Московский Государственный Университет
имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Дипломная работа

Разработка и исследование алгоритма PSO для решения задачи кластеризации документов

Выполнил студент 521-й группы
кафедры АСВК Хоанг Т. Л.

Научные руководители:
д.ф.-м.н., профессор, член-корр. РАН Королёв Л.Н.
к.ф.-м.н., доцент Попова Н.Н.

Москва
Май 2007

Содержание

Аннотация

Алгоритм оптимизации стаи (Particle swarm optimization или PSO) появился в 1995 году, и является вариантом эволюционных вычислений (Evolutionary computing). Алгоритм PSO часто используется для решения задач оптимизации. В рамках дипломной работы разрабатывается улучшение алгоритма оптимизации стаи. Результатjv исследования является разработка нового варианта алгоритма - “Heuristic Particle Swarm optimization (HPSO)”. Экспериментальные результаты на стандартном наборе тестов показали, что новый алгоритм даёт лучшие результаты (по скорости сходимости, качеству решения и устойчивости алгоритма) относительно базисного алгоритма оптимизации стаи и его недавних вариантов. В работе рассмотрено применение PSO алгоритма к задаче кластеризации документов. Для сравнения эффективности выполнения HPSO алгоритма со стандартным алгоритмом PSO и K-Mean алгоритмом были проведены эксперименты на наборах тестов с очень большим количеством документов.

Введение

Эволюционные вычисления являются важной частью искусственного интеллекта. Эволюционные вычисления имеют широкое применение, например, генетическое программирование, искусственная жизнь (Artificial Life) в робототехнике, иммунная система, нейтронная сеть в области распознавания образов, particle swarm optimization (PSO), ant colony, co-evolutionary algorithm, в оптимизации. Эти методы имеют общую основу – они все используют известные модели в биологии.

Целью дипломной работы является исследование алгоритма PSO, дальнейшее его улучшение и применение к задаче кластеризации документов. Алгоритм PSO, предложенный в 1995 году [1], является вариантом эволюционных вычислений. Сравнительно с другими эволюционными алгоритмами, PSO быстрее сходится к решению. Однако, алгоритм PSO не всегда находит верное решение. Алгоритм интенсивно исследуется и развивается.

В настоящее время предложено несколько вариантов PSO [1,2,5,7,8,9]. В дипломной работе предлагается дальнейшее развитие алгоритма PSO, основанное на использовании эвристического выбора следующих частиц для обновления и механизма переинициализации векторов скорости частиц при появлении признаков ранней сходимости. За основу предлагаемого метода улучшения алгоритма взяты идеи традиционного эвристического поискового алгоритма. Результатом разработки является новый алгоритм Heuristic PSO (HPSO). Разработанный алгоритм протестирован на теоретическом наборе тестов и на прикладной задаче. Новый алгоритм продемонстрировал лучшие результаты по качеству полученных решений, по устойчивости и по скорости сходимости алгоритма в сравнении с другими вариантами PSO. Результаты работы представлены на международную конференцию по эволюционным вычислениям “Genetic and Evolutionary Computation Conference” (GECCO’07) University Of College London [30].

Дипломная работа состоит из Введения, двух глав, заключения и списка литературы. В первой главе дипломной работы представлены основные понятия, идеи алгоритма PSO и описаны предлагаемые методы улучшения алгоритма , идеи и описание алгоритма HPSO. В главе рассмотрено сравнение результатов работы разработанного алгоритма HPSO с другими вариантами алгоритма PSO на теоретическом наборе тестов.….

Во второй главе исследовано применение разработанного алгоритма HPSO к задаче кластеризации документов, приведены сравнительные результаты работы нескольких вариантов PSO и разработанного алгоритма HPSO на некоторых наборах документов.

Глава I

Разработка эвристического алгоритма оптимизации, основанного на базе алгоритма PSO.

`Cheshire Puss,' she began, rather timidly.. `Would you tell me, please, which way I ought to go from here?'

`That depends a good deal on where you want to get to,' said the Cat.

`I don't much care where--' said Alice.

`Then it doesn't matter which way you go,' said the Cat.

`--so long as I get *SOMEWHERE*,' Alice added as an explanation.

`Oh, you're sure to do that,' said the Cat, `if you only walk long enough.'

—Lewis Carroll, *Alice in Wonderland*

1. Обзор алгоритм оптимизации стаи PSO

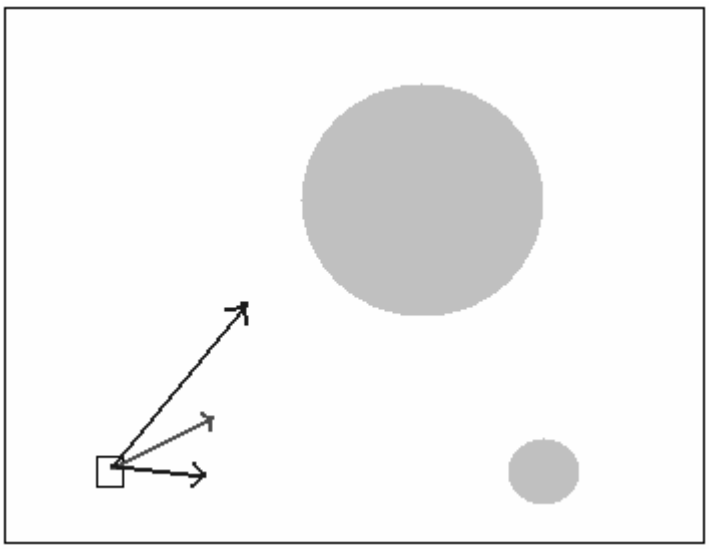
1.1 Стая и интеллект стаи

Идея алгоритма оптимизации стаи опирается на наблюдении того, что происходит в природе, более точно, в биологии. Термин “Bio-inspired” означает копирование человеком действий, специальных способностей животных, растений чтобы создавать полезные средства в жизни. Примеры “Bio-inspired” систем существуют везде. Чтобы летать как птицы, люди создали самолёты с крыльями. Чтобы создавать средства накопления рентгеновских лучей света, человек наблюдал глаза омаров. Биологи верят, что природа всегда впереди, человек следует и открывает тайну природы.

Алгоритм оптимизации стаи является одним из многих примеров “bio-inspired” алгоритмов. Алгоритм PSO, как и другие эволюционные алгоритмы (генетический алгоритм, Ant colony optimization, immune system, artificial life), опирается на биологические модели, или биологические основы. Чтобы понять алгоритм оптимизации стаи, надо выяснить понятие “интеллект стаи”, потому что идея алгоритма основана на этом понятии.

