Question Recommender

You are a visiting Professor in the first and only University of Mars, which admits exclusively Martians (off-planet tuition would be too expensive for Earthlings anyway). You have joined the Department of Astronomy and are in charge of teaching Astrometrics. Unfortunately the university is quite overcrowded and each semester there are close to 13,000 martian students that take Astrometrics.

Given the size of your class, and a disturbing lack of teaching assistants, you are forced to limit the number of questions given to each student on the mid-term exam to only five multiple choice questions. To further complicate matters the department head has mandated that all test questions be pulled from a bank of approximately 400 “approved” test questions. To increase the robustness and ease of creation of your midterm exam you decide to randomly and uniformly choose 5 questions from the question bank for each student. This means that each student’s exam consists of disjoint, intersecting, or identical sets of 5 questions.

To assist you with running such a large class, the university has provided you with last year’s student performance data (where the professor created exams using a process similar to your own). Additionally, the department head has admitted to you that she thinks that not all the questions in the “approved” question bank are actually useful for evaluating each student’s relative understanding of Astrometrics. She also warns you that while it is OK not to use all the questions from the bank, you must, at minimum, use 50% of them in order to achieve sufficient coverage of the curriculum.

With the mid-term exam only a few days away, which questions should you **exclude from** the exam generation algorithm such that the exam results will provide the most meaningful ranking of the students in the class? Why? Please explain both your reasoning and methodology; also, please include any code you used to generate your results. Finally, please include an estimate of the time you spent solving this problem. Note: when solving this problem you may not use Matlab or Octave and we discourage the use of R.

The data set you are given by the department has one record per line, where each record consists of:

1. A unique identifier for a question
2. A unique identifier for a student
3. The correctness of a student's response. A correct answer is marked as a 1; an incorrect response is marked as a 0

Solution: Collaborative Filtering

May 28, 2013

Barnett Chiu

1. Introduction

This problem can be addressed via a three-stage pipeline: i) collaborative filtering ii) clustering (or alternatively, sampling) and iii) question assignment optimization. To predict student response to unanswered questions, two types of collaborative filtering algorithms are implemented in this demo with one based on weighted averaging with a question-specific similarity matrix and the other based on regression analysis.

For the purpose of developing question assignment strategy, one can first start with defining an optimization objective, hypothesizing an initial solution (as a vector for instance), followed by incrementally adjusting the solution towards the desired objective (or as close as possible). This is a good use case for applying genetic algorithm, which progressively “mutates” current solutions towards creating elite solutions with higher fitness degree; i.e. the higher the fitness, the closer to the optimal solution. However, since the student number is very high (and in general without an upper limit), the solution space can potentially be intractable. To reduce the solution space, one way is to apply clustering techniques and identify groups of similar questions and similar users. In this manner, the question assignment strategy can be derived on the level of clusters (rather than the individual question or student).

An important observation is that the number of questions is relatively small compared to the number of students, which will continue to increase semester after semesters. From the experience of experiments, clustering questions is feasible while clustering students can be computationally expensive and thus should be avoided if possible. Another way of reducing solution space is to use sampling methods, particularly for the case of students. That is, in each trial, only a small subset of student population is used to evaluate an optimal question assignment strategy. To reduce bias, genetic algorithm is run multiple times, through which a weighted score is accumulated for each question across different episodes of training to determine the final ranking of questions. For the purpose of this demo, I will focus on the regression-based collaborative filtering algorithm while using its result as the basis for the subsequent (question-) cluster analysis and deriving optimal question assignment policy with genetic algorithm. Both kmeans++ [1] and spectral clustering [2] are implemented for the purpose of experiment and illustrating clustering results for identifying similar questions. For visualizing high-dimensional data points obtained from regression-based collaborative filtering, multidimensional scaling [4] is used in this demo. This helps to show the relationship between questions and users in lower dimensional space (e.g. 2D) that reflects their distances in high dimensional space. For future reference, clustering and sampling part of this three-stage pipeline can continue to develop for better performance of this recommender system.

