Group No. 3

Multiple Sequence Alignment Using Tabu Search

A Project report submitted as a part of the external evaluation process for the course **19BIO112 : Intelligence of Biological Systems 4**

Team Members :

Name	Roll Number
Aditya Anil	AM.EN.U4AIE19006
Anand Balaji	AM.EN.U4AIE19011
Anjith Prakash	AM.EN.U4AIE19012
Namitha S	AM.EN.U4AIE19042
Sreerag K Vivek	AM.EN.U4AIE19064

Acknowledgement

In performing our project, we had to take the help and guideline of some respected persons, who deserve our greatest gratitude. The completion of this project gives us much pleasure. We would like to show our gratitude to **Dr. Georg Gutjahr**, Course Instructor, for giving us a good guideline for assignment throughout numerous consultations. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us.

Many people, especially our classmates and team members, have made valuable comments and suggestions on this proposal which gave us inspiration to improve our project . We thank all the people for their help directly and indirectly to complete the task on time.

ADITYA ANIL (AM.EN.U4AIE19006)
ANAND BALAJI (AM.EN.U4AIE19011)
ANJITH PRAKASH (AM.EN.U4AIE19012)
NAMITHA S (AM.EN.U4AIE19042)
SREERAG K VIVEK (AM.EN.U4AIE19064)

Contents Page No.

Abstract

Tabu search is a meta-heuristic method for solving combinatorial optimization problems. The paper will be aiming on multiple sequence alignment of given sequences using Tabu Search. We will be converting the MSA problem into a travelling salesman problem and implementing the tabu search on it. Tabu search guarantees to give a near optimal solution to TSP. Tabu search gives the tabu list which will be used on the sequence to give the final alignment sequence .

Introduction

The project will be using the concepts of tabu Search and Travelling salesman problem to solve the problem of multiple sequence alignment .

Tabu Search

Tabu search is a meta-heuristic technique that has been proven to be an effective solution for combinatorial optimization problems. A meta-heuristic is a general strategy for guiding and controlling actual heuristics. Tabu Search is often thought of as incorporating memory structures into local search strategies. Since local search has many drawbacks, Tabu Search is designed to address many of them. Tabu Search's basic concept is to penalise steps that carry the solution into previously visited search spaces. The method of tabu search provides a solution very close to superiority and is among the most effective ways to tackle the difficult problems at hand. This has made tabu Search extremely common among those looking for the best solutions to large combinatorial problems found in a variety of realistic settings.
We will employ Tabu search in our technique, which will use a local or neighbourhood search strategy to iteratively shift from one possible solution to an improved solution in the neighbourhood until some stopping requirement is fulfilled. As the search advances, Tabu search carefully investigates the neighbourhood of each solution.The solutions admitted to the new neighborhood are determined through the use of memory structures. The search advances by repeatedly shifting from the present solution to a better solution using these memory structures. These memory structures make up the tabu list, which is a collection of rules and forbidden solutions used to determine which solutions will be permitted to the neighbourhood.

The memory structures used in tabu search can roughly be divided into three categories:

- Short-term: The list of solutions recently considered. If a potential solution appears on the tabu list, it cannot be revisited until it reaches an expiration point.

- Intermediate-term: Intensification rules intended to bias the search towards promising areas of the search space.
- Long-term: Diversification rules that drive the search into new regions

Multiple sequence alignment

Multiple sequence alignment is a well-known technique in molecular biology. MSA is simply a method of organising data in such a way that identical sequence features are matched together. Any related biological material, such as structure, function, or homology to the common ancestor, may be considered an attribute. The aim is to either uncover characteristics that may be shared by several sequences or to recognise changes that may explain functional and phenotypic heterogeneity. It has a broad variety of uses, including detecting motifs in biological genomes, tracing evolutionary pathways using sequence similarities, defining the consensus sequence, and predicting secondary and tertiary structures. Methods such as sequence alignment are used to detect and quantify similarities between different DNA and protein sequences that may have evolved from a common ancestor.

