**Overview**

What make S-Din work are the algorithms behind it. Our model group worked very hard to develop the recommend and simulation algorithms to make S-Din possible in functioning as expect. We combined the search and recommend algorithm together so the software can work faster. Simulation algorithm are developed to cover situations which is any circuits designed by users.

**Recommendation System based on Collaborative Filtering strategy**

***Section1. Introduction***

The users of our software are all innovative researchers on Synthetic Biology, who are interested in many different biological fields. The purpose of our system is to recommend most related genetic parts to the users based on the research interest the users offer to our system.

The information that we use to make recommendation is a database built by NLP and Random Walk which contains scores between each key word and each genetic part. Formally, it is a matrix , where is the number of key words and is the number of parts . The element at row , column is the score between the key word and genetic part , which reflects the connection between them(higher score means higher connection).

The overall strategy of our system is Collaborative Filtering , i.e. we first search the similar key words in our database of the unknown word offered by users and then recommend the genetic parts which are highly related to key words that we found to the users.

A natural question is how to find the most similar key words in our dataset of a given unknown word in an accurate and efficient way? Here is our general solution : To quantify the semantic similarities between words accurately ,we use technique in Deep Learning to convert words into numerical vectors and the cosine similarities between each vectors can represents the semantic similarities between words. To search the similar key words efficiently, we use the KD Tree Algorithm, a fast algorithm based on binary tree, to implement the K Nearest Neighbors strategy.

***Section2. Models used in the system***

* Word2Vec Algorithm

Word2vec is an algorithm that produces word embedding , i.e. it converts a corpus of text into a high dimensional real vector space(in our case , the dimension is 400) and each word in the corpus is assigned to a vector in the vector space. If two words are similar semantically , then their will be close under cosine distance measure.

In our Recommendation System, we use gensim , an open source Python module focused on Natural Language Processing , to train our word2vec model and the corpus we use to feed the model is wikimedia, which can be downloaded from: <https://dumps.wikimedia.org/> .

The reason why we use Word2vec is that it can distinguish the semantic meanings of words accurately by Deep Learning technique , which outperforms the traditional semantic analysis methods greatly.

* KD Tree Algorithm

The KD Tree algorithm is rather efficient when searching for the most similar items with averaged time complexity ,thus the users can get recommendation instantly after they enter their interested new word to our system.

The KD Tree Algorithm consists of the following two parts :

1) Construct KD Tree based on the normalized word vectors of the key words in the database which we previously mentioned. (Offline)

2) When the user enter a new word , we search the similar key words along the KD Tree that we constructed in step 1) . (Online)

Note that KD Tree Algorithm is based on Euclidean distance measure , while in our case , we need to calculate the cosine distance of the word vectors. To solve this problem ,we normalize all the word vectors of key words before constructing KD Tree and normalize the word vector of new words offered by users before letting it travel along the constructed KD Tree. The reason behind the method is that by Law of Cosines :

,

while since are normalized , we have :

, ,

Now we see that the calculation of cosine distance between normalized vectors can be replaced by the calculation of Euclidean distance since they can determine each other in this normalization context.

In our Recommendation System , we use the KD Tree implementation in scikit-learn , a simple and efficient open source machine learning module in Python. For more details of KD Tree algorithm , please visit : <https://en.wikipedia.org/wiki/K-d_tree>

* Random Walk with Restart Algorithm

Key word n

……

……

Key word3

Key word2

Key word1

Random Walk with Restart(RWR) is an algorithm adapted from the PageRank algorithm and it focuses on characterizing the affiliation between each item. We treat the relation between key words and genetic parts as an undirected graph where nodes represent key words or parts and edges represent connection between words and parts. Imagine there is a walker travelling on the graph mentioned above and each time he faces two choices: 1) Randomly travelling along an edge connected to the current node. 2) Teleport to node K. After a long time of random travelling , the frequency the walker reaches each node represents the affiliation between each node and node K , which we use to characterize the relation between key words and parts. For more detailed mathematical formulation of PageRank , please visit : <https://en.wikipedia.org/wiki/PageRank> .

