International Journal of Research in Engineering, Science and Management
Volume-2, Issue-10, October-2019
www.ijresm.com | ISSN (Online): 2581-5792

On the Spaced Repetition Scheduling Problem

Rohan Gupta
Student, Department of Mathematics and Computer Science, TISB, Bangalore, India

***Abstract*: In this paper, we investigate solutions to the Spaced Repetition Scheduling problem, a combinatorial optimization problem pertinent to the fields of behavioral and brain science, among others. The problem, involving placement of elements, each with a certain benefit, from a fixed set and allowed spaced repetition, asks to maximize the net benefit in a given integral time, with each element taking up unit time. The problem is explored through the lens of complex networks, by remodeling it as graphs and subgraphs. Using the well-defined tools within network science, we analyze the problem and hence find directed clusters as a heuristic for element placement.**

***Keywords*: scheduling problem, complex network, graph theory**

1. Introduction

The Spaced Repetition Scheduling problem is a combinatorial optimization problem which involves the selection and placement of k elements in S_1, the final set of elements, from set S containing up to k unique elements. For each element u_i in S_1, placement is constrained by $u_i[1]$, denoting the minimum distance in number of elements between two instances of u_i (notated as interval). The objective, given a k and S, is to create an S_1 that maximizes sum of benefit (u[0]), given by $max \sum_{i=0}^{k} u_i[0]$.

2. Problem

A. Example

With the problem defined as in §1, we note an arbitrary example to elucidate it as follows:

S = {a, b, c, d}, where
a = {30, 4}, b = {20, 3}, c = {10, 2}, d = {4, 1}
k = 3
We define S_1 as the final set of elements, with $|S_1| = k$.
A valid permutation is (a, c, b)
However, (a, c, a) is invalid, since a[1] = 4 and the distance between two consecutive a's in this configuration is 2 (< 4)

B. Motivation

Spaced repetition is a technique for learning which uses repeated study of content following a schedule determined by a spaced repetition algorithm. A field of research in the behavioral and brain sciences, spaced repetition is used to improve long-term retention of information as well as aid in language learning.

Furthermore, it is likely that in machine scheduling problems, job-shop scheduling problems, or variants of scheduling problems pertinent to machine or processor scheduling, such a configuration could arise (with intervals representing time between usage of machines or resources).

Despite promising results based on [1] and [2], it is noted that a quick literature survey returns few results for algorithms on scheduling spaced repetitions.

Therefore, I chose to investigate solutions to the problem.

3. Approach

Since the problem in itself is NP-Hard (reducible to bounded Knapsack problem), it is imperative to consider computationally cheap heuristics in my approach.

My learning-based approach involves:

- Finding likely pairs
- Finding likely clusters of size k

A. Procedure to generate elementary set

My approach to each data point given, in the ordered set format (S_1), is as follows

Find the next unique element, u, with index i

- Find next instance of this element, with index j
- Extract subset N from set S_1, on the interval [i, j)

With subset N extracted, we can transmute the sub-problem into a graph.

- First, consider u as the root of a directed graph G = {V, E}
- Create edges for each element in N as follows:

$$\forall\, k \in N \; create \; edges \; \{u, k\}$$

- For each edge, assign edge weight as

$$\forall k \in N$$

$$weight = 1/(index(k) - index(u) \; * \; (j \; - \; i)\,)$$

Hence, for the set {a, b, c, d, a} the first subset is{a, b, c, d}; there exists a graph with the following vertices and edges:

V = {a, b, c, d}
E = {{a,b -1/8}, {a,c - 1/12}, {a,d - 1/16}]

B. Rationale

Essentially, for the outdegree of a specific instance of a, we calculate its 'attachment' to each other node. It is sufficient to arbitrarily assign attachment as (1/k) since the numerical value

of edge weight holds no significance and is only used to compare between elements to assign a ranking.

Furthermore, since distance in elements between all instances of a particular element is not necessarily constant (but bounded below by a[1] for any element a), we must normalize weight by multiplying it by 1/(j-i), essentially weighting attachments in wider separated (sparser) instances lower.

Thus, we now have, for one instance of a particular element, a measure of attachment to all the elements between it and the next instance. Of course, it is imperative to realize that these elements and their permutations need not stay constant across instances.

Hence, we repeat for all instances of the element (say a), keeping in mind three procedures:

- After finding the edge set for each graph with root node a, for each common pair, we take a cumulative mean. For example, say out of 25 edge sets, 20 of them contain {a,b}, we take mean of the weights of all these 20, to obtain the average 'attachment' of a with b.
- If {a,n}, for any n representing an element, occurs more than once (in varying positions) in one sub-set, add their weights.
- If an edge does not exist in the edge set, add it to the set

We repeat the same procedure for every element in the set (including their instances), to obtain a comprehensive edge set from each element, representing a set of weighted outdegrees of each element.