2. Collaborative Filtering

Here, we wish to determine the set of questions that can best distinguish the student performance and therefore, in the ideal case, we will need to know how they respond to the entire question set and select those questions that lead to higher variability in scores. However, this is often not feasible because in truth only a very small subset of questions will be assigned to each student during the exam. Intuitively, one can predict how a student will respond to an unanswered question through how they performed on other question set in the past. In other words, the prediction is based on similarity of questions. If a new question is similar to those other questions that were being answered very well by a student in the past, then one can conclude with high confidence that he or she will also respond well to this new question. The similarity between two questions can easily be determined if the content of the questions were known, which is not the case in this setting. However, one could look into how other students respond to the two question of interest, and if they share similar feedback from students (i.e. score obtained for the question), then these questions very likely cover similar topics in class. This is similar to the case where two commercial products with similar spec and features often receive similar ratings from customers. In this manner, one can construct a similarity matrix for the questions with each entry representing similarity score. In this manner, one can predict the student response for a new question by weighted average of other questions (or selected subset with sufficient degree of similarity) that the student had answered in the test. The higher the similarity to a previously answered question, the higher its contribution to the final prediction. One may also consider constructing user-based similarity matrix and by comparing common set of questions answered by all pairs of students. However, since the number of students is much larger, there will be much less overlap in questions between any two students. As a result, item-based (or question-based) collaborative filtering will perform better in this scenario.

On the other hand, had we had access to the content of questions (i.e. topics covered), then we could potentially describe the question with a set of question-specific “features,” each of which represents the weighting of a topic covered in Astrometrics such as zero-point energy device [3]. Subsequently, one could build a regression model for each student using his/er responses of other questions as training set to find out how much each feature contributes to the score (e.g. feature weights in linear regression). If we know how much a student favors certain topics (while not others), then the prediction can be made to an arbitrary question described in terms of the topic-specific features[1]. This approach is referred to as content-based recommender system. Specifically, suppose that *n* features $\{x_i\}_{i=1,\dots,n}$ are used to represent a question and for each question, we know the feature values (assuming for now that the content is known). Specifically, with linear regression, one can represent the predictive function of

[1] Certainly, there are other features that can potentially help as well to predict student performance such as level of difficulty for a particular topic, the time in the semester when a topic was introduced, whether there was an exercise associated with a topic in question. NLP features may also be relevant including question position, question type, question focus, sentence length, vocabulary-specific features or other linguistically useful features, among many others.

student performance by $f(x) = \theta_i x^i = \theta^T x$. The total cost for a student performance can be represented by

$$\frac{1}{2} \sum_{i,j:R(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 \tag{1}$$

where $\theta^{(j)}$ denotes the feature weight vector for *jth* user and $x^{(i)}$ denotes the feature vector for the *ith* question. With the training set, one can then estimate optimal feature weights $\{\theta_k\}_{k=1,\ldots,n}$ that minimizes (1) for each student. By similar token, suppose instead that the student preference for each question-specific feature is known beforehand (i.e. $\{\theta_k\}_{k=1,\ldots,n}$ is known), then by symmetry, $\{x_i\}_{i=1,\ldots,n}$ can be estimated via the same optimization process. For this problem domain, because neither the question content nor the student preference is known, both $\{x_i\}_{i=1,\ldots,n}$ and $\{\theta_k\}_{k=1,\ldots,n}$ are considered as free parameters to estimate. By introducing regularization terms for both parameter sets to avoid overfitting, the final cost function can be represented by:

$$\frac{1}{2} \sum_{i,j:R(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{nU} \left\| \theta^{(j)} \right\|^2 + \frac{\lambda}{2} \sum_{i=1}^{nQ} \left\| x^{(i)} \right\|^2 \tag{2}$$

where $R(i,j) = 1$ if and only if the jth user was assigned to ith question in the test; 0 otherwise. *nU* and *nQ* are the total number of users/students and questions respectively and λ is the regularization constant. Next, we then apply an appropriate optimization method to determine the parameters that minimize (2). In this demo, I included both the conjugate gradient method available from scipy and the Polack-Ribiere flavor of conjugate gradients adapted from Carl E. Rasmussen's implementation in Matlab (with minor changes). See the file *minimizer.py* and *cofi.py* for more details. One could also use logistic regression (as a binary classifier) for making predictions since the correctness only assumes two possible values in this case. However, this is unlikely to make a significant difference in system performance in this case since there are no "outliers" in correctness scores (i.e. invalid correctness that goes outside of allowable range 0 and 1).

