 10.1109/CIBCB55180.2022.9863021.

[14] D. R. Shier, "A min-max theorem for p-center problems on a tree," *Transportation Science*, Aug. 1, 1977, Publisher: INFORMS. DOI: 10.1287/trsc.11.3.243. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/trsc.11.3.243 (visited on 08/02/2023).

[15] M. J. Kuby, "Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems," *Geographical Analysis*, vol. 19, no. 4, pp. 315–329, 1987, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1538-4632.1987.tb00133.x, ISSN: 1538-4632. DOI: 10.1111/j.1538-4632.1987.tb00133.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1538-4632.1987.tb00133.x (visited on 08/02/2023).

[16] E. Erkut, "The discrete p-dispersion problem," *European Journal of Operational Research*, vol. 46, no. 1, pp. 48–60, May 4, 1990, ISSN: 0377-2217. DOI: 10.1016/0377-2217(90)90297-O. [Online]. Available: https://www.sciencedirect.com/science/article/pii/037722179090297O (visited on 08/02/2023).

[17] L. F. Santos, M. H. Ribeiro, A. Plastino, and S. L. Martins, "A hybrid GRASP with data mining for the maximum diversity problem," in *Hybrid Metaheuristics*, M. J. Blesa, C. Blum, A. Roli, and M. Sampels, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2005, pp. 116–127, ISBN: 978-3-540-31898-9. DOI: 10.1007/11546245_11.

[18] "Local search algorithms and optimization problem - TAE." (), [Online]. Available: https://www.tutorialandexample.com/local-search-algorithms-and-optimization-problem (visited on 07/27/2023).

[19] S. Spiers, H. T. Bui, and R. Loxton, "An exact cutting plane method for the euclidean max-sum diversity problem," *European Journal of Operational Research*, vol. 311, no. 2, pp. 444–454, Dec. 2023, ISSN: 03772217. DOI: 10.1016/j.ejor.2023.05.014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S037722172300379X (visited on 07/18/2023).

[20] M. Lozano, D. Molina, and C. Garcı´a-Martı´nez, "Iterated greedy for the maximum diversity problem," *European Journal of Operational Research*, vol. 214, no. 1, pp. 31–38, Oct. 1, 2011, ISSN: 0377-2217. DOI: 10.1016/j.ejor.2011.04.018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221711003626 (visited on 08/02/2023).

[21] "What is metaheuristic?: AI terms explained - AI for anyone." (), [Online]. Available: https://www.aiforanyone.org/glossary/metaheuristic (visited on 09/12/2023).

[22] F. Glover and K. Sörensen, "Metaheuristics," *Scholarpedia*, vol. 10, no. 4, p. 6532, Apr. 27, 2015, ISSN: 1941-6016. DOI: 10.4249/scholarpedia.6532. [Online]. Available: http://www.scholarpedia.org/article/Metaheuristics (visited on 09/12/2023).

[23] G. Silva, M. Andrade, L. Ochi, S. Martins, and A. Plastino, "New heuristics for the maximum diversity problem," *J. Heuristics*, vol. 13, pp. 315–336, Aug. 1, 2007. DOI: 10.1007/s10732-007-9010-x.

[24] J. B. Ghosh, "Computational aspects of the maximum diversity problem," *Operations Research Letters*, vol. 19, no. 4, pp. 175–181, Oct. 1996, ISSN: 01676377. DOI: 10.1016/0167-6377(96)00025-9. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0167637796000259 (visited on 09/17/2023).

[25] F. Sandoya, R. Aceves, F. Sandoya, and R. Aceves, "Grasp and path relinking to solve the problem of selecting efficient work teams," in *Recent Advances on Meta-Heuristics and Their Application to Real Scenarios*, IntechOpen, Jan. 30, 2013, ISBN: 978-953-51-0913-6. DOI: 10.5772/53700. [Online]. Available: https://www.intechopen.com/chapters/42288 (visited on 07/26/2023).

[26] C. Venkateswarlu, "A metaheuristic tabu search optimization algorithm: Applications to chemical and environmental processes," in *Engineering Problems - Uncertainties, Constraints and Optimization Techniques*, IntechOpen, Jun. 4, 2021, ISBN: 978-1-83969-368-7. DOI: 10.5772/intechopen.98240. [Online]. Available: https://www.intechopen.com/chapters/77046 (visited on 08/03/2023).