It is critical to note that edge, say {a,b} will not have the same weight as {b,a} due to the graph being directed. This is relevant in considering clustering as directed, non-commutative clustering.

Similarly, repeat for each new data point (ensuring that the elements are all from the same set S), and compute averages accordingly.

Thus, each weight per edge represents the average attachment to the second element in the pair of vertices. Since a uniform procedure has been applied to each element, we can consider the edge with highest weightage in the edge set for each element to be the most favorable pair.

C. *Extension*

Now, a suitable method to pair elements has been found.

So, upon fixing a starting element, one can search the edge pair most 'attached' to that element which also satisfies the interval restriction, and then repeat for the second element, and so on.

Additionally, once the edge set, along with weights has been calculated, accession is O(1) and therefore generating a new list of size k, with starting element fixed is O(k). It is also clear to see that generation of the edge set itself does not take up exponential time.

A problem arises when intending to find the initial element. One can naively compute the mode of all first elements in given data and select that. Alternatively, the element with the highest benefit: interval ratio can be chosen. It is also likely (conjectured) that as k, the size of S_1, grows large, the starting element will tend not to matter in improving benefit.

D. *Clustering of size > 2*

For analytical purposes, or otherwise, it might be relevant to compute likelihood of certain clusters of size > 2.

In this case, we would like to find not only the closest neighbor of distance 1 but also the closest neighbor of distances n where n > 1. To do this, we must consider neighbors that are connected by walks of length n.

It is important to make a distinction between naively choosing the nth highest rank in our rudimentary out degree weight list and this procedure. The list represents a direct connection from the start node to a target node and not a walk of length n.

To compute walks of length n, we use the following well-known theorem.

Theorem: Raising an adjacency matrix A of a simple graph G to the n^{th} power gives the number of n-length walks between two vertices v_i, v_j of G in the resulting matrix. [3]

This gives,

Corollary: Raising an adjacency matrix of weights W of a simple graph G to the n^{th} power gives the relative weighted attachment between two vertices v_i, v_j connected by n-length walks of G in the resulting matrix.

Hence, we are to simply raise W to the n^{th} power and then observe the row of the starting element we are concerned with. The element in the row with the greatest weight is the most common element that is connected by walks of length n in the matrix, and is therefore the ideal nth element in the cluster. (assuming starting element (0^{th}) is fixed)

Such clusters could be potentially useful for determining trends in the data and in the nature of the graph itself. In the case of spaced repetition for learning, it could signify trends with a particular event/element being fixed – lending to analysis into the qualitative nature of the relationship between elements in the cluster.

4. Generating Test Data

Due to the nature and applicability of the problem, procuring test data might be difficult or time-consuming.

We propose a greedy heuristic as follows:

- Create an empty list of k elements
- Calculate element with next highest benefit: interval ratio
- Place element at first empty position and as frequently as possible, satisfying the condition that the element is placed in the next empty space which

also satisfies interval distance restriction.

- Repeat steps two and three till list filled

This greedy approach should provide reasonable test data (in polynomial time), for analytical purposes at the very least.

A more accurate, albeit exponential time, algorithm could employ a dynamic programming-based approach.

5. Conclusion

In conclusion, we note that our approach is quite effective in identifying clustering trends of particular elements in the list. Moreover, in computing benefit, with each element being a function of the previous element (using the edge set), it is seen that the top weight or attachment will not always satisfy interval restriction and second or further preferences will have to be chosen.

When applied to real data, a measure for tallying the number of times the top weight is not chosen could be implemented. If this number is relatively (relative to k) large, for a particular element, the real-life system can be modified to decrease interval of a particular element, to potentially increase benefit.

Valid extensions include modifications to accommodate for differing time taken per element, a dynamic tweaking procedure for the weights, as well as dependencies between elements, altering clustering properties.

References

[1] Baranov, I. V. (2018). Improving Listening Skills in Language Learning with Spaced Repetition Technique. *European Research*, 40

[2] Ausubel, D. P., and Youssef, M. (1965). The Effect of Spaced Repetition on Meaningful Retention. *The Journal of General Psychology*, *73*(1), 147–150.

[3] Connotations of k^{th} Graph Power and path lengths (n.d.). Retrieved from http://mathworld.wolfram.com/GraphPower.html.