2.1 Implementation

System Requirements:

(a) The code was implemented in Python 2.7 on Mac OS X 10.7 platform.

(b) Required packages: numpy, scipy, matplotlib, PIL (for visualizing similar questions and users). PIL can be tricky to install on Mac OS (and so are numpy and scipy perhaps). As a reference, I found this link useful:

http://jetfar.com/libjpeg-and-python-imaging-pil-on-snow-leopard/

In particular, ***cofi.py*** is the main entry of the code. Most of the modules include test and “main” functions (recall C++) that demonstrate the main usage of the modules. The file set {*cluster1.py,* ***cluster2.py****, cluster_util.py, similarity.py, multidim_scaling.py*} deals with clustering algorithms and data visualization (for the purpose of identifying similar questions). The file set {**evoluation.py**, cofi.py} contain the code for running genetic algorithm (fitness function is defined in cofi.py whereas the main genetic optimization algorithm is implemented in evolution.py. Note that the files may seem to be a bit messy but the idea here is to group key functions relevant to this question in one file i.e. cofi.py for the purpose of this demo. Moreover, the file set {cofi.py, cofi_ref.py} implements both types of collaborative filtering algorithms (i.e. weighted averaging with similarity matrix and regression-based scheme). Lastly, {minimizer.py} implements Polack-Ribiere flavour of conjugate gradients. The rest of the files mostly contain helper functions.

2.1.1 Data Preprocessing

The file *DataStore.py* is used to convert the input file (*astudentData.csv*) into a set of useful matrics **(X, Theta, Y, R)** and also detect duplicate or inconsistent data (which do exist in this data set).

X is a *nQ-by-nF* matrix representing the question set, where *nQ* is the number of questions and *nF* is the number of features. In this demo, nF is chosen to be 10. Since the ideal parameters for question set is unknown at first, X is initialized to a zero matrix of size nQ-by-nF. Similarly, Theta is a nU-by-nF matrix representing the users, where *nU* is the number of users and *nF* again is the number of features. Note that the number of features in both *X* and *Theta* needs to agree. Each question and user are represented by a row vector of dimensionality *nF*.

Y is a *nQ-by-nF* matrix with entries Y[i,j] corresponding to the correctness score for the case of ith question answered by jth student. R on the other hand is also a nQ-by-nF matrix with R[i,j] indicating if ith question was assigned to jth student in the test. R[i,j] = 1 iff student/user j answered the question; 0 otherwise. These matrices are useful in the vectorized implementation of cost function and its gradient function. Vectorized expressions often gain significant speedup in converging to the optimal or nearly optimal parameters and increase readability of the code.

On a relevant note, an internal ID map is also maintained by DataStore. Question IDs and user IDs are mapped to 0-based indices in accordance with the ascending order. This is done to facilitate matrix indexing. For instance, X[i,:] corresponds to the question (feature vector) with ith largest ID.

Major files include:

(a) DataStore.py
- loadToTable(): Load data set from *astudentData.csv*, resolve duplicates and data inconsistency (see comments of the code for more info)

2.1.2 Collaborative Filtering Based on Weighted Average

This method serves as a baseline algorithm for the purpose of comparing with regression-based collaborative filtering. The major disadvantage for this method is that questions (and users) do not assume "concrete representations" as feature vectors. Consequently, more detailed cluster analysis and data visualization is more difficult.

(a) *cofi_ref.py*: The algorithm first compute student affinity matrix and subsequently predict student response to unanswered questions via weighted average over those exam questions (they had answered) depending on the similarity between the unanswered question and exam questions.

- *evalQuestionAffinity()* computes similarity matrix for questions.
- predict() makes prediction on student response to unanswered questions
- main() demonstrates the use of major functions in this file including the above two.

Nonetheless, clustering analysis can still be performed here to identify groups of similar questions because we have similarity matrix available. Clustering users can be very slow and is not recommended.

Although main() also contains a test for creating question clusters via spectral clustering algorithm using the similarity matrix as input. However, clusters cannot be directly visualized in terms of groups of related data points because there are no features associated with each question in this scheme. In general, if data points go beyond 3 dimensional, only their projection in lower dimensional space can be visualized.