Travelling Salesman Problem

Travelling Salesman Problem can be modelled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. It is a minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once. Often, the model is a complete graph (i.e., each pair of vertices is connected by an edge). If no path exists between two cities, adding a sufficiently long edge will complete the graph without affecting the optimal tour. There are various alignment methods used within multiple sequences to maximize scores and correctness of alignments. Each is usually based on a certain heuristic with an insight into the evolutionary process. Most try to replicate evolution to get the most realistic alignment possible to best predict relations between sequences.

Implementation

In our project , we have implemented a method in which we convert the data sequences into a graph in the form of a traveling salesman problem and then use a tabu search algorithm to find the best score so that we can find an optimal solution for the Multiple Sequence Alignment for the given sequences.

Tabu Search Algorithm :

Here we keep track of recent searches and place them in a tabu list so that the algorithm may investigate alternative options. The steps we took to implement this method are as follows. :

1. Choose random initial state where the algorithm should start
2. Enters in a loop. This loop will continue searching for an optimal solution until a user-specified stopping condition is met
3. Create an empty candidate list , where each of the candidates in a given neighbour which does not contain a tabu element are added to this empty candidate list.
4. The algorithm keeps track of the best solution in the neighbourhood, that is not tabu.
5. If its cost is better than the current best ,it's marked as a solution.
6. If the best local candidate has a higher fitness value than the current best, it is set as the new best.(The term "fitness" refers to an assessment of a candidate solution as reflected in a mathematical optimization objective function.)
7. The best local candidate is always added to the tabu list, and if the tabu list is already full, some components are eliminated.
8. This procedure is repeated until the user-specified stopping condition is reached, at which time the best solution found throughout the search is returned.

Tabu search pseudocode

```
tabuList ← []
tabuList.push(s0)
while (not stoppingCondition())
    sNeighborhood ← getNeighbors(bestCandidate)
    bestCandidate ← sNeighborhood[0]
    for (sCandidate in sNeighborhood)
        if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) >
fitness(bestCandidate)) )
            bestCandidate ← sCandidate
        end
    end
    if (fitness(bestCandidate) > fitness(sBest))
        sBest ← bestCandidate
    end
    tabuList.push(bestCandidate)
    if (tabuList.size > maxTabuSize)
        tabuList.removeFirst()
    end
end
return sBest
```

There is a primitive short-term memory in this implementation, but no intermediate or long-term memory structures.
Fitness function : A fitness function is a mathematical function that returns a score or determines whether or not the aspiration requirements are met.

Procedure

Our project aims at multiple sequence alignment of a given input sequence . The process involves conversion of the sequence alignment problem into a traveling salesman problem where a graph is formed from the given sequences. This graph will be used up by the tabu algorithm in determining the best solution as a tabu list . A tabu list is a short-term collection of options that have been explored recently (less than n iterations ago, where n is the number of previous solutions to be stored is also called the tabu tenure)..A tabu list is a collection of solutions that have evolved as a result of the process of going from one solution to the next. This tabu list will also be utilised in the align_output_sequences function, which will utilise it to locate gaps in our final output sequence and insert a '-' in the locations of the gaps. .

Fig : Observation table

Input Sequence	Output Sequence
GARFIELDTHELASTFATCAT	GARFIELDTHEL----ASTFATCAT
GARFIELDTHEFASTCAT	GARFIELDTHEV----AS---TCAT
GARFIELDTHEVERYFASTCAT	GARFIELDTHEVERYFAS---TCAT
THEFATCAT	--------TH--E--FA----TCAT
GARFIELDTHEVASTCAT	GARFIELDTH--E--FA---STCAT

As shown in the observation table we can see how the sequence alignment is being done to the input sequence to form the set of strings with aligned letters. Using the '-' character , we have made sure to get a maximum alignment of the sequences possible. According to the output sequence , we are able to align the letter 'TH', 'A' and 'TCAT in all the output sequences . Although due to the lack of a few other letters , we were not able to align any more , for all the sequence .Different metaheuristic approaches might show different alignments , although .We have also analysed the efficiency of the multiple sequence alignment done using the tabu search algorithm . From the score calculated , we have got a score of 83% which is comparatively a good result .Hence we conclude that this might be one of the best methods possible for problems related to sequence alignment .