[27] A. Duarte, R. Martí, and R. Marti, "Tabu search for the maximum diversity problem,"

[28] F. Liang. "Optimization techniques — tabu search," Medium. (Jul. 27, 2020), [Online]. Available: https://towardsdatascience.com/optimization-techniques-tabu-search-36f197ef8e25 (visited on 08/03/2023).

[29] R. Aringhieri and R. Cordone, "Tabu search versus GRASP for the maximum diversity problem," *4OR*, vol. 6, Mar. 1, 2008. DOI: 10.1007/s10288-007-0033-9.

[30] R. Martí, M. Gallego, A. Duarte, and E. G. Pardo, "Heuristics and metaheuristics for the maximum diversity problem," *Journal of Heuristics*, vol. 19, no. 4, pp. 591–615, Aug. 1, 2013, ISSN: 1572-9397. DOI: 10.1007/s10732-011-9172-4. [Online]. Available: https://doi.org/10.1007/s10732-011-9172-4 (visited on 08/11/2023).

[31] P. Hansen and N. Mladenović, "Variable neighborhood search," in *Handbook of Heuristics*, R. Martí, P. M. Pardalos, and M. G. C. Resende, Eds., Cham: Springer International Publishing, 2018, pp. 759–787, ISBN: 978-3-319-07124-4. DOI: 10.1007/978-3-319-07124-4_19. [Online]. Available: https://doi.org/10.1007/978-3-319-07124-4_19 (visited on 08/03/2023).

[32] N. Mladenovic, "A tutorial on variable neighborhood search," Jan. 10, 2004.

[33] P. Hansen, N. Mladenovic, R. Todosijević, and S. Hanafi, "Variable neighborhood search: Basics and variants," *EURO Journal on Computational Optimization*, vol. 5, Aug. 12, 2016. DOI: 10.1007/s13675-016-0075-x.

[34] P. Garg, "A comparison between memetic algorithm and genetic algorithm for the cryptanalysis of simplified data encryption standard algorithm," no. 1, 2009.

[35] "What is a genetic algorithm (and how does it work)? — cylab.be." (), [Online]. Available: https://cylab.be/blog/172/what-is-a-genetic-algorithm-and-how-does-it-work (visited on 08/08/2023).

[36] V. Mallawaarachchi. "Introduction to genetic algorithms — including example code," Medium. (Mar. 1, 2020), [Online]. Available: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (visited on 07/25/2023).

[37] "ML - convergence of genetic algorithms," GeeksforGeeks. Section: Machine Learning. (Jun. 9, 2020), [Online]. Available: https://www.geeksforgeeks.org/ml-convergence-of-genetic-algorithms/ (visited on 08/08/2023).

[38] A. E. Ezugwu, O. J. Adeleke, A. A. Akinyelu, and S. Viriri, "A conceptual comparison of several metaheuristic algorithms on continuous optimisation problems," *Neural Computing and Applications*, vol. 32, no. 10, pp. 6207–6251, May 1, 2020, Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 10 Publisher: Springer London, ISSN: 1433-3058. DOI: 10.1007/s00521-019-04132-w. [Online]. Available: https://link.springer.com/article/10.1007/s00521-019-04132-w (visited on 08/15/2023).

[39] F. Neri and C. Cotta, "Memetic algorithms and memetic computing optimization: A literature review," *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, Feb. 1, 2012, ISSN: 2210-6502. DOI: 10.1016/j.swevo.2011.11.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650211000691 (visited on 08/11/2023).

[40] J.-K. Hao, "Memetic algorithms in discrete optimization,"

[41] X. Lai, J. Hao, D. Yue, and H. Gao, "Diversification-driven memetic algorithm for the maximum diversity problem," in *2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, Nov. 2018, pp. 310–314. DOI: 10.1109/CCIS.2018.8691160.

[42] Y. Zhou, J.-K. Hao, and B. Duval, "Opposition-based memetic search for the maximum diversity problem," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 5, pp. 731–745, Oct. 2017, Conference Name: IEEE Transactions on Evolutionary Computation, ISSN: 1941-0026. DOI: 10.1109/TEVC.2017.2674800.

[43] H. Tizhoosh, "Opposition-based learning: A new scheme for machine intelligence," in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent*

*Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, vol. 1, Nov. 2005, pp. 695–701. DOI: 10.1109/CIMCA.2005.1631345.

[44] M. Dorigo, "Ant colony optimization," *Scholarpedia*, vol. 2, no. 3, p. 1461, Mar. 28, 2007, ISSN: 1941-6016. DOI: 10.4249/scholarpedia.1461. [Online]. Available: http://www.scholarpedia.org/article/Ant_colony_optimization (visited on 07/27/2023).