Стая - совокупность птиц, животных. Первый раз описали алгоритм оптимизации стаи в статье IEEE конференции в 1995 году , Kennedy и Eberthart - основатели алгоритма описали стаю как стая птиц, но не важно это стая птиц, рыбак, или человек, интеллект стаи существует на любой стае, мы видим такой интеллект каждый день.

Пусть например стая птиц, стая птица содержит частицы (в этой работе согласен, что термины “частица” и “птица” имеют одна и та же смыль), которые летают и ищут пищу. Каждая птица летает на свой пути, кроме этого наблюдает других. Когда одна из них нашла перспективную область, в которой содержит признаки пищи, сразу исходит сигналы от неё к остальным, которые летает недалеко от неё. Получая сигналы, они регулируют их скорости и направления к этому место. Мы видим такое событие, каждый раз кормит стаю голуби, при бросании пищу, частицы стаи сразу летают к место пищи благодаря специальному механизму, через который информация распространяется на всю стаю. Стало понятно, что механизм распространения информации передает частицам стаи полезные информации или на другом слове “знание” о место пищи. Такой механизм называется интеллектом стаи.



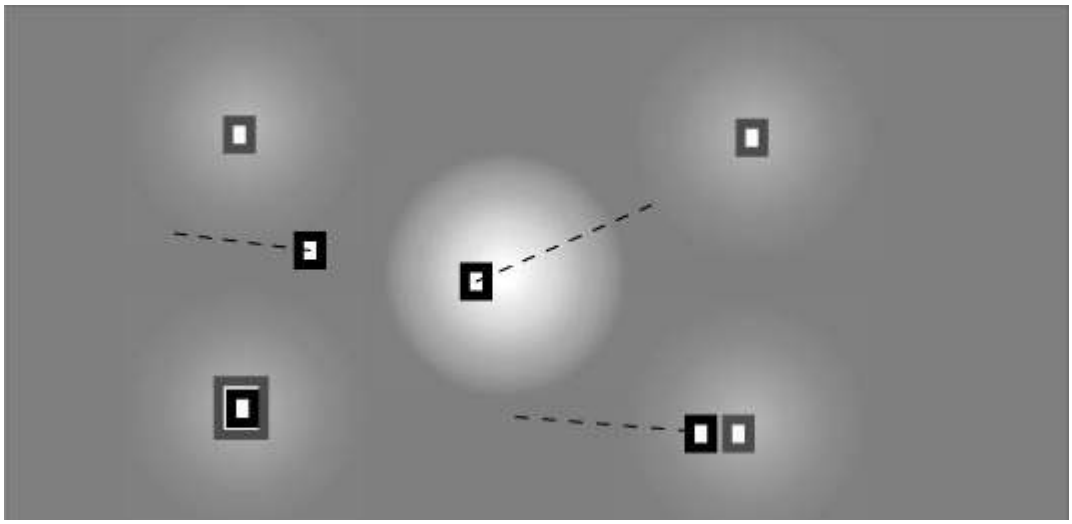
Рисунка 1: поведение птицы

На рисунке 1 показывает одну птицу и разные информации о место пищи. Красная стрелка покажет настоящее направление и скорость птиц. Бывает два сигнала о место

пищи, одно с маленькой окружностей и другое с большой окружностей, чем большая окружность, тем много пищи в этом место. Как птица регулирует свою скорость и направление зависит от интеллекта стаи. Две голубые стрелки покажут разные выборы птиц. Мы уже определили интеллект стаи, а как применяет и моделирует интеллект стаи чтобы служить человеческую жизнь, ответ на этот вопрос будет на следующей часть.

1.2 Моделирование интеллект стаи - алгоритм оптимизации стаи (PSO)

Большинство проблем информационной технологии приводится к оптимизационным проблемам. Разработка это работа учёных информационной технологии. Мы посмотрим задачу оптимизации и моделирование интеллект стаи для решений этих задач. Подобие между поиском пищи и поиском оптимальных решений функции очевидно, если предложим что, пространство в котором птицы летают и ищут пищу это N-мерное пространство (поисковое пространство), а места пищи является областей, в которой содержит оптимальную точку функции F, определенную в этом пространстве. Каждая локальная оптимальная точка функции соответствует одно место пищи. На рисунке 2[1] покажет поисковое пространство. Освещение области соответственно места пищи (оптимальные точки). Самая большая область освещения соответственна глобальная оптимальная точка функции F (место содержит самое большое количество пищи).



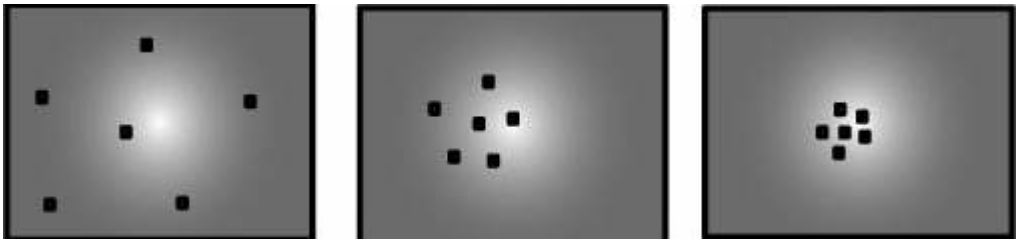
Рисунка 2: поисковое пространство

На рисунке 3 покажет стаю с 6 частиц. Сначала они распределяются по разным областям поискового пространства и потом летают к области пищи. Подобие между поиском пищи и поиском оптимальных решений функции разрешает нас построить искусственную стаю для поиска оптимальной точки функции. Как построить искусственную стаю и их поведения, чтобы искусственная стая действует как реальная стая? Эта проблема называется моделированием и может делать по разному. От момента рождения PSO до настоящего времени существует только один метод моделирования [1] и такое моделирование действительно похоже на реальный биологический модель, вопрос о том что, можно ли построить по другому, пока открыт. Я не дошел до конца ответа этого вопроса, но верю что ответ вопроса интересно и полезно для развития алгоритма. В рамках наше работы мы посмотрим только моделирование, которое предложили Kennedy и Eberthart.

Механизм распространения информации, описанный выше существует в биологической жизни стаи, для моделирования этого механизма Kennedy и Eberthart предложили модель,

[1] Рисунки 2,3,4 скопированы с **[2]** спасибо Beville Jean за разрешение использования его ресурсы.

в котором каждая частица обладает вектором скорости V_i и вектором позиции X_i, для задач оптимизации N-мерной функции, такие векторы имеют размерности N. Как в биологической жизни Kennedy и Eberthart предполагали что, каждая птица регулирует свои векторы позиции и скорости по своим опытом (когнитивный-cognitive) и по полученным информациям от других (социальный-Social interaction). Здесь опыт птицы понимается как её знание о лучшей позиции (место пищи), через которую она сама пролетела, а социальное знание птицы понимается как её знание о лучшей позиции (место пищи), через которую прошла одна из птиц в одной группе с ней. В задаче оптимизации лучшая позиция птицы понимается как позиция, в которой значения функции качества(функция должна минимизировать) наименее. Чем менее значения функции качества, тем лучшая эта позиция.