Application

In reality there is a great variety of discrete optimization problems. There are several various kinds of problems, such as assignment type problems and sequencing problems, in the field of production planning. Problems of this nature can arise in a variety of cases, each with its own set of variations . Since the majority of these problems are NP-hard, heuristics are the most effective way to solve them. Metaheuristics such as tabu quest have been used to solve a wide range of problems. Tabu Search algorithm can be used mainly for metaheuristic problems. Here we have referred to solving certain sample problems like N-Queens Problem, Traveling Salesman Problem (TSP), Minimum Spanning Tree (MST), Assignment Problems, Vehicle Routing and DNA Sequencing.

Some of the main Tabu Search Applications were Employee Scheduling, Character recognition, Space planning and architectural design, Telecommunications path assignment and Probabilistic logic problems.

Conclusion

The results reported in this paper show the superior capability of our improved tabu search compared to other methods. Efficient mechanisms to neighborhood structure generation and other features such as intensification and diversification strategies are the key of this improvement. As a perspective of this work, the improvement of the start solution by a specific heuristic is desired. In addition, we can integrate other mechanisms to the neighborhood generation step or incorporate other strategies in the intensification/ diversification phase to improve the quality of the solution.

Here we have demonstrated that tabu characteristics of the tabu search help it to ding such good solutions that are otherwise hard to be reached.Experiments show that tabu search is not only good at finding solutions for test cases that are easy to align, but it also performs well for the test cases that are relatively hard to align.Tabu search comes with a number of parameters that can be experimented with to observe the respective effect on the search process. The parameters like tabu list size, tabu tenure, termination criteria, and neighborhood size can have a direct influence on the quality of the final alignment.

With a larger number of sequences in test cases, the problem of multiple alignment turns into a large-scale combinatorial problem. A huge number of solution combinations need to be evaluated that causes the software to take a long time to generate the alignment. This problem can be overcome by introducing parallel computing in the algorithm.

Reference

1. M. Zachariasen and M. Dam, “Tabu Search on the Geometric Traveling Salesman Problem,” *Meta-Heuristics*, pp. 571–587, 1996.
2. T. A. R. I. Q. RIAZ, W. A. N. G. YI, and K. U. O.-B. I. N. LI, “A TABU SEARCH ALGORITHM FOR POST-PROCESSING MULTIPLE SEQUENCE ALIGNMENT,” *Journal of Bioinformatics and Computational Biology*, vol. 03, no. 01, pp. 145–156, 2005.
3. J. Taheri and A. Y. Zomaya, “RBT-GA: a novel metaheuristic for solving the multiple sequence alignment problem,” BMC Genomics, 07-Jul-2009. [Online]. Available: https://bmcgenomics.biomedcentral.com/articles/10.1186/1471-2164-10-S1-S10. [Accessed: 22-May-2021].
4. I. Alkallak and R. Sha’ban, “Tabu Search Method for Solving the Traveling Salesman Problem,” *AL-Rafidain Journal of Computer Sciences and Mathematics*, vol. 5, no. 2, pp. 141–153, 2008.
5. W. B. Carlton and J. W. Barnes, “Solving the Traveling-Salesman Problem with Time Windows Using Tabu Search,” *IIE Transactions*, vol. 28, no. 8, pp. 617–629, 1996.
6. “Tabu Search Algorithm,” *Tabu Search Algorithm - an overview | ScienceDirect Topics*. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/tabu-search-algorithm. [Accessed: 22-May-2021].
7. “Multiple Sequence Alignment Using Tabu Search,” *CRPIT 29:223-232 - Multiple Sequence Alignment Using Tabu Search*. [Online]. Available: https://crpit.scem.westernsydney.edu.au/abstracts/CRPITV29Riaz.html. [Accessed: 22-May-2021].
8. Bahr, A., Thompson, J. D., Thierry, J. C., and Poch, O. (2001): BAliBASE (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. Nucleic Acids Res., 29(1): 323-6.
9. Fred Glover (1990). "Tabu Search: A Tutorial". Interfaces.