[45] "Introduction to ant colony optimization," GeeksforGeeks. Section: Write From Home. (May 15, 2020), [Online]. Available: https://www.geeksforgeeks.org/introduction-to-ant-colony-optimization/ (visited on 08/15/2023).

[46] "A gentle introduction to particle swarm optimization - MachineLearningMastery.com." (), [Online]. Available: https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/ (visited on 08/15/2023).

[47] S. Sanyal. "An introduction to particle swarm optimization (PSO algorithm)," Analytics Vidhya. (Oct. 30, 2021), [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-particle-swarm-optimization-algorithm/ (visited on 09/12/2023).

[48] "Particle swarm optimization (PSO) - an overview," GeeksforGeeks. Section: Machine Learning. (Apr. 22, 2021), [Online]. Available: https://www.geeksforgeeks.org/particle-swarm-optimization-pso-an-overview/ (visited on 09/12/2023).

[49] A. rezaee jordehi and J. Jasni, "Particle swarm optimisation for discrete optimisation problems: A review," *Artificial Intelligence Review*, vol. 43, Dec. 1, 2014. DOI: 10.1007/s10462-012-9373-8.

[50] J. Brownlee. "Simulated annealing from scratch in python," MachineLearningMastery.com. (Feb. 18, 2021), [Online]. Available: https://machinelearningmastery.com/simulated-annealing-from-scratch-in-python/ (visited on 09/13/2023).

[51] "Hill climbing algorithm in AI - TAE." (), [Online]. Available: https://www.tutorialandexample.com/hill-climbing-algorithm (visited on 07/27/2023).

[52] "Simulated annealing," GeeksforGeeks. Section: Mathematical. (Aug. 11, 2017), [Online]. Available: https://www.geeksforgeeks.org/simulated-annealing/ (visited on 09/13/2023).

[53] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical Science*, vol. 8, no. 1, pp. 10–15, Feb. 1993, Publisher: Institute of Mathematical Statistics, ISSN: 0883-4237, 2168-8745. DOI: 10.1214/ss/1177011077. [Online]. Available: https://projecteuclid.org/journals/statistical-science/volume-8/issue-1/Simulated-Annealing/10.1214/ss/1177011077.full (visited on 09/13/2023).

[54] "Understanding UMAP." (), [Online]. Available: https://pair-code.github.io/understanding-umap/ (visited on 08/17/2023).

[55] "UMAP: Uniform manifold approximation and projection for dimension reduction — umap 0.5 documentation." (), [Online]. Available: https://umap-learn.readthedocs.io/en/latest/ (visited on 09/22/2023).

[56] "How UMAP works — umap 0.5 documentation." (), [Online]. Available: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html (visited on 09/22/2023).

[57] C. Cheng. "Principal component analysis (PCA) explained visually with zero math," Medium. (May 6, 2023), [Online]. Available: https://towardsdatascience.com/principal-component-analysis-pca-explained-visually-with-zero-math-1cbf392b9e7d (visited on 09/22/2023).

[58] "Chapter 21 principal component analysis — data visualization." (), [Online]. Available: https://andrewirwin.github.io/data-visualization/pca.html (visited on 09/22/2023).

[59] "Enzyme classification." (), [Online]. Available: https://iubmb.qmul.ac.uk/enzyme/rules.html (visited on 09/25/2023).

[60] "Retrieve/ID mapping results — UniProtKB — UniProt." (), [Online]. Available: https://www.uniprot.org/id-mapping/uniprotkb/8b6adf2f748f15a3cadfb5c14overview (visited on 09/11/2023).

# Appendix A

# Appendix

## A.1 Input Data

## A.2 Summary tables

### A.2.1 Summary of Precision Test Results

### A.2.2 Summary of Repeatability Test Results

## A.3 Outputted Proteins Summaries

The outputted summaries can be found through the ID - mapping in the SwissProt database found here [60]