2.1.3 Regression-based collaborative filtering

Please consult with the following files:

(a) cofi.py: The primary program where regression and the optimization are implemented. In particular, *cost()* and *gradCost()* represent the cost function; i.e. $J(x^{(1)}, x^{(2)}, ...x^{(nQ)}, \theta^{(1)}, ..., \theta^{(nU)})$ and its gradient with respect to all parameters (i.e. $\frac{\partial J}{\partial x_k^{(i)}}$ and $\frac{\partial J}{\partial \theta_k^{(j)}}$)

Major functions in cofi.py:

- learn(): Load the dataset, initialize (X, Theta, Y, R) tuple (see section 2.1.1), and compute optimal parameter/feature values for (X, Theta) given (Y, R) using (2) as the cost function (with 2-norm regularization). gradCost() computes the gradient of cost function. λ is currently set to 10.

  Note that conjugate gradient optimization scipy.optimization.fmin_cg() or the version in **minimizer.py** both take $(J, \nabla J)$ as input, perform “smart” line search to arrive at the (nearly-) optimal parameter values. If the optimization is successful, each coordinate in ∇J should end up being very small, meaning that not much change can be made to further reduce the cost function J significantly.

- predict(): Predict correctness score with any combinations of question and user with learned parameter matrices: (X, Theta).

- assignment(): Perform genetic algorithm with cluster sampling to find the ranking of questions with the optimization objective: assignment that leads to higher variations in correctness scores.

(b) evoluation.py: implements genetic algorithm (see section 2.1.4)

2.1.4. Clustering, Sampling and Multidimensional scaling

Recall from Section 1 that genetic algorithm is used to derive the ranking of questions that best differentiate student’s understanding of the material. However, since the number of the students is very large (and potentially unbounded), it is often not feasible (or too costly) to derive question assignment strategy strictly based on the feedback from the entire student population. In this demo, we will use cluster sampling method over question set. By contrast, sampling on users is simply accomplished via random sampling (since clustering users is often too costly). Why sampling? For the purpose of applying genetic algorithm (to be discussed shortly) in a general setting, we would want to represent the solution in a simple yet flexible way. Consider the solution represented in the form of a vector $\left\langle q_1^{(1)},...,q_5^{(1)},q_1^{(2)},...,q_5^{(2)},...,q_{nU}^{(2)},...,q_{nU}^{(2)} \right\rangle$, where the superscripts indicate the user IDs and subscripts indicate the question IDs. Subscripts for each user goes from 1 to 5 because we are interested in selecting 5 questions for each user. Using this scheme, the dimension of solution vector is given by *nU * 5*. That is, with large number of students, say 10,000, the solution vector can potentially be very large (i.e. 50,000 dimensional vector). In addition, the range of each coordinate can also be relatively large depending on the number of questions. Let nQ=m, nU=n. This means that we are trying to identify the distribution of optimal solutions from a space of size m^n. Sampling and clustering come into the picture to achieve a significant reduction of the search space without influencing the optimality of the solution if performed multiple times. In particular, suppose one decides to cluster question set into 20 groups (the ideal number of clusters is beyond the scope of this demo). Additionally, suppose that in each run, we sample 30 users to find the best question assignment strategy. The size of the search space is thus reduced to 20^{30}, which is still large but far better than 400^{10000} if one considers 400 individual questions in this problem set! Technically speaking, the solution space is less than 20^{30} because of the constraint that the 5 questions assigned to a student should not have duplicates (therefore $(C_5^{20})^{30}$ to be exact). One possible ***optimization objective*** used in this demo is to ***the find the question assignment such that it leads to higher variance in correctness scores***.

As a preview, in this demo, the genetic algorithm is configured to run 20 trials. Also please refer to *genetic_optimize()* and *fitness()* in **cofi.py** for more details. Specifically, the cluster sampling method goes as follows:

i) **Run clustering algorithm** with a selected similarity (or distance measure, which is essentially a dissimilarity measure). In this demo, the following clustering algorithms are implemented for experimental purposes: k-means++ [], frequency-balanced k-means++ [], and spectral clustering [] with 3 possible graph Laplacian (unnormalized, random walk, and symmetric). The goal for this step is primarily to derive question clusters (and data visualization) but not for users.

ii) Sample *nU'* users out of the entire population. *nU'* is currently set to 100 (perhaps too large? need further experiments)