Code:

```
import math
import time
from itertools import combinations
import sys
from random import randint
from random import shuffle
def MSA_to_TSP(sequences):
    node = []
    for i in range(len(sequences)):
        node.append(sequences[i])
    graph = [[0.0]*len(node) for i in range(len(node))]
    max_score = 0.0
    for i in range(len(node)):
        graph[i][i] = 0.0
        for j in range(i+1,len(node)):
            score, align1, align2 = get_alignment_score(node[i],node[j])
            graph[i][j] = score
            graph[j][i] = score
            if max_score < score:
                max_score = score
    for i in range(len(node)):
        for j in range(i+1,len(node)):
            graph[i][j] = max_score - graph[i][j] + 1
            graph[j][i] = max_score - graph[j][i] + 1
    return graph
def get_alignment_score(v, w, match_penalty=1, mismatch_penalty=-1,
deletion_penalty=-1):
    n1 = len(v)
    n2 = len(w)
    s = [[0.0]*(n2+1) for i in range(n1+1)]
    b =[[0.0]*(n2+1) for i in range(n1+1)]
    for i in range(n1+1):
        s[i][0] = i * deletion_penalty
        b[i][0] = 1

    for j in range(n2+1):
        s[0][j] = j * deletion_penalty
        b[0][j] = 2
    for i in range(1,n1+1):
        for j in range(1,n2+1):
            if v[i-1] == w[j-1]:
                ms = s[i-1][j-1] + match_penalty
```

```
            else:
                ms = s[i-1][j-1] + mismatch_penalty

            test = [ms, s[i-1][j] + deletion_penalty, s[i][j-1] +
deletion_penalty]
            p = max(test)
            s[i][j] = p
            b[i][j] = test.index(p)
    i = n1
    j = n2
    sv = []
    sw = []
    while i != 0 or j != 0:
        p = b[i][j]
        if p==0:
            i-=1
            j-=1
            sv.append(v[i])
            sw.append(w[j])
        elif p == 1:
            i-=1
            sv.append(v[i])
            sw.append("-")
        elif p == 2:
            j-=1
            sv.append("-")
            sw.append(w[j])
        else:
            break

    sv.reverse()
    sw.reverse()

    return (s[n1][n2], "".join(sv), "".join(sw))
def parse_data(linkset):
    links = {}
    max_weight = 0

    for tmp in linkset:
        if int(tmp[2]) > max_weight:
            max_weight = int(tmp[2])
    for link in linkset:
        try:
            linklist = links[str(link[0])]
            linklist.append(link[1:])
            links[str(link[0])] = linklist
```

```
        except:
            links[str(link[0])] = [link[1:]]

    return links, max_weight
def getNeighbors(state):
    return two_opt_swap(state)
def hill_climbing(state):
    node = randint(1, len(state)-1)
    neighbors = []

    for i in range(len(state)):
        if i != node and i != 0:
            tmp_state = state.copy()
            tmp = tmp_state[i]
            tmp_state[i] = tmp_state[node]
            tmp_state[node] = tmp
            neighbors.append(tmp_state)

    return neighbors

def two_opt_swap(state):
    global neighborhood_size
    neighbors = []

    for i in range(neighborhood_size):
        node1 = 0
        node2 = 0

        while node1 == node2:
            node1 = randint(1, len(state)-1)
            node2 = randint(1, len(state)-1)

        if node1 > node2:
            swap = node1
            node1 = node2
            node2 = swap
        tmp = state[node1:node2]
        tmp_state = state[:node1] + tmp[::-1] +state[node2:]
        neighbors.append(tmp_state)

    return neighbors
def fitness(route, graph):
    path_length = 0

    for i in range(len(route)):
        if(i+1 != len(route)):
```

```
            dist = weight_distance(route[i], route[i+1], graph)
            if dist != -1:
                path_length = path_length + dist
            else:
                return max_fitness # there is no  such path
        else:
            dist = weight_distance(route[i], route[0], graph)
            if dist != -1:
                path_length = path_length + dist
            else:
                return max_fitness # there is no  such path

    return path_length

# not used in this code but some datasets has 2-or-more dimensional data
points, in this case it is usable