### A.3.1 Protein Summary of TSME

### A.3.2 Protein Summary of ITSv2

## A.4 Supplementary Items

| Input Protein Embedding Sample Data | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Protein Label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Q5P6B7 | 0.037 | 0.0785 | -0.0001564 | 0.0427 | 0.002745 | 0.01357 | -0.0608 | -0.05862 | -0.014244 | -0.05783 | -0.02118 |
| Q1J5K7 | -0.03217 | -0.05896 | 0.011086 | 0.01746 | -0.02524 | 0.02138 | 0.04514 | -0.1622 | -0.01776 | -0.07025 | -0.0557 |
| Q03GV8 | 0.004322 | 0.0802 | 0.03363 | -0.0397 | -0.05057 | 0.0773 | -0.0872 | -0.0635 | -0.05356 | 0.0079 | 0.02191 |
| Q12ZJ5 | -0.01236 | -0.0371 | 0.04706 | -0.02126 | 0.002844 | 0.0525 | -0.04803 | -0.08966 | 0.006393 | -0.03662 | -0.05933 |
| A3CJY9 | 0.005165 | -0.03503 | 0.02635 | 0.03198 | -0.01947 | 0.042 | -0.06174 | -0.0548 | -0.012985 | -0.03864 | 0.03894 |
| B2TTJ6 | 0.02258 | 0.1248 | 0.008 | 0.03488 | -0.005444 | 0.0432 | -0.02815 | -0.0386 | 0.001903 | -0.0817 | -0.03983 |
| A3M3G1 | 0.1024 | 0.02635 | -0.03345 | 0.02808 | -0.02483 | 0.0264 | -0.02834 | -0.0691 | -0.008125 | 0.01942 | -0.003672 |

Table A.1: Table showing an example of the input data for the brute force and metaheuristic algorithms. When extracted from the H5 file the data is in the form of key value pairs with the protein label and then an array of 1024 embedding values. Therefore the real data would have 1024 columns

| 25 Summary of the Precision Test | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Test Number | Algorithm | In/Above top 99.9th percentile | In/Above top 99.5th percentile | In/Above top 99th percentile | In/Above top 95th percentile | In/Above top 90th percentile | In/Above top 80th percentile | Highest Rank | Lowest Rank | Range |
| 1 | IATS | 0 | 4 | 6 | 12 | 17 | 20 | 277 | 66735 | 66458 |
| 2 | IATS | 0 | 3 | 5 | 10 | 17 | 20 | 237 | 70944 | 70707 |
| 3 | IATS | 0 | 5 | 5 | 12 | 20 | 20 | 301 | 54651 | 54350 |
| 1 | ITSv1 | 1 | 8 | 11 | 18 | 19 | 20 | 0 | 62048 | 62048 |
| 2 | ITSv1 | 1 | 9 | 10 | 18 | 20 | 20 | 1 | 34609 | 34608 |
| 3 | ITSv1 | 1 | 9 | 10 | 18 | 20 | 20 | 1 | 34609 | 34608 |

Table A.2: Table showing the summary results of the precision test.

| Repeatability Test Summary | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Test Number | Algorithm | Top 99.5% | Top 99% | Top 95% | Top 90% | Top 80% | SD | Range | Highest Rank | Lowest Rank |
| 1 | TSME | 1 | 3 | 3 | 3 | 4 | 160065.80 | 498163 | 191 | 498354 |
| 2 | TSME | 1 | 1 | 6 | 7 | 9 | 202952.73 | 551147 | 140 | 551287 |
| 3 | TSME | 1 | 2 | 4 | 5 | 8 | 181809.83 | 533296 | 942 | 534238 |
| 2 | ITSv2 | 0 | 0 | 3 | 4 | 7 | 162289.70 | 551596 | 9696 | 561292 |
| 3 | ITSv2 | 0 | 0 | 3 | 4 | 7 | 162289.70 | 551596 | 9696 | 561292 |
| 1 | ITSv2 | 0 | 0 | 3 | 4 | 7 | 162289.70 | 551596 | 9696 | 561292 |

Table A.3: Table showing algorithm performance summary in the repeatability tests.

Table A.4: Summary of Proteins of the TSME subset

| Entry | Protein names | Organism | Length | EC number | Gene Ontology (biological process) |
|---|---|---|---|---|---|
| P60687 | Na(+)/H(+) antiporter subunit D1 (Mnh complex subunit D1) | Staphylococcus aureus (strain MW2) | 498 | | ATP synthesis coupled electron transport [GO:0042773]; sodium ion transport [GO:0006814] |
| Q5BJT2 | Ubiquitin-like protein 3 (Membrane-anchored ubiquitin-fold protein) (MUB) | Rattus norvegicus (Rat) | 117 | | |
| O78747 | NADH-ubiquinone oxidoreductase chain 1 (EC 7.1.1.2) (NADH dehydrogenase subunit 1) | Ovis aries (Sheep) | 318 | 7.1.1.2 | |
| Q03GV8 | Uracil-DNA glycosylase (UDG) (EC 3.2.2.27) | Pediococcus pentosaceus (strain ATCC 25745 / CCUG 21536 / LMG 10740 / 183-1w) | 230 | 3.2.2.27 | Base-excision repair [GO:0006284] |