***Section3. Specific steps to construct the Recommendation System***

* Step 1: Calculating affiliation between key words and parts

We used random walk with restart to calculate the affiliation between key words and all parts in the dataset. The calculation is done by updating the probability vector iteratively ,once the probability vector is converged, it can be viewed as the affiliation between various parts and key words. Therefore, the higher the probability is, the closer the key word and part are.

Here is the Algorithm that we use in our project:

1. Define transition matrix

let be the transition matrix of the graph with size of where represents the number of key words and represents the number of parts. The entry in row and column of is if node of has degree , and one of the adjacent nodes is . Otherwise, this entry is 0. Note that the first columns(rows) represent key words and the other columns(rows) represent the parts.

1. Prespecify

is the probability that the walker continues to follow the edges connected to the current node, thus is the probability the walker will teleport to the initial node .

3) Iterate over

for key word in ：

{

a. Define and Initialize :

Let be the column vector that has 1 in the row for node and 0’s elsewhere, i.e.

And is the column vector such that each item reflects the probability the walker stays at each node of the graph and the initial is set to be .

b. Update by:

update until the terminal condition is satisfied, i.e.

c. Modify :

Since the first n components of are the affiliation between key word and the other key words which we are not interested , therefore we only keep the last m components of , which reflects the affiliation between key word and all the parts.

}

* Step 2: Build KD Tree of Key words

Once the user enter a new word to our system, we need to search for similar key words efficiently , thus we build a KD Tree of key words to implement this function.

Here is our Algorithm :

1) Train word vector model

We feed our word vector model with Wikimedia corpus . After the training process is done, we have a model that can convert words into numerical vectors , i.e.

where denotes the corpus space.

2) Convert key words into normalized vectors

Let denotes the set of word vectors.

for key word in

{

.append( )

}

3) Build KD Tree

We use the standard construction procedure to build KD Tree of set . Details will be ignored here. For the people who are interested , we offered a link to wikipedia in section2.

* Step 3: Collaborative Filtering

This is the final step to construct our Recommendation System , overall we use collaborative filtering strategy to make prediction.

Here is our Algorithm:

1) Transform users' input

We get users' input word , we then convert it into normalized word vector , i.e.

2) Search K most similar key words (KNN)

Let travel along the KD Tree in a way like binary search , we can end up with (in our case , is set to be 5) most similar key words(KNN) to the current word . For more details about how the searching algorithm works , please visit the link that we offered in section2.

3) Make Recommendation

In section3 -step1 , we have calculated the affiliation between key words and parts which shall be used to guide our final recommendation. Let denotes the connection between the part and the current word , we calculate by formula :

where denotes the most similar key words ,

denotes the connection between and the part and denotes the Euclidean distance between the normalized word vectors of and word .

After the computation of } , we use heap sort algorithm to get highest and recommend to the users the corresponding parts.

**Simulation for General Genetic Circuits**

***Section1. Introduction***

The users of S-Din are all active and innovative researchers in Synthetic Biology, who come up with many interesting and new ideas in a wide range of fields. Our software helps them achieve their creative goals by allowing them to combine various genetic parts together to form brand-new genetic circuits that may never exist before. To help the researchers have a deeper understanding of how the genetic system that they created works, we construct an ODE system to simulate the dynamic behaviors of the genetic system mathematically. The main challenge of simulation is the uncertainty of genetic circuits since the users can construct them arbitrarily. Therefore, we build a rather general model capable of simulating various genetic circuits.

***Section2. Model***

The interaction among the substances in genetic circuits can be characterized by a Relation Matrix , where is number of substances. The entry in row and column has 3 possible values for 3 possible relations, i.e.

Therefore , the column records all the impacts the system exerts to the substances .

Let denotes the concentration of substance at time , then we use the following ODE to characterize for :

where represents the degradation rate and :

***Section3. Implementation***

We use Scipy, an efficient open source numerical module in Python, to give the numerical solutions to the ODE system. Specifically, we use the odeint function in Scipy. For more information about odeint function, please visit: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html>