Overview

Recommendation system based on KD tree and random walk

General simulator for various parts

Recommendation system based on KD tree and random walk

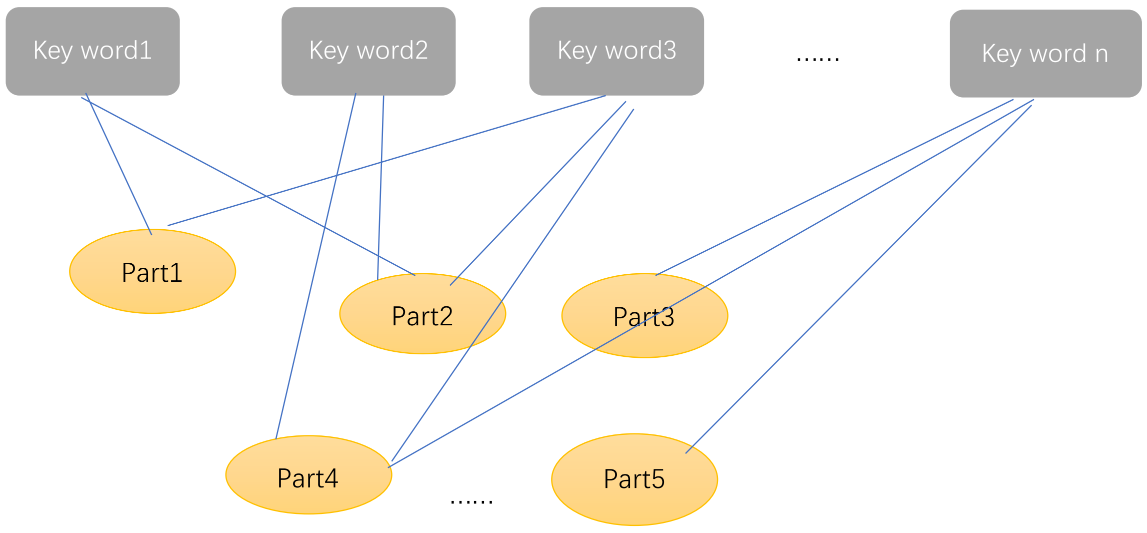
Basic Knowledge

- KD tree Algorithm

Each node is the k-dimensional point of binary trees. All non-leaf nodes can be considered a hyperplane space partitioned into two half-spaces (Half-space). Representatives of the left subtree of the node are on behalf on the left side of the hyperplane, representative of the right subtree of the node are on behalf of the right side of the hyperplane. Select the hyper-plane as follows: each node is associated with the dimension that is perpendicular to the hyperplane in the k-dimension. Therefore, if you choose to divide by x axis, all the x values less than the specified value will appear in the left subtree of nodes, all the x values greater than the specified value will appear in the right subtree. This hyperplane can be determined by the x value, the normal of which is the unit vector for x axis.

This algorithm can be used to find k points closet to a specified point. The result is like the KNN algorithm, but the complexity is much lower than the KNN algorithm.

- Random walk with restart on graphs



RWR is an algorithm originally proposed for image segmentation. It iteratively explores the global structure of the network to estimate the proximity (affinity score) between two nodes.

Starting at a node, the walker faces two choices at each step: either moving to a randomly chosen neighbor, or jumping back to the starting node. This algorithm only contains one fixed parameter r called 'the restarting probability' (with 1 − r for the probability of moving to a neighbor). After iteratively reaching stability, the steady probability vector contains the affinity score of all nodes in the network to the starting node. This steady probability vector can be viewed as the 'influential impact' over the network imposed by the starting node.

Algorithms

Algorithm 1: Algorithm for Calculating Similarity among key words and parts

We used random walk with restart method to calculate the similarity between a certain key word and all parts in the dataset. The idea is similar to the Pagerank algorithm invented by Google. Once the probability vector is converged, it can be viewed as the similarity of various parts and the certain key word. Therefore, the higher the probability is, the closer the key word and part are.

for key word n in 1：N {

1) Define transition matrix M

   let M be the transition matrix of the graph G. That is, the entry in row i and column j of M is 1/k if node j of G has degree k, and one of the adjacent nodes is i. Otherwise, this entry is 0. Note that the first n columns represent key words and the other m columns represent the parts.