*Continued on next page*

| | | | | | |
|---|---|---|---|---|---|
| B7UPJ7 | Maltoporin (Maltose-inducible porin) | Escherichia coli O127:H6 (strain E2348/69 / EPEC) | 446 | | Monoatomic ion transport [GO:0006811] |
| Q17828 | ATP-dependent RNA helicase SUV3 homolog, mitochondrial (EC 3.6.4.13) | Caenorhabditis elegans | 719 | 3.6.4.13 | DNA duplex unwinding [GO:0032508]; mitochondrial RNA 3'-end processing [GO:0000965]; RNA catabolic process [GO:0006401] |
| Q29125 | Elafin (Protein WAP-1) | Sus scrofa (Pig) | 167 | | Antibacterial humoral response [GO:0019731]; innate immune response [GO:0045087]; negative regulation of peptidase activity [GO:0010466] |
| A9NGE2 | Polyribonucleotide nucleotidyltransferase (EC 2.7.7.8) (Polynucleotide phosphorylase) (PNPase) | Acholeplasma laidlawii (strain PG-8A) | 715 | 2.7.7.8 | mRNA catabolic process [GO:0006402]; RNA processing [GO:0006396] |

*Continued on next page*

| | | | | | |
|---|---|---|---|---|---|
| Q311U9 | Pantothenate synthetase (PS) (EC 6.3.2.1) (Pantoate–beta-alanine ligase) (Pantoate-activating enzyme) | Oleidesulfovibrio alaskensis (strain ATCC BAA-1058 / DSM 17464 / G20) (Desulfovibrio alaskensis) | 281 | 6.3.2.1 | Pantothenate biosynthetic process [GO:0015940] |
| I6VXS1 | Anticoagulant protein rhipilin-2 | Rhipicephalus haemaphysaloides (Tick) | 195 | | Envenomation resulting in negative regulation of voltage-gated potassium channel activity in another organism [GO:0044562]; negative regulation of peptidase activity [GO:0010466] |
| P56141 | Tryptophan synthase alpha chain (EC 4.2.1.20) | Helicobacter pylori (strain ATCC 700392 / 26695) (Campylobacter pylori) | 262 | 4.2.1.20 | Tryptophan biosynthetic process [GO:0000162] |
| Q830Y0 | Redox-sensing transcriptional repressor Rex 1 | Enterococcus faecalis (strain ATCC 700802 / V583) | 216 | | Negative regulation of DNA-templated transcription [GO:0045892]; response to redox state [GO:0051775] |

*Continued on next page*

| | | | | | |
|---|---|---|---|---|---|
| A4QLY3 | Large ribosomal subunit protein bL32c (50S ribosomal protein L32, chloroplastic) | Nasturtium officinale (Watercress) (Rorippa nasturtium-aquaticum) | 52 | | Translation [GO:0006412] |
| P30258 | Protamine Z3 (Scylliorhinine Z3) | Scyliorhinus canicula (Small-spotted catshark) (Squalus canicula) | 37 | | Cell differentiation [GO:0030154]; chromosome condensation [GO:0030261]; spermatogenesis [GO:0007283] |
| Q8NGZ6 | Olfactory receptor 6F1 (Olfactory receptor OR1-38) | Homo sapiens (Human) | 308 | | |
| Q661H4 | Aspartate–tRNA(Asp/Asn) ligase (EC 6.1.1.23) (Aspartyl-tRNA synthetase) (AspRS) (Non-discriminating aspartyl-tRNA synthetase) (ND-AspRS) | Borrelia garinii subsp. bavariensis (strain ATCC BAA-2496 / DSM 23469 / PBi) (Borrelia bavariensis) | 586 | 6.1.1.23 | Aspartyl-tRNA aminoacylation [GO:0006422] |