At this point, we have both question clusters and user samples ready …

iii) With question clusters, **derive question assignment strategy** using the solution representation discussed earlier via genetic algorithm. In particular,

Loop until maximum iteration is reached:

(iii.1) Start with initial guess by random assignment of question cluster IDs (abbrev. as **q-cluster** later) for each sample user ID. For instance, with 20 clusters and 100 sample users, we have 5 * 100 = 500 dimensional solution vector with each coordinate having 20 possible variations. Produce N sample solution vectors where N is the population size (which is set to 100 currently, see assignment() in cofi.py)

(ii.2) **Convert the solution vector from clustered IDs to true IDs.** Specifically, for each user, sample the question (IDs) from the corresponding q-cluster without replacement **in proportion to the cluster size**. That is, the larger the cluster, the more samples is drawn. This gives the solution vector where each coordinate corresponds to a true question ID instead of clustered question IDs. This is reasonable considering that each q-cluster consists of similar questions.

(ii.3) With the tentative solution vectors obtained from ii.2 (i.e. “de-clustered” solution vectors), evaluate their costs using the given fitness function and rank them in the descending order (assuming that the larger the fitness, the better the solution)

(ii.4) Take the elite solutions (size set to 20 for now) from the population and (probabilistically) perform mutation or crossover operator to progressively change the solution (while preserving “good genes”) and subsequently add to the population (such that the population will gradually be filled with better and better solutions consistent with the optimization objective enforced by the fitness function). Note that mutation and crossover is selectively performed but not both depending on mutation probability (set to 0.2 for now).

iv) Given the ranked solutions, compute the accumulated, weight-adjusted ranked score for each question. The weighted ranked score is a function of two parameters: rank and frequency. In other words, we wish to preserve questions that frequently appear in higher ranked solution. In this demo, squared-exponential kernel (or radial kernel, Gaussian kernel in other contexts) is used to convert fitness score to weight in [0, 1]. Based on this scheme, compute the rank score for each question as follows:
Loop through all ranked solutions:
    Given a ranked solution, accumulate the weight reflecting the rank score of a question using a dictionary/map that maps question ID to its weighted accumulated score.

Major files include:

(a) cluster.py:

Include Kmean++ and frequency-balanced Kmeans++ (fsk-means) [5] that encourages balanced clusters with approximately equal sizes. However, for some reason, this does not work as intended in this problem set because very often a single gigantic cluster is formed. This is somewhat reasonable because if the data set is not sufficiently dissimilar to each other according to the given similarity measure, then forming more clusters (with unequal sizes) will be penalized. Nonetheless, this does not influence the cluster sampling process because number of samples is determined by the cluster size.

(b) cluster2.py:
Implement spectral clustering algorithm. See **qcluster_d12_radial_3.png** for an example of the experimental result. Because each question is represented by a higher-dimensional vector (10-D in this demo), it is not possible to draw question data points on 2D space. For the purpose of illustration, the relation along the first and 2nd feature coordinates are drawn, which may or may not be meaningful in terms of identifying similar clusters. One could use PCA or other dimensionality reduction techniques to find the principle axes with higher data variations to help identifying the lower-D representation of the data for the purpose of data visualization. This is why multidimensional scaling comes in handy.

(c) mutlidim_scaling.py: Used to visualize the question feature vectors that best explain the data based on collaborative filtering with 2-norm regularization. See ***questions2d_cofi_ref.jpg*** for an example that illustrates how questions are related to each other in 2D space that reflects their approximate distances in the original, potentially higher dimensional space (nF =10 in this demo).

Why do we visualize questions and users in terms of their vectorized representations? This makes it easier to identify their similarity and potentially help to explain why certain questions receive better responses from students and by symmetry, the other direction also applies, mutatis mutandis.

2.1.5. Ranking Questions via Genetic Algorithm

The use of genetic algorithm is coupled with the use of clustered-based sampling method discussed earlier. Details to the algorithm are given in section 2.1.4. Another obvious reason to use genetic algorithm as the optimization method is that we do not have the analytical form (at least not without sophisticated Monte Carlo estimation) of the gradient of the solution in this case. Consequently, numerical optimization techniques such as gradient decent, conjugate gradient, BFGS may not be applicable in this case. Another possible way to derive optimal question assignment strategy is use reinforcement learning (this is beyond the scope of the demo).