2) Initialize β

   β is the probability that the walker continues at random, so 1 − β is the probability the walker will teleport to the initial node n .

3) Define and Initialize v

   Let e_N be the column vector that has 1 in the row for node n and 0’ s elsewhere. v is a column vector reflecting the probability the walker is at each of the nodes at a particular round, and v′′ is the probability the walker is at each of the nodes at the next round.

4) Update v

$$\mathbf{v}' = \beta M\mathbf{v} + (1-\beta)\mathbf{e}_N$$

5) Update v until the change of it is smaller than some prespecified value like 10-5.

6) Delete the first n rows of n, which are representatives of key words and get the vector u_N, rank the elements of u_N in descending order and save u_N.

}

After the iteration, we could get a matrix P consisting of N columns, which represent N key words. Number of rows are equal to number of parts in the dataset.

Algorithm 2: Algorithm for Parts Recommendation

When the user inputs a key word k in the search engine, we first judge whether it exists in our key words lexicon.

If does, we will extract the related column vector in matrix P and recommend the first M row names of the vector, which are the most M related parts to this key word.

If not, we first transfer the new key word into word vector a by xxx. Then we calculated the Euclidean distance between the new key word and words in lexicon and select the most closet W words, distances of which are denoted by $d_1, \cdots, d_W$. Extract the related W column vectors in matrix P got in Algorithm 1, denoted by $a_1, \cdots, a_W$. After that, we give every part a score not only according to the distance between the new key word and the most closet W words, but also based on the similarities between the most closet W words and various parts. To be specific, we divide each elements in a_i by d_i ($i = 1, \cdots W$) for the closer the words are, the higher probability they are alike and get W new vectors $b_1, \cdots, b_W$. Arrange the W vectors in column to form a matrix B, column names of which are W key words and row names of which are all parts in the dataset. Then we add each row of matrix B and get scores for every part in the dataset. Rank the highest M part scores and recommend those M parts to the key word k.

```
1)  Input a key word k
2)  If k exists in Key Words Lexicon :
        extract the related column vector in matrix P and recommend the first M row names
        of the vector to the key word k.
    Else:
3)  Transfer the new key word into word vector
4)  Calculated the Euclidean distance between the new key word and words in lexicon and
    select the most closet W words, distances of which are denoted by d1, ···, dW
5)  Extract the related W column vectors in matrix P got in Algorithm 1, denoted by $a_1, \cdots, a_W$.
6)  Divide each elements in $a_i$ by $d_i$ ($i = 1, \cdots W$) and get W new vectors $b_1, \cdots, b_W$.
7)  Arrange the W vectors in column to form a matrix B and add each row of matrix B to get
    T (number of parts in the datasets) part scores.
8)  Rank the highest M part scores and recommend those M parts to the key word k.
```

General simulator for various parts

General Rules

Most teams modeled the process based on ODEs, so we want to summarize a common formula to automatically simulate the performance of the system designed by users. In other words, we want to make the models to be applicable to many biological parts and systems.

- Hill Function

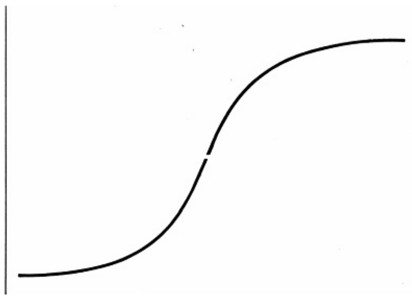
$$\theta = \frac{[\mathrm{L}]^n}{Kd + [\mathrm{L}]^n} = \frac{[\mathrm{L}]^n}{(\mathrm{K}_A)^n + [\mathrm{L}]^n} = \frac{1}{\left(\frac{K_A}{[\mathrm{L}]}\right)^n + 1}$$