*Continued on next page*

| | | | | | |
|---|---|---|---|---|---|
| P18305 | Uncharacterized protein 443R | Invertebrate iridescent virus 6 (IIV-6) (Chilo iridescent virus) | 2432 | | |
| P20738 | Alpha-2-macroglobulin homolog (Alpha-2-M) | Pacifastacus leniusculus (Signal crayfish) | 32 | | Negative regulation of peptidase activity [GO:0010466] |
| A8Y987 | Minor capsid protein VP2 (Minor structural protein VP2) | Squirrel monkey polyomavirus | 332 | | Viral entry into host cell [GO:0046718]; viral penetration into host nucleus [GO:0075732] |
| Q3KSQ3 | Envelope glycoprotein H (gH) | Epstein-Barr virus (strain GD1) (HHV-4) (Human herpesvirus 4) | 706 | | Fusion of virus membrane with host plasma membrane [GO:0019064]; viral entry into host cell [GO:0046718] |

Table A.5: Summary of Proteins of the ITSv2 subset

| Entry | Protein names | Organism | Length | EC number | Gene Ontology (biological process) |
|---|---|---|---|---|---|
| Q9XDP0 | Protein-disulfide oxidoreductase DsbI | Lelliottia amnigena (Enterobacter amnigenus) | 221 | | protein folding [GO:0006457] |
| Q9V5Z7 | Aquaporin | Drosophila melanogaster (Fruit fly) | 245 | | multicellular organismal-level water homeostasis [GO:0050891]; renal water homeostasis [GO:0003091]; transmembrane transport [GO:0055085]; water transport [GO:0006833] |

*Continued on next page*

| | | | | | |
|---|---|---|---|---|---|
| Q9TUX7 | Pro-FMRFamide-related neuropeptide FF (FMRFamide-related peptides) [Cleaved into: Neuropeptide SF (NPSF); Neuropeptide FF (NPFF); Neuropeptide AF (NPAF)] | Bos taurus (Bovine) | 115 | | excitatory postsynaptic potential [GO:0060079]; neuropeptide signaling pathway [GO:0007218] |
| Q9SXS8 | Ethylene-responsive transcription factor 3 (Ethylene-responsive element-binding factor 3 homolog) (Ethylene-responsive element-binding factor 5) (EREBP-5) (NtERF5) | Nicotiana tabacum (Common tobacco) | 225 | | defense response [GO:0006952]; ethylene-activated signaling pathway [GO:0009873] |
| Q9PKV0 | Large ribosomal subunit protein bL28 (50S ribosomal protein L28) | Chlamydia muridarum (strain MoPn / Nigg) | 89 | | translation [GO:0006412] |

*Continued on next page*

| | | | | | |
|---|---|---|---|---|---|
| Q9NS84 | Carbohydrate sulfotransferase 7 (EC 2.8.2.-) (EC 2.8.2.17) (Chondroitin 6-sulfotransferase 2) (C6ST-2) (Galactose/N-acetylglucosamine/N-acetylglucosamine 6-O-sulfotransferase 5) (GST-5) (N-acetylglucosamine 6-O-sulfotransferase 4) (GlcNAc6ST-4) (Gn6st-4) | Homo sapiens (Human) | 486 | 2.8.2.-; 2.8.2.17 | chondroitin sulfate biosynthetic process [GO:0030206]; N-acetylglucosamine metabolic process [GO:0006044]; polysaccharide metabolic process [GO:0005976]; sulfur compound metabolic process [GO:0006790] |

*Continued on next page*

| | | | | | |
|---|---|---|---|---|---|
| Q9JHK7 | Interleukin-10 (IL-10) (Cytokine synthesis inhibitory factor) (CSIF) | Marmota monax (Woodchuck) | 178 | | immune response [GO:0006955]; negative regulation of B cell proliferation [GO:0030889]; negative regulation of cytokine production involved in immune response [GO:0002719]; negative regulation of inflammatory response [GO:0050728]; negative regulation of interleukin-6 production [GO:0032715]; negative regulation of membrane protein ectodomain proteolysis [GO:0051045]; positive regulation of B cell apoptotic process [GO:0002904]; positive regulation of cytokine production [GO:0001819]; positive regulation of DNA-binding transcription factor activity [GO:0051091]; positive regulation of DNA-templated transcription [GO:0045893]; |