(a) cofi.py:

See assignment() and section 2.1.4 for more details

(c) evolution.py: main genetic optimization subroutine is defined here.

As of now, I am still gathering and analyzing data from the genetic optimization. However, the code works pretty okay at the moment except that the speed is not optimal. A sample run indicates that the fitness gradually increases over time … However, to justify this method more carefully, one needs to conduct multiple runs (set to 20 at the moment), take the average and draw error bars to observe if proper learning occurred. As a preview, in the appendix includes a sample run. You are welcome to play with the code to gather your own empirical data.

3. Ongoing Work

Here are a few examples: i) more detailed cluster analysis to identify similar questions and their relationship with student performance ii) completion of genetic optimization procedure. The code is implemented but it's time consuming to gather sufficient empirical data to justify the method; see section 5 for a sample run. In particular, the hyperparameters of the similarity measure such as the bandwidth of the squared-exponential kernel can be learned from the data towards producing better clustering results.

4. Reference

[1] D. Arthur and S. Vassilvitskii, "Kmeans++: The Advantages of Careful Seeding", ACM-SIAM Symposium on Discrete Algorithms, pages 1027–1035, 2007.

[2] Ulrike von Luxburg. A Tutorial on Spectral Clustering. Statistics and Computing, 17 (4), 2007.

[3] Amardeep Kaleka, Steven Greer. Sirius Documentary. 2013.

[4] Andreas Buja et al. Data Visualization with Multidimensional Scaling. Journal of Computational and Graphical Statistics, Volume 17, Number 2, pp 444–472, 2008.

[5] Arindam Banerjee, Joydeep Ghosh. On Scaling Up Balanced Clustering Algorithms.