Рисунка 3: стая ищет пищу

Простая версия PSO алгоритм обновляет векторы позиции и скорости птицы по формуле:

$$V_i = V_i + c_1.r_1.(G - X_i) + c_2.r_2(P_i - X_i) \quad (1)$$

$$X_i = X_i + V_i \quad (2)$$

V_i вектор скорости i-той птицы,

X_i вектор позиции i-той птицы,

P_i лучший вектор позиции достигала i-тая птица (Personal best),

G лучше всех вектор позиции достигала группа, $G = Maximum_{j \in \Gamma_w}(P_j)$ (Global best)

где Γ_i группа птицы содержит i-тую птицу.

c_1, c_2 параметры алгоритма.

r_1, r_2 случайные числа в интервале (0,1)

c_1, c_2 контролируют степени важности социального знания и когнитивного знания, чем больше это значение чем важнее соответственное знание. Здесь очевидно, что птица обновляет свою позицию к лучшей позиции группы или к её лучшей позиции, которую прошла в прошлом. Если c_1 имеет большое значение, стая намеревается широко поиск, а если c_2 имеет большое значение, то стая намеревается поиск в узкой области поискового пространства.

r_1, r_2 выберется как случайные числа в интервале (0,1) чтобы поддерживать разные траектории птиц при поиске.

В несколько версиях PSO [3], авторы ограничат элементы векторов скорости не превосходиться предопределенное значение $V_{\max}$, чтобы уверять сходимости стаи. Но иногда можно гарантировать сходимости стаи по-другому, поэтому некоторые авторы не используют ограничение скорости в своих версиях [4]**.** Формула ограничения скорости показывает в (3):

$$\textbf{\textit{If}}\ (V_{id} > V_{\max})\ V_{id} \leftarrow V_{\max}$$

$$\textbf{\textit{If}}\ (V_{id} < -V_{\max})\ V_{id} \leftarrow -V_{\max} \quad (3)$$

Высшие формулы показываются, что птица обновляет свои векторы по векторам G (Global best) и P_i (Personal best), эти векторы обновляет в каждом шаге по формуле (4):

If $(f(X_i) < P_i)$ $P_i \leftarrow f(X_i)$

If $(f(X_i) < G)$ $G \leftarrow f(X_i)$ (4)

Где $f(X_i)$ значение функции качества f в точке X_i.

Pseudo-код ниже показывает как алгоритм оптимизации стаи работает:

```
InitializationSwarm()
While (!EndCondition)
{
 For  i=0 to Npop
  {
     Update vector velocity: using (1)
     Check the velocity: using (3)
     Update vector  position: using (2)
     Update personal best position and global best position: using (4)
  }
}
```

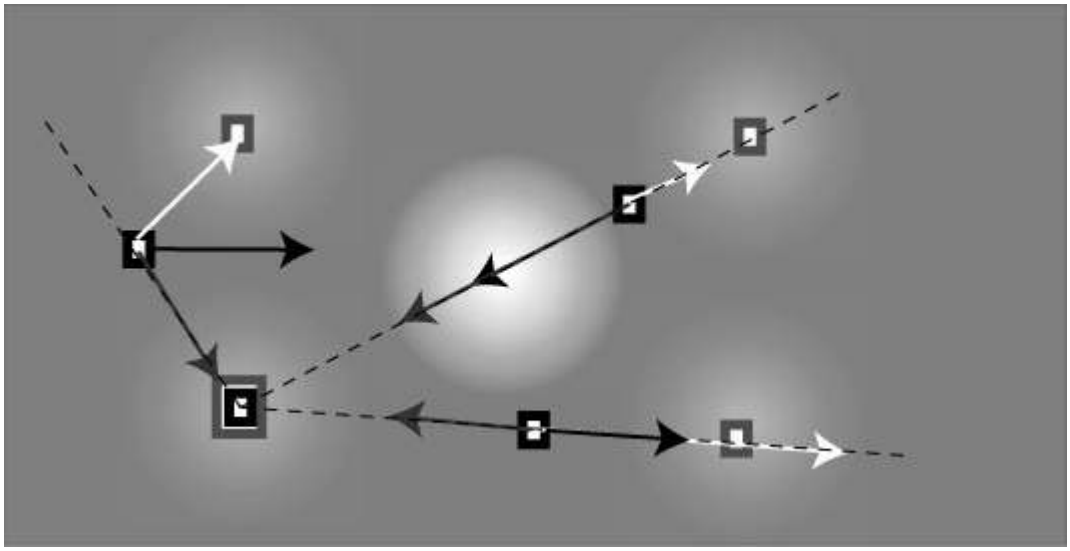
где *InitializationSwarm()* инициализирует начальные векторы скорости и позиции, обычно векторы позиции выбирают случайно в пространстве определения функции качества, а векторы скорости начинаются от начала координата.

Npop : число птиц в стае.

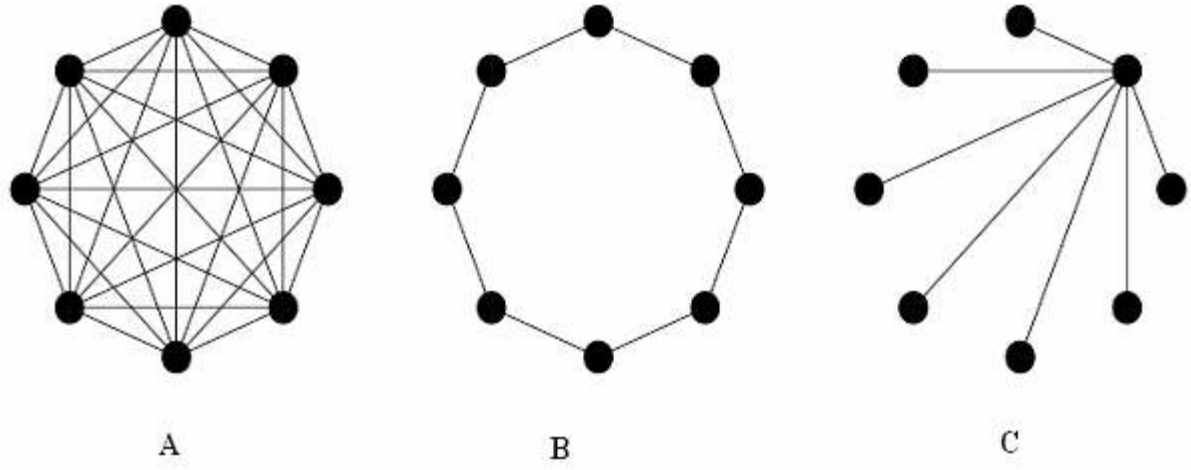
EndCondition: либо является ограничением числа итерации, которые программа исполняется, либо является условием полученных решений.