| | | | | | |
|---|---|---|---|---|---|
| Q9I9P7 | Extracellular fatty acid-binding protein (Ex-FABP) (Lipocalin Q83) | Coturnix japonica (Japanese quail) (Coturnix coturnix japonica) | 178 | | defense response to bacterium [GO:0042742]; innate immune response [GO:0045087] |
| Q9GPJ1 | Protein Skeletor, isoforms D/E | Drosophila melanogaster (Fruit fly) | 1503 | | cell division [GO:0051301]; meiotic cell cycle [GO:0051321]; nucleus organization [GO:0006997]; spindle assembly [GO:0051225] |
| Q9FFX1 | B3 domain-containing protein At5g38500 | Arabidopsis thaliana (Mouse-ear cress) | 411 | | |
| Q9F9L1 | Alanine racemase (EC 5.1.1.1) | Piscirickettsia salmonis | 166 | 5.1.1.1 | D-alanine biosynthetic process [GO:0030632] |
| Q9BZM1 | Group XIIA secretory phospholipase A2 (GXII sPLA2) (sPLA2-XII) (EC 3.1.1.4) (Phosphatidylcholine 2-acylhydrolase 12A) | Homo sapiens (Human) | 189 | 3.1.1.4 | arachidonic acid secretion [GO:0050482]; lipid catabolic process [GO:0016042]; phospholipid metabolic process [GO:0006644] |

*Continued on next page*

| | | | | | |
|---|---|---|---|---|---|
| Q96BJ8 | Engulfment and cell motility protein 3 | Homo sapiens (Human) | 720 | | actin filament organization [GO:0007015]; apoptotic process [GO:0006915]; cell motility [GO:0048870]; phagocytosis [GO:0006909] |

*Continued on next page*

| | | | | | |
|---|---|---|---|---|---|
| Q92847 | Growth hormone secretagogue receptor type 1 (GHS-R) (GH-releasing peptide receptor) (GHRP) (Ghrelin receptor) | Homo sapiens (Human) | 366 | | actin polymerization or depolymerization [GO:0008154]; adult feeding behavior [GO:0008343]; cellular response to insulin stimulus [GO:0032869]; cellular response to insulin-like growth factor stimulus [GO:1990314]; cellular response to lipopolysaccharide [GO:0071222]; cellular response to thyroid hormone stimulus [GO:0097067]; decidualization [GO:0046697]; G protein-coupled receptor signaling pathway [GO:0007186]; ghrelin secretion [GO:0036321]; growth hormone secretion [GO:0030252]; hormone-mediated signaling pathway [GO:0009755]; insulin-like growth factor receptor signaling pathway [GO:0048009]; learning or memory [GO:0007611]; |

| | | | | | |
|---|---|---|---|---|---|
| Q8YUT2 | Gas vesicle protein K (GvpK) | Nostoc sp. (strain PCC 7120 / SAG 25.82 / UTEX 2576) | 155 | | gas vesicle organization [GO:0031412] |
| Q8ERT7 | Global transcriptional regulator Spx 1 | Oceanobacillus iheyensis (strain DSM 14371 / CIP 107618 / JCM 11309 / KCTC 3954 / HTE831) | 131 | | negative regulation of DNA-templated transcription [GO:0045892] |
| Q8EMI3 | Fructose-1,6-bisphosphatase class 3 (FBPase class 3) (EC 3.1.3.11) (D-fructose-1,6-bisphosphate 1-phosphohydrolase class 3) | Oceanobacillus iheyensis (strain DSM 14371 / CIP 107618 / JCM 11309 / KCTC 3954 / HTE831) | 644 | 3.1.3.11 | gluconeogenesis [GO:0006094] |
| Q89FP2 | Thiazole synthase (EC 2.8.1.10) | Bradyrhizobium diazoefficiens (strain JCM 10833 / BCRC 13528 / IAM 13628 / NBRC 14792 / USDA 110) | 260 | 2.8.1.10 | thiamine diphosphate biosynthetic process [GO:0009229] |

*Continued on next page*

| | | | | | |
|---|---|---|---|---|---|
| Q80TS5 | Zinc finger protein 423 (Early B-cell factor-associated zinc finger protein) (Olf1/EBF-associated zinc finger protein) (Smad- and Olf-interacting zinc finger protein) | Mus musculus (Mouse) | 1292 | | brown fat cell differentiation [GO:0050873]; cerebellar granule cell precursor proliferation [GO:0021930]; cilium assembly [GO:0060271]; negative regulation of cold-induced thermogenesis [GO:0120163]; negative regulation of DNA-templated transcription [GO:0045892]; Notch signaling pathway [GO:0007219]; positive regulation of BMP signaling pathway [GO:0030513]; positive regulation of DNA-templated transcription [GO:0045893]; protein localization to cilium [GO:0061512]; regulation of transcription by RNA polymerase II [GO:0006357]; smoothened signaling pathway involved in regulation of cerebellar granule cell precursor cell proliferation [GO:0021938]; |