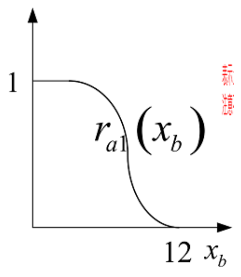
θ	Fraction of the ligand-biding sites on the receptor protein occupied by the ligand
[L]	Free(unbound) ligand concentration
K_d	Apparent dissociation constant derived from the law of mass action
K_A	Ligand concentration producing half occupation
n	Hill coefficient

This equation describes “the fraction of the macromolecule saturated by ligand as a function of the ligand concentration”, and is used in determining the degree of cooperativeness of the ligand binding to the enzyme or receptor.

- Sigmoid Function Acted as Regulators

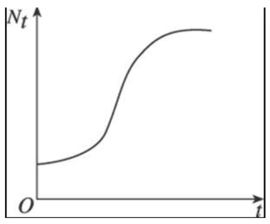
As the concentration of proteins can influence promote or inhibit the performance of 启动子 and different concentration may lead to different results, we applied smooth sigmoid functions here to reflect the influence. We plot the performance vs the concentration of protein in the following two plots, with the left one representing the promotion sigmoid function and the right one corresponding to the inhibition sigmoid function.





- Cells Division

Most models are based on parts in a single cell, whereas in the real experiments, cells will divide into new cells during the observations. In order to get more exact result, we utilized logistic function to describe the number of cells in an system and got our final concentration results considering cell divisions.



Assumptions

- Leakage does not exist.
- The growth of cells is neglected.
- 某些 RNA 对于启动子的影响不考虑，在这里只考虑通路中表达出的蛋白质对其他通路的启动子的影响。

Basic Form of ODEs

- 任意一个通路的启动子只受一个蛋白质调控
  首先，根据界面中已有的通路个数（启动子个数）可以判断出该系统中应该存在的蛋白质个数 n（同样也为 mRNA 个数和对应的基因的个数 n）。
  - 定义

$$I_{ij} = \begin{cases} 1 & \text{蛋白质}p_i\text{对基因}g_j\text{有作用}, \\ & \text{蛋白质}p_i\text{对基因}g_j\text{有作用}. \end{cases}$$

$$J_{ij} = \begin{cases} 1 & \text{蛋白质}p_i\text{对基因}g_j\text{有促进作用}, \\ & \text{蛋白质}p_i\text{对基因}g_j\text{有抑制作用}. \end{cases}$$

$$\tilde{I}_{ij} = \begin{cases} 1 & \text{蛋白质}p_i\text{mRNA}m_j\text{有作用}, \\ & \text{蛋白质}p_i\text{对mRNA}m_j\text{有作用}. \end{cases}$$

$$\tilde{J}_{ij} = \begin{cases} 1 & \text{蛋白质}p_i\text{对mRNA}m_j\text{有促进作用}, \\ & \text{蛋白质}p_i\text{对mRNA}m_j\text{有抑制作用}. \end{cases}$$

  i = 1,⋯n; j = 1,⋯ n。并且对于某个固定的 j，有且只有唯一的 i 使得 $I_{ij} = 1, \tilde{I}_{ij} = 1$ 。

  - ODE 方程组

$$\frac{d[m_j]}{dt} = -\gamma_{m_j} \cdot [m_j] + \alpha_j[g_j] \cdot (1 - \sum_{i=1}^{n} I_{ij}) + \sum_{i=1}^{n} I_{ij} \cdot \left[J_{ij} \cdot \frac{1+\alpha_j}{1+(\frac{k_{mj}}{[p_i]})^n} + (1-J_{ij}) \cdot \frac{\alpha_j}{1+(\frac{[p_i]}{[k_{mj}]})^n} \right] \tag{mRNA}$$

$$\frac{d[p_j]}{dt} = -\gamma_{p_j} \cdot [p_j] + \beta_j[m_j] \cdot (1 - \sum_{i=1}^{n} \tilde{I}_{ij}) + \sum_{i=1}^{n} \tilde{I}_{ij} \cdot \left[\tilde{J}_{ij} \cdot \frac{1+\beta_j}{1+(\frac{k_{pj}}{[p_i]})^n} + (1-\tilde{J}_{ij}) \cdot \frac{\beta_j}{1+(\frac{[p_i]}{[k_{pj}]})^n} \right], \tag{protein}$$

$j = 1, \ldots, n$

总共有 2n 个未知数,2n 个方程.