На рисунке 4 демонстрирует птицы с векторами скорости, 4 чёрные квадраты соответственно настоящие позиции четырёх птиц, а 4 красные квадраты соответственно их лучшим позициям (Personal best). Самой большой красный квадрат соответствует лучшей позиции группы (Global best). Каждая птица регулирует свою скорость (чёрный вектор) как комбинация белого вектора (направлен к её лучшей позиции) и бурого вектора (направлен к лучшей позиции группы).

В формуле обновления вектора скорости используется лучшую позицию группы (Global best position of group), вся стая разделяется на группы. Каждая птица используется лучшую позицию группы, в которой участвует она, в её формуле обновления. Есть некоторые варианты разделения стаи на группы, такое разделение называется топологией соседи.



Рисунка 4: Каждая птица регулирует свою скорость (чёрный вектор) как комбинация белого вектор (направлен к её лучшей позиции) и бурого вектор (направлен к лучшей позиции группы)



Picture 5: A. gbest топология B. k-best топология C. wheel топология

Например, на рисунке 5 иллюстрирует разные топологии соседи стаи. Каждая топология отличается от других по степени соединения. На рисунке 5.A показано полное соединение топологии gbest, а в остальных показано топологию с меньшей степеней соединения. Топология соседи влияет на работы алгоритма, gbest топология помогает стаю быстро передать информации о лучшей позиции стаи, а остальные топологии передает информации медленнее. В соответствии с [5] алгоритм PSO с gbest топологией стая обычно быстро сходится, но может сходиться рано либо к локальному оптимуму либо просто к точке, которая не является оптимумом.

2. Некоторые варианты PSO

2.1 Стандартный вариант

В настоящее время разработано несколько вариантов алгоритма PSO [1,3,4,6,8,9]. Рассмотрим базовый вариант (BPSO) [6]. В этом варианте предложено понятие инерциального веса "inertia weight " ω, на который умножается на предыдущая скорость частицы для контроля поисковой способности. В первых вариантах базового алгоритма ω выбиралась постоянным. Затем были предложены некоторые варианты, в которых ω изменяется шаг по шагом [7]**.** В соответствии с [6], Pseudo-код базисного варианта PSO описан в фрагмента кода ниже:

```
InitializationSwarm()
ω = ω_max
While (!EndCondition)
    Update ω : using (6)
    For i=0 to Npop
        Update vector velocity: using (5)
        Check the velocity: using (3)
        Update vector position: using (2)
        Update personal best position and global best position: using (4)
```

Рисунка 6: Pseudo-code базисного варианта PSO

В отличие от варианта описано в 1.2, базисный вариант PSO используется другую формулу обновления вектора скорости - по формуле (5)

$$V_i = \omega V_i + c_1.r_1.(G - X_i) + c_2.r_2(P_i - X_i) \quad (5)$$

Кроме этого формула обновления "inertia weight" описается в (6), в этой формуле ω изменяется линейно от $\omega_{\max}$ до $\omega_{\min}$ по итерации, MaxIter является числом обновления

позиции каждой птицы или число итерации алгоритма, MaxIter обычно указывается пользователем алгоритма.

$\omega = \omega - (\omega_{max} - \omega_{min})/\text{MaxIter}$ (6)

По [6] ω_{max}=0.9, ω_{min}=0.4 и c1=c2=2.0, большое значение ω даёт стаю возможности широкого поиска, а маленькое значение ω даёт стаю возможности точной настройки (fine tuning) решений, поэтому с начало ω получает большое значение и уменьшается шаг по шагом.

2.2 Расширенные варианты

Существует несколько расширенных вариантов PSO. Рассмотрим некоторые из них.

Проведенные исследования PSO алгоритма показывались, что PSO ставит перед проблемой ранней сходимости [2], эта проблема встречается более часто в варианте с gbest топологией. Чтобы устранить эту проблему есть много подходов, один пример является [5]. В этой работе авторы используются механизм предсказания конфликта, перед чем, как обновляет свою позицию, каждая птица смотрит, что в пространстве будет ли конфликт, значит если минимальное расстояние между новой позицией и позицией остальных птиц стаи меньше предопределенного параметра конфликта, то эта птица обновляет позицию по обратному направлению вектора скорости. Если нет конфликта, то обновление проводится по-обычному. Таким образом, механизм предсказания помогает птице обнаружить конфликт и благодаря этому предупредит ранней сходимости.

Использование механизм предсказания конфликтов действительно улучшить работу PSO алгоритма, особенно в задаче с большим количеством оптимума, в которой поисковой алгоритм может попадать в локальном оптимуме и трудно избежать от этой мертвой точки. Но обновление позиции по обратному направлению вектора скорости может уменьшать скорость сходимости.

По этой причине были некоторые работы, которые решают эту проблему по-другому, пример “Gregarious PSO” (GPSO) в [8] является одном из них. Алгоритм GPSO ускоряет скорость сходимости путём использованием только социальную части в формуле обновления вектора скорости, т.е. делает проблему ранней сходимости более часто. Чтобы удаляется от мертвой точки, авторы используют механизм инициализации вектора скорости, когда одна птица очень близко (расстояние меньше предопределенного параметра) к лучшей позиции группы. Такой механизм ограничения расстояния между каждой позицией птицы и лучшей позицией группы использовались раньше в HPSO-TVAC [9]. Алгоритм GPSO применит этот механизм и действительно сдаёт перспективные результаты на стандартные наборы тестов.

3. Методы улучшения алгоритма оптимизации стаи частиц.

Основная идея нового алгоритма базируется на идеях традиционного эвристического поискового алгоритма.

Традиционный эвристический поисковый алгоритм использует оценочную функцию для принятия решения, т.е. следующий шаг поискового процесса зависит от значения оценочной функции. Чем меньше значение оценочной функции, тем лучше. Простым примером является игра шахматы, в этой игре, каждой шаг оценивается значением, и зависит от этого значения, игрок принимает решения. Основные понятия эвристического поискового алгоритма описаны в [10].

Следуя идее традиционного эвристического поискового алгоритма, HPSO алгоритм использует эвристику для принятия решения. Выбор следующей птицы для обновление опирается на значение оценочной функции. Как мы уже знали в базисном варианте и также в других вариантах алгоритма оптимизации стаи авторы не использовали эвристику для выбора следующей птицы на обновление, а порядок выбора фиксируется. Все птицы играют равномерные ролы в процессе поиска. Такая схема выбора на обновление похожа на схему, использующую в алгоритме поиска в ширину.