| | | | | | |
|---|---|---|---|---|---|
| Q7YR44 | Corneodesmosin (S protein) | Pan troglodytes (Chimpanzee) | 529 | | cell-cell adhesion [GO:0098609]; skin morphogenesis [GO:0043589] |

| Tested parameters for Algorithms | | |
|---|---|---|
| TestNum | Algorithm | Parameters |
| 1 | IATS | 'num_proteins': 20, 'max_iterations': 100, 'local_iterations': 50 |
| 2 | IATS | 'num_proteins': 20, 'max_iterations': 500, 'local_iterations': 50 |
| 3 | IATS | 'num_proteins': 20, 'max_iterations': 750, 'local_iterations': 50 |
| 4 | IATS | 'num_proteins': 20, 'max_iterations': 1000, 'local_iterations': 50 |
| 5 | IATS | 'num_proteins': 20, 'max_iterations': 500, 'local_iterations': 100 |
| 6 | IATS | 'num_proteins': 20, 'max_iterations': 1000, 'local_iterations': 100 |
| 7 | IATS | 'num_proteins': 20, 'max_iterations': 1000, 'local_iterations': 150 |
| 8 | IATS | 'num_proteins': 20, 'max_iterations': 1000, 'local_iterations': 200 |
| 1 | TSME | 'dna_size': 20, 'max_epochs': 1000, 'local_iterations': 500, 'population_size': 500, 'retain_percent': 0.05 |
| 2 | TSME | 'dna_size': 20, 'max_epochs': 5000, 'local_iterations': 500, 'population_size': 500, 'retain_percent': 0.05 |
| 3 | TSME | 'dna_size': 20, 'max_epochs': 1000, 'local_iterations': 500, 'population_size': 1000, 'retain_percent': 0.05 |
| 4 | TSME | 'dna_size': 20, 'max_epochs': 1000, 'local_iterations': 1000, 'population_size': 500, 'retain_percent': 0.05 |
| 5 | TSME | 'dna_size': 20, 'max_epochs': 1000, 'local_iterations': 500, 'population_size': 1000, 'retain_percent': 0.05 |
| 6 | TSME | 'dna_size': 20, 'max_epochs': 2000, 'local_iterations': 1000, 'population_size': 500, 'retain_percent': 0.05 |
| 7 | TSME | 'dna_size': 20, 'max_epochs': 1000, 'local_iterations': 500, 'population_size': 1000, 'retain_percent': 0.05 |
| 1 | ITSv1 | 'num_proteins': 20, 'max_iterations': 100 |
| 2 | ITSv1 | 'num_proteins': 20, 'max_iterations': 500 |
| 3 | ITSv1 | 'num_proteins': 20, 'max_iterations': 50 |
| 3 | ITSv1 | 'num_proteins': 20, 'max_iterations': 1000 |
| 3 | ITSv1 | 'num_proteins': 20, 'max_iterations': 600 |
| 4 | ITSv1 | 'num_proteins': 20, 'max_iterations': 50 |
| 4 | ITSv1 | 'num_proteins': 20, 'max_iterations': 250 |
| 5 | ITSv1 | 'num_proteins': 20, 'max_iterations': 750 |
| 1 | ITSv1 | 'num_proteins': 20, 'max_iterations': 500, 'local_iterations': 50, 'local_sample_sizes': 500 |
| 2 | ITSv2 | 'num_proteins': 20, 'max_iterations': 1000, 'local_iterations': 100, 'local_sample_sizes': 100 |
| 3 | ITSv2 | 'num_proteins': 10, 'max_iterations': 500, 'local_iterations': 50, 'local_sample_sizes': 100 |
| 4 | ITSv2 | 'num_proteins': 20, 'max_iterations': 500, 'local_iterations': 100, 'local_sample_sizes': 100 |
| 5 | ITSv2 | 'num_proteins': 20, 'max_iterations': 1000, 'local_iterations': 100, 'local_sample_sizes': 100 |
| 6 | ITSv2 | 'num_proteins': 20, 'max_iterations': 500, 'local_iterations': 100, 'local_sample_sizes': 200 |
| 6 | ITSv2 | 'num_proteins': 20, 'max_iterations': 500, 'local_iterations': 500, 'local_sample_sizes': 100 |
| 7 | ITSv2 | 'num_proteins': 20, 'max_iterations': 50, 'local_iterations': 1000, 'local_sample_sizes': 100 |