def euclidean_distance(city1, city2):
    return math.sqrt((city1[0] - city2[0])**2 + ((city1[1] -
city2[1])**2))

def weight_distance(city1, city2, graph):
    global max_fitness

    neighbors = graph[str(city1)]

    for neighbor in neighbors:
        if neighbor[0] == int(city2):
            return neighbor[1]

    return -1 #there can't be minus distance, so -1 means there is not
any city found in graph or there is not such edge
def tabu_search(input_file_path):
    global max_fitness, start_node
    graph, max_weight = parse_data(input_file_path)

    ## Below, get the keys (node names) and shuffle them, and make
start_node as start
    s0 = list(graph.keys())
    shuffle(s0)

    if int(s0[0]) != start_node:
        for i in range(len(s0)):
            if int(s0[i]) == start_node:
                swap = s0[0]
                s0[0] = s0[i]
                s0[i] = swap
```

```
                break
    # max_fitness will act like infinite fitness
    max_fitness = ((max_weight) * (len(s0)))+1
    sBest = s0
    vBest = fitness(s0, graph)
    bestCandidate = s0
    tabuList = []
    tabuList.append(s0)
    stop = False
    best_keep_turn = 0

    while not stop :
        sNeighborhood = getNeighbors(bestCandidate)
        bestCandidate = sNeighborhood[0]
        for sCandidate in sNeighborhood:
            if (sCandidate not in tabuList) and ((fitness(sCandidate,
graph) < fitness(bestCandidate, graph))):
                bestCandidate = sCandidate

        if (fitness(bestCandidate, graph) < fitness(sBest, graph)):
            sBest = bestCandidate
            vBest = fitness(sBest, graph)
            best_keep_turn = 0

        tabuList.append(bestCandidate)
        if (len(tabuList) > maxTabuSize):
            tabuList.pop(0)

        if best_keep_turn == stoppingTurn:
            stop = True

        best_keep_turn += 1

    return sBest, vBest

maxTabuSize = 10000
neighborhood_size = 500
stoppingTurn = 500
max_fitness = 0
start_node = 0

def find_gap_indices(A, alignedA):
    i = 0
    j = 0
    pointer = []
    while j < len(alignedA):
```

```
        if alignedA[j] == '-' and (i > len(A) or A[i] != '-'):
            pointer.append(j)
            j += 1
        else:
            j += 1
            i += 1
    return pointer

def insert_gaps(S,gap_indices_for_A):
    copy_of_S = S
    if len(gap_indices_for_A) > 0 and len(gap_indices_for_A) > 0:
        gap_indices_for_A.sort()
        for i in gap_indices_for_A:
            copy_of_S = (copy_of_S[0:i]+'-')+copy_of_S[i:]
    return copy_of_S

def output_sequences(sequences, order) :
    aligned_sequences = []

    for i in order:
        aligned_sequences.append(sequences[i])

    for i in range(len(aligned_sequences)-1):
        A = aligned_sequences[i]
        B = aligned_sequences[i+1]
        score, alignedA, alignedB = get_alignment_score(A,B)
        gap_indices_for_A = find_gap_indices(A, alignedA)
        for j in range(i):
            S = aligned_sequences[j]
            newly_alinged_S = insert_gaps(S,gap_indices_for_A)
            aligned_sequences[j] = newly_alinged_S

        aligned_sequences[i] = alignedA
        aligned_sequences[i+1] = alignedB

    return aligned_sequences

def getdist(data):
    out = []
    for i in range(len(data)):
        idx = len(data[i])
        for j in range(idx):
            if i != j:
                out.append(f"{i} {j} {data[i][j]}")
    return out
```

```
def score(sequences):
    t = len(sequences)
    k = len(sequences[0])
    score = 0
    for i in range(t):
        A = sequences[i]
        for j in range(i+1,t):
            B = sequences[j]
            for idx in range(k):
                if A[idx] != B[idx]:
                    score += 1
    return score

if __name__ == "__main__":
    x = []
    input_file = sys.argv[1]
    f = open(input_file, "r")
    seq = f.read().strip("\n").split("\n")
    tspdata = MSA_to_TSP(seq)
    newdata = getdist(tspdata)
    x, y = tabu_search(newdata)
    x = list(map(int, x))
    aligned_sequences = output_sequences(seq, x)
    for i in range(len(aligned_sequences)):
        print(aligned_sequences[i])
    print(f"Score: {score(aligned_sequences)}")
```