В HPSO все птицы стаи не рассматриваются одинаково, только птица с лучшим значением оценочной функции выбирается на обновление. Оценочная функция может быть разными, в этой части мы посмотрим самый простой случай, это оценочная функция совпадает с функцией качества. Продолжение работы на HPSO пока открыто, действительно, что в рамках дипломной работы, я стараюсь только предложить основную идею и основные результаты. Как мы будем смотреть, бегает по этой пути можно получать перспективные результаты.

Кроме использования эвристики в выборе на обновление, HPSO использует механизм инициализации вектора скорости, когда одна птица очень близко (расстояние меньше предопределенного параметра) к лучшей позиции группы как в [8-9]. Использования эвристики в выборе на обновление увеличивает вероятности ранней сходимости, поэтому механизм инициализации вектора скорости является необходимым, для того чтобы предупреждать раннюю сходимость. Алгоритм HPSO использует формулу обновления скорости, как описано в [8-9]. В этой формуле предыдущая («старая») скорость не присутствует. Таким образом, предлагаемый метод может ускорить скорость сходимости к оптимальному решению. Комбинация трёх описанных механизмов: использования эвристики в выборе для обновления, механизм инициализации вектора скорости , и использование формулы обновления скорости как в [8-9], является основами HPSO. Рисунка 7 иллюстрирует работы алгоритма HPSO.

Описание HPSO

$t \leftarrow 1$
Swarm's initialization
While $(t \le MaxIter * Npop)$
  Choose next particle: using (7)
  If (Distance(X_{Next} ,gbestposition)$\le \lambda$)
    $V_{Next,j} \leftarrow random(D_{\max}, D_{\min}) \quad \forall j = 1,..,d$
  else
    Update velocity: using (8)
    Limit velocity in range $[-V_{\max}, V_{\max}]$
  Update position: using (9)
  Fitness Function evaluation: $f(X_{Next})$
  Personal best: pbest$\leftarrow f(X_{Next})$ if pbest > $f(X_{next})$
  Global best: gbest$\leftarrow f(X_{Next})$ if gbest > $f(X_{Next})$

Рисунка 7: Pseudo-код HPSO

[Где *MaxIter*: число итерации
Npop: число птиц в стае
Distance(X,Y) : расстояние между векторами X и Y.
Операторы (7)-(9) описаны ниже:

$$f(X_{Next}) = Min\{f(X_i)\} \quad i = 1,.., Npop \qquad (7)$$

$$V_{id} = c_1 . r_1 .(P_{id} - X_{id}) + c_2 . r_2 .(G_d - X_{id}) \qquad (8)$$

$$X_{id} = V_{id} + X_{id} \qquad (9)$$

Если расстояние между позицией птицы и лучшей позицией группы меньше, чем преопределённого константа λ, тогда вектор скорости инициализируется в интервале $(D_{\min}, D_{\max})$, где $D_{\min}$ и $D_{\max}$ является параметрами алгоритма. Выбор значения для этих параметров проблемно зависимо, в следующей части предполагается метод выбора этих параметров. Ещё надо заметить, что в HPSO алгоритме в формуле обновления скорости нет части старой скорости (ω=0) как в HPSO-TVAC [9].

Поведение птиц в HPSO алгоритме описано таким образом, что только самые лучшие птицы летают и ищут пищу в области пищи, а остальные останавливаются. Если в

течение времени птицы ни как не найдёт лучшую позицию в этой области, они шаг за шагом удаляются от этой области и дают остальным птицам возможность летать. Поведение птиц в HPSO алгоритме отличается от поведения птиц в BPSO в том, что он всегда предпочитает поиск в надежной области. Это является основной идеей алгоритма HPSO. В нашем обществе, талантливый человек обычно получаются высшие привилегия, и они действительно является пионерами в пути поиска лучших средств, поддерживающие нашу жизнь.

4. Экспериментальные результаты HPSO на наборе стандартных тестов

Чтобы продемонстрировать эффективность работы нового алгоритма, в этой части рассмотрено несколько экспериментов на наборах стандартных функции. Эти функции описаны и выбраны для тестирования эффективности многих эволюционных алгоритмов оптимизации [7] и являются фактически стандартными для оценки эффективности работы поискового алгоритма [4-11]. Каждая функция обладает специальными свойствами В таблице 1 описаны математические формулы этих функции, в таблице 2 описаны соответствующие начальные значение векторов позиции и ограничение поисковой области. D(d) есть размерности поисковой пространства.

Function Name	**D**	**Mathematical Representation**
Sphere $f_1(x)$	30	$\sum_{i=1}^{d} x_i^2$
Rastrigin $f_2(x)$	30	$\sum_{i=1}^{d} (x_i^2 - 10\cos 2\pi x_i + 10)$
Rosenbrock $f_3(x)$	30	$\sum_{i=1}^{d-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$
Griewank $f_4(x)$	30	$\frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos\frac{x_i}{\sqrt{i}} + 1$
Ackley $f_5(x)$	30	$-20\exp(-0.2\sqrt{\frac{1}{d}\sum_1^d x_i^2})$ $-\exp(\frac{1}{d}\sum_1^d \cos 2\pi x_i) + 20 + e$
Schaffer's $f_6(x)$	2	$0.5 + \frac{(\sin\sqrt{x^2+y^2})^2 - 0.5}{(1.0 + 0.001(x^2+y^2))^2}$
Shekel's Foxhole $f_7(x)$	2	$(\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6})^{-1}$

Таблица 1: Математическое описание тестов

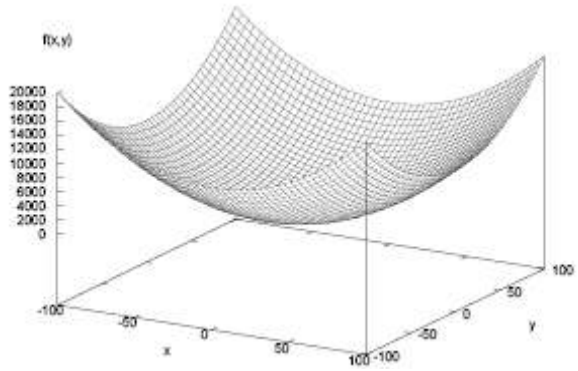
Function	Search range	Initialization range
f_1	$[-100,100]^d$	$[50,100]^d$
f_2	$[-10,10]^d$	$[2.56,5.12]^d$
f_3	$[-100,100]^d$	$[15,30]^d$
f_4	$[-600,600]^d$	$[300,600]^d$
f_5	$[-32,32]^d$	$[15,32]^d$
f_6	$[-100,100]^d$	$[15,30]^d$
f_7	$[-65.536,65.536]^d$	$[0,65.536]^d$