5. Appendix

```
[iter=0] max fitness: 380441.7
[iter=1] max fitness: 436860.576
[iter=2] max fitness: 475495.2
[iter=3] max fitness: 507120.3
[iter=4] max fitness: 545067.756
[iter=5] max fitness: 546118.731
[iter=6] max fitness: 573192.675
[iter=7] max fitness: 609934.059
[iter=8] max fitness: 631221.075
```

```
[iter=9] max fitness: 645272.091
[iter=10] max fitness: 655480.116
[iter=11] max fitness: 659788.299
[iter=12] max fitness: 683329.896
[iter=13] max fitness: 690898.176
[iter=14] max fitness: 706230.576
[iter=15] max fitness: 714706.524
[iter=16] max fitness: 715351.851
[iter=17] max fitness: 719083.116
[iter=18] max fitness: 725054.499
[iter=19] max fitness: 729173.979
[iter=20] max fitness: 741783.771
[iter=21] max fitness: 744271.875
[iter=22] max fitness: 750421.8
[iter=23] max fitness: 767339.136
[iter=24] max fitness: 771379.659
[iter=25] max fitness: 774659.484
[iter=26] max fitness: 788402.619
[iter=27] max fitness: 788402.619
[iter=28] max fitness: 797342.571
[iter=29] max fitness: 801905.076
[iter=30] max fitness: 809333.064
[iter=31] max fitness: 809333.064
[iter=32] max fitness: 813721.419
[iter=33] max fitness: 813721.419
[iter=34] max fitness: 824444.856
[iter=35] max fitness: 836250.579
[iter=36] max fitness: 836250.579
[iter=37] max fitness: 837131.4
[iter=38] max fitness: 839583.675
[iter=39] max fitness: 846155.259
[iter=40] max fitness: 853560.891
[iter=41] max fitness: 855998.811
```

```
[iter=42] max fitness: 858518.964
[iter=43] max fitness: 860224.284
[iter=44] max fitness: 868610.475
[iter=45] max fitness: 871073.091
[iter=46] max fitness: 875114.856
[iter=47] max fitness: 877014.675
[iter=48] max fitness: 880340.859
[iter=49] max fitness: 887915.196
[iter=50] max fitness: 893859.516
[iter=51] max fitness: 895130.739
[iter=52] max fitness: 901445.499
[iter=53] max fitness: 903628.8
[iter=54] max fitness: 903846.204
[iter=55] max fitness: 905525.1
[iter=56] max fitness: 906040.8
[iter=57] max fitness: 913085.496
[iter=58] max fitness: 913085.496
[iter=59] max fitness: 916063.875
[iter=60] max fitness: 916063.875
[iter=61] max fitness: 918863.064
[iter=62] max fitness: 936181.764
[iter=63] max fitness: 938177.379
[iter=64] max fitness: 943281.171
[iter=65] max fitness: 949167.0
[iter=66] max fitness: 949167.0
[iter=67] max fitness: 964955.619
[iter=68] max fitness: 964955.619
[iter=69] max fitness: 967499.136
[iter=70] max fitness: 979828.731
[iter=71] max fitness: 987580.8
[iter=72] max fitness: 989650.179
[iter=73] max fitness: 998065.584
[iter=74] max fitness: 1012368.744
```

```
[iter=75] max fitness: 1012769.379
[iter=76] max fitness: 1022887.8
[iter=77] max fitness: 1022887.8
[iter=78] max fitness: 1026170.856
[iter=79] max fitness: 1034446.356
[iter=80] max fitness: 1034446.356
[iter=81] max fitness: 1039690.179
[iter=82] max fitness: 1041540.939
[iter=83] max fitness: 1044171.864
[iter=84] max fitness: 1046034.324
[iter=85] max fitness: 1046034.324
[iter=86] max fitness: 1047912.984
[iter=87] max fitness: 1055748.6
[iter=88] max fitness: 1057641.3
[iter=89] max fitness: 1069895.475
[iter=90] max fitness: 1069895.475
[iter=91] max fitness: 1079836.776
[iter=92] max fitness: 1081596.699
[iter=93] max fitness: 1083354.156
[iter=94] max fitness: 1084641.444
[iter=95] max fitness: 1085184.0
[iter=96] max fitness: 1088230.131
[iter=97] max fitness: 1088230.131
[iter=98] max fitness: 1093274.1
[iter=99] max fitness: 1094550.804
[iter=100] max fitness: 1101310.731
[iter=101] max fitness: 1101610.251
[iter=102] max fitness: 1104604.2
[iter=103] max fitness: 1104833.691
[iter=104] max fitness: 1106154.756
[iter=105] max fitness: 1112035.419
[iter=106] max fitness: 1113805.836
[iter=107] max fitness: 1114560.675
```

```
[iter=108] max fitness: 1118742.651
[iter=109] max fitness: 1118742.651
[iter=110] max fitness: 1120140.216
[iter=111] max fitness: 1132656.444
[iter=112] max fitness: 1133246.475
[iter=113] max fitness: 1136426.859
[iter=114] max fitness: 1136950.875
[iter=115] max fitness: 1137371.499
[iter=116] max fitness: 1138311.144
[iter=117] max fitness: 1139328.675
[iter=118] max fitness: 1143196.164
[iter=119] max fitness: 1144192.824
[iter=120] max fitness: 1146088.764
[iter=121] max fitness: 1151996.931
[iter=122] max fitness: 1153796.976
[iter=123] max fitness: 1154009.7
[iter=124] max fitness: 1155820.275
[iter=125] max fitness: 1157347.584
[iter=126] max fitness: 1157538.699
[iter=127] max fitness: 1158441.939
[iter=128] max fitness: 1185221.916
[iter=129] max fitness: 1187950.059
[iter=130] max fitness: 1190174.931
[iter=131] max fitness: 1190174.931
[iter=132] max fitness: 1191519.675
[iter=133] max fitness: 1191519.675
[iter=134] max fitness: 1193458.176
[iter=135] max fitness: 1193805.216
[iter=136] max fitness: 1194374.475
[iter=137] max fitness: 1208820.636
[iter=138] max fitness: 1210319.676
[iter=139] max fitness: 1212239.484
[iter=140] max fitness: 1213015.824
```

```
[iter=141] max fitness: 1213015.824
[iter=142] max fitness: 1213308.819
[iter=143] max fitness: 1214082.459
[iter=144] max fitness: 1215693.675
[iter=145] max fitness: 1216715.4
[iter=146] max fitness: 1218632.211
[iter=147] max fitness: 1222236.459
[iter=148] max fitness: 1222533.099
[iter=149] max fitness: 1226437.875
```