- 存在一个通路的启动子受到大于等于两个蛋白质的调控
  首先，根据界面中已有的通路个数（启动子个数）可以判断出该系统中应该存在的蛋白质个数 n（同样也为 mRNA 个数和对应的基因的个数 n）。
  - 定义

$$I_{ij} = \begin{cases} 1 & \text{蛋白质}p_i\text{对基因}g_j\text{有作用,} \\ & \text{蛋白质}p_i\text{对基因}g_j\text{有作用.} \end{cases}$$

$$J_{ij} = \begin{cases} 1 & \text{蛋白质}p_i\text{对基因}g_j\text{有促进作用,} \\ & \text{蛋白质}p_i\text{对基因}g_j\text{有抑制作用.} \end{cases}$$

$$\tilde{I}_{ij} = \begin{cases} 1 & \text{蛋白质}p_i\text{mRNA}m_j\text{有作用,} \\ & \text{蛋白质}p_i\text{对mRNA}m_j\text{有作用.} \end{cases}$$

$$\tilde{J}_{ij} = \begin{cases} 1 & \text{蛋白质}p_i\text{对mRNA}m_j\text{有促进作用,} \\ & \text{蛋白质}p_i\text{对mRNA}m_j\text{有抑制作用.} \end{cases}$$

i = 1,⋯n; j = 1,⋯ n。并且对于某个固定的 j，有且只有唯一的 i 使得 $I_{ij} = 1, \tilde{I}_{ij} = 1$ 。

$$f_{ij}(x) = f_1(x)J_{ij} + f_2(x)(1 - J_{ij})$$
$$\tilde{f}_{ij}(x) = f_1(x)\tilde{J}_{ij} + f_2(x)(1 - \tilde{J}_{ij})$$

$f_1(x)$ is promotion sigmoid function,$f_2(x)$ is inhibition sigmoid function
$f_{ij}(x)$ is the performance of protein x regulating the promoter,$\tilde{f}_{ij}(x)$ is the performance of protein x regulating the mRNA

此时这个函数就表示了某个蛋白对于启动子的抑制或者促进作用的大小,随着该蛋白浓度的变化而变化,是一个 sigmoid function.

  - ODE 方程组

$$\frac{d[m_j]}{dt} = -\gamma_{m_j} \cdot [m_j] + \alpha_j \cdot \Pi_{i=1}^{n} \left[I_{ij} \cdot f_{ij}([p_i]) + (1 - I_{ij}) \cdot [g_j] \right] \qquad \text{(mRNA)}$$

一般我们取$[g_j] = 1$，从而变成下面的方程

$$\frac{d[m_j]}{dt} = -\gamma_{m_j} \cdot [m_j] + \alpha_j \cdot \Pi_{i=1}^{n} \left[I_{ij} \cdot f_{ij}([p_i]) + (1 - I_{ij}) \right] \qquad \text{(mRNA)}$$

$$\frac{d[p_j]}{dt} = -\gamma_{p_j} \cdot [p_j] + \beta_j \cdot \Pi_{i=1}^{n} \left[\tilde{I}_{ij} \cdot tildef_{ij}([p_i]) + (1 - tildeI_{ij}) \cdot [m_j] \right] \qquad \text{(protein)}$$

$$j = 1, \ldots, n$$

总共有 2n 个未知数,2n 个方程.

Final Concentration Results

由于我们考虑细胞分裂所带来的影响，最后系统的物质浓度= 细胞分裂函数 S(t) × 单个细胞物质浓度随时间的变化函数。