Таблица 2: Области определения функции и область инициализации векторов позиции

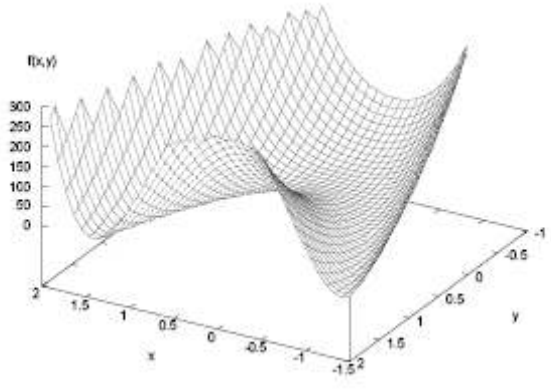
Каждая функция в наборе тестов имеет свои особенности. Рисунки 8-14 показывает виды графики каждой функции. Все функция имеет единственная глобальная точка минимума, значение минимума равно нулю.

Функция сфера является самым простым случаем, на рисунке 8 показывает её графику в трёх мерном пространстве. Такая функция имеет только один оптимум, значение которого равно нулю, и этот оптимум совпадает с глобальным минимумом в точке начала координата (0,0,..,0). Включение такая функция в наборе тестов чтобы оценить скорость сходимости поискового алгоритма в простом случае.

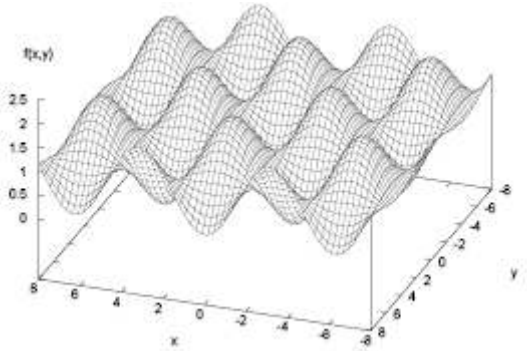
Функция Rosenbrock так же как функция сфера имеет только один оптимум, значение которого равно нулю, и этот оптимум совпадает с глобальным минимумом в точке (1,1,..,1). Но для поиска минимума более трудно, так как переменные сильно зависят друг от друга и вектор градиента не направлен к точке оптимума



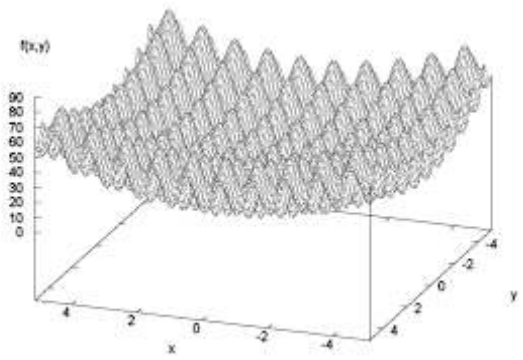
Рисунка 8: Графика функции сфера



Рисунка 9 : Графика функции Rosenbrock

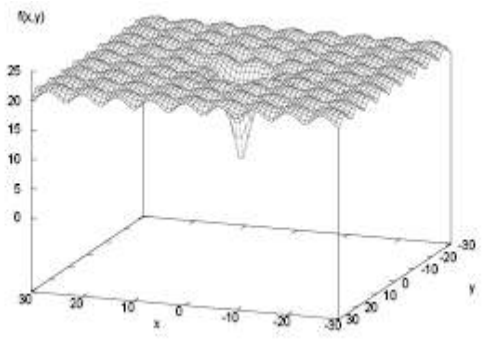


Рисунка 10 : Графика функции Griewank

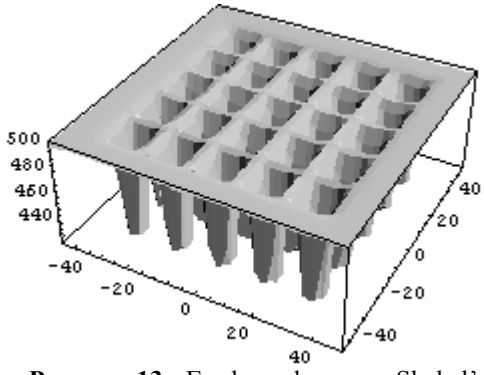


Рисунка 11 : Графика функции Rastrigin

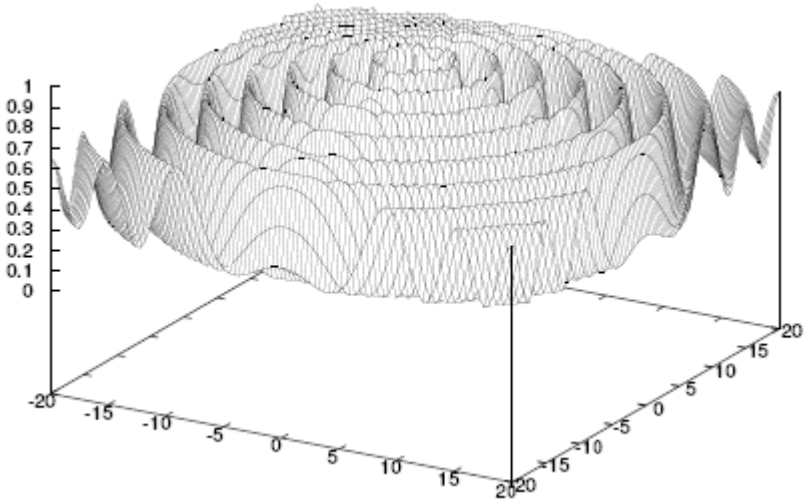
.



Рисунка 12 : Графика функции Ackley



Рисунка 13 : Графика функции Shekel's Foxhole



Рисунка 14: Графика функции Schaffer's

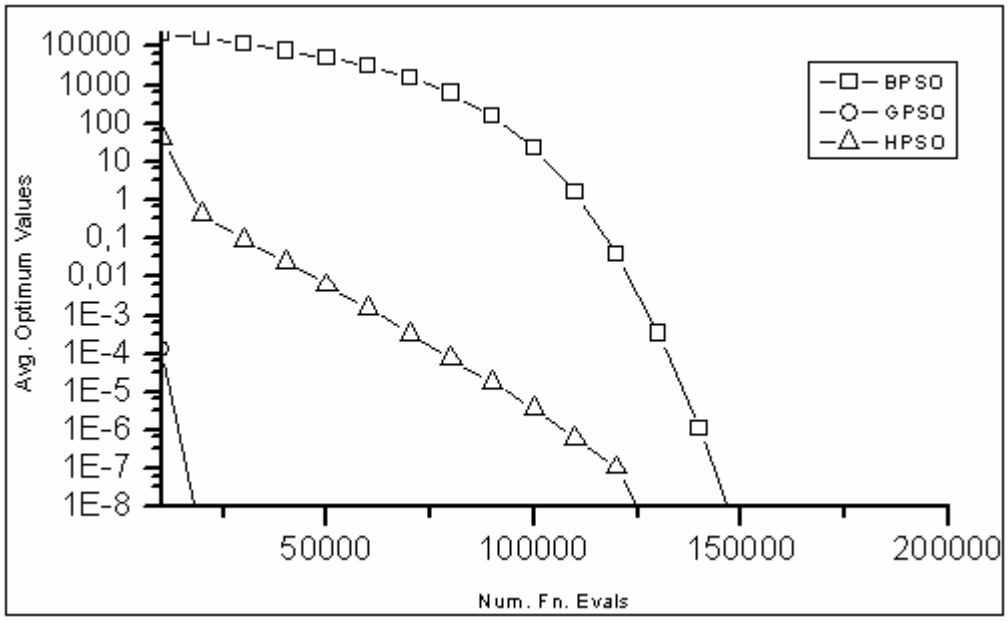
Функция Griewank является мультимодальной (multimodal), то есть число оптимумов функции возрастает по размерности пространства определения, функция имеет только одна глобальная точка минимума в точке начала координата (0,0,..,0), значение минимума равно нулю. Когда размерность пространства определения больше 30, вид графики функции выглядит как унимодальная (unimodal) функция.

Функция Rastrigin является мультимодальной, имеет очень много точек оптимума, но имеет только одна глобальная точка минимума в точке начала координата (0,0,..,0), значение минимума равно нулю.

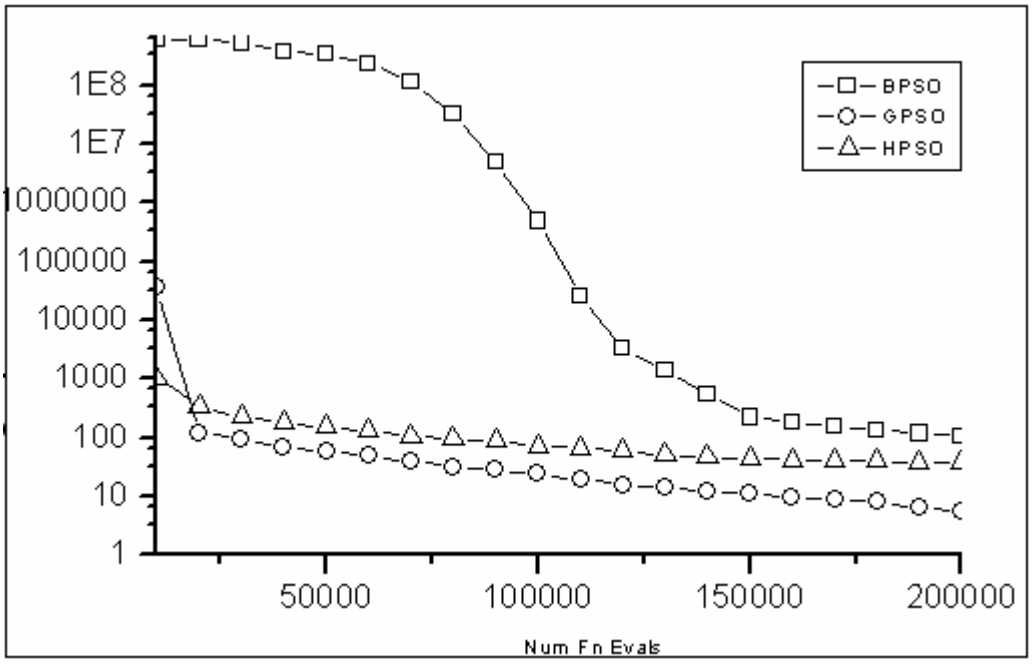
Функция Acley является мультимодальной, имеет очень много точек оптимума, но имеет только одна глобальная точка минимума в точке начала координата (0,0,..,0), значение минимума равно нулю.

Функция Shekel's Foxhole как показана на рисунке 13, имеет 25 оптимумов, среди которых один глобальный минимум в точке (-31.95,-31.95), значение которого равно 0,998. Такая функция оценит способности избежания от локальной минимума поискового алгоритма.

Функция Schaffer's как показана на рисунке 14, имеет много оптимумов, много долин вокруг глобального максимума в начала координата (0,0), значение которого равно 1. Такая функция оценит способности избежания от локальной минимума поискового алгоритма . Отметим что в нашем эксперименте мы исследуем частной случай функции Schaffer's как описано в таблице 1, такая функция так же как функция Schaffer's , имеет много оптимумов но отличается от неё в том что имеет глобальный минимум в начала координата (0,0), значение которого равно 0.



Рисунка 15 : Экспериментальные результаты на Sphere



Рисунка 16 : Экспериментальные результаты на Rosenbrock

Параметры селекции

В HPSO алгоритм много параметров и выбор значения для них является не простой задачей. В этой части описываются методы выбора параметров и дается объяснение на их выбор.

Выбор значения λ был описан в [8-9]. Для исследуемых тестов выбрано значение $\lambda=10^{-8}$. Значение λ в других тестах необязательно равно 10^{-8}. В случае больших размерностей поискового пространства значение λ дольно быть увеличено. В противном случае увеличивается риск того, что птицы не могут попадать в область вокруг Gbest позиции. Во второй главе значение λ выбирается совсём по-другому.

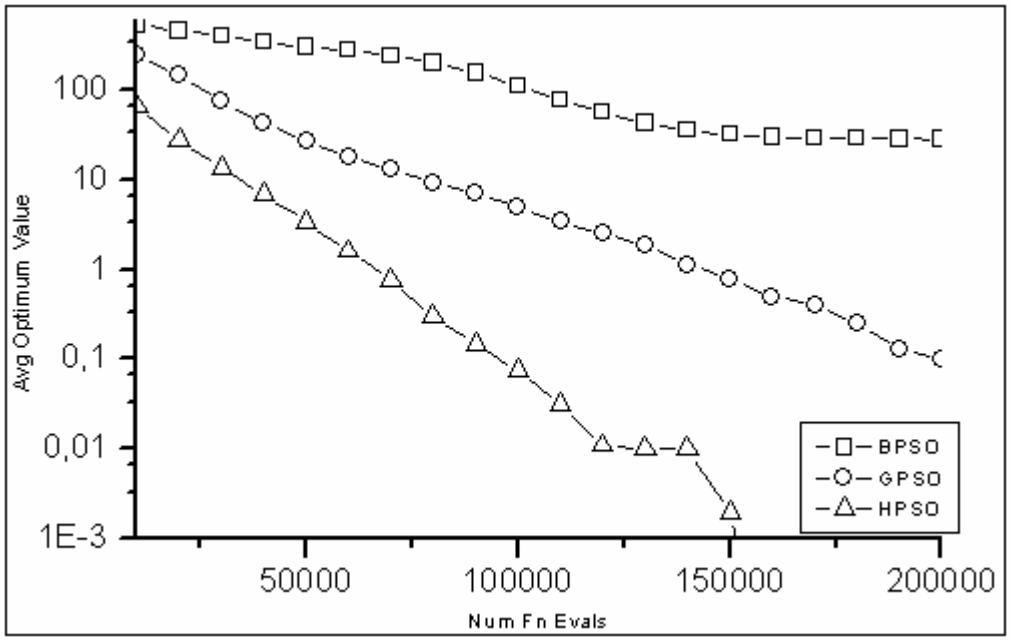
Vmax: - максимальное значение скорости выбираются так же как в [8-9]. Выбор значения $D_{\min}$ и $D_{\max}$ был описан в [8-9], и равны Vmax . В работе предлагается выбирать маленькие значения $D_{\min}$ и $D_{\max}$, чтобы ободрять птиц на исследование области пищи аккуратно, перед тем как улетать от области пищи. Эмпирически определены значения

$D_{\min}$ и $D_{\max}$ соответственно они равны -0.1 и 0.1.Значение c_1 и c_2 изменяются от 2.25 до 2.0, выбор этих значений сильно влияет на результаты работы алгоритма. Их значения выбраны эмпирически.

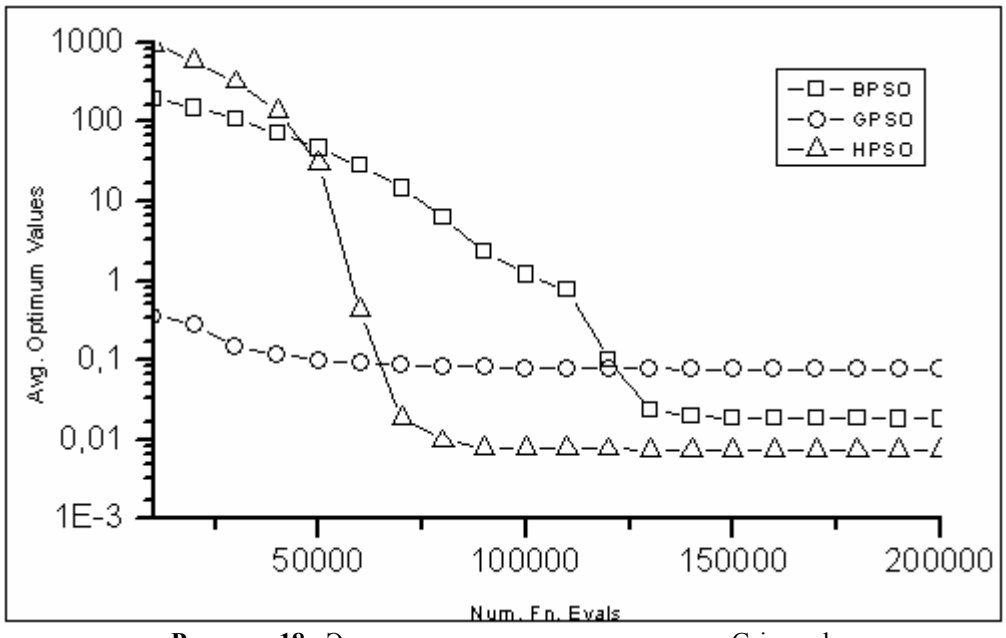
Как в [8-9] число птиц в стае равно 40 (Npop=40), программа заканчивает свою работу после 200000 вычислении функции качества, т.е. число итерации равно 5000 (MaxIter=4000). Для каждой тестовой задачи, программа работает 100 раз, и среднее значение gbest вычисляется, мы так же вычислим среднеквадратичное отклонение, чтобы сравнить стабильность работы алгоритмов.

Экспериментальные результаты

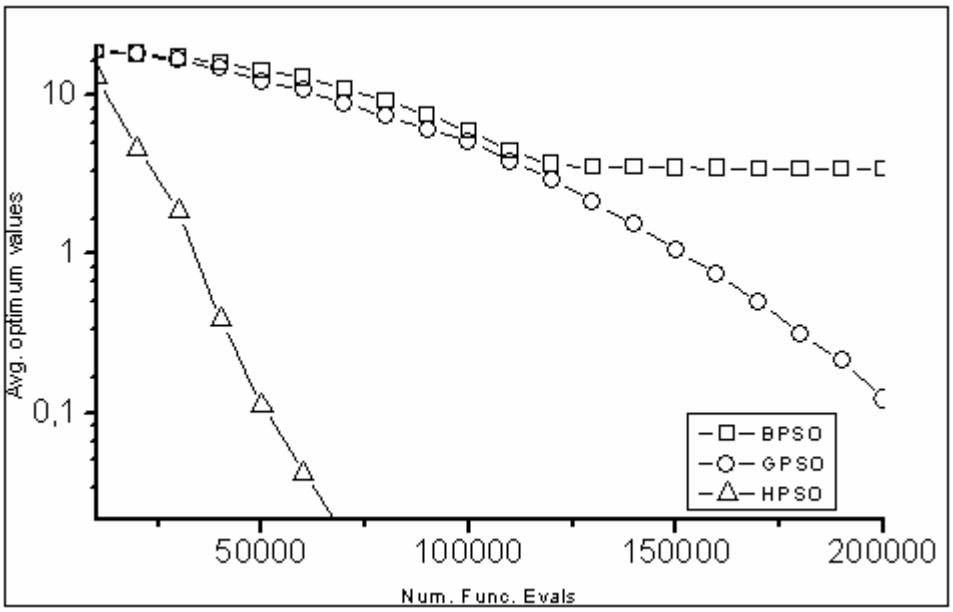
Для сравнения мы посмотрим экспериментальные результаты HPSO алгоритма и стандартного варианта PSO (BPSO) и GPSO (Gregarious PSO [8]) , в соответствие с [8] GPSO даёт лучше результаты сравнительно с другим вариантом PSO алгоритма, поэтому мы выбрали GPSO, чтобы сравнить с нашем алгоритмом. Параметры BPSO выбираются так же как описано в 2.1, а параметры GPSO выбираются так же как описано в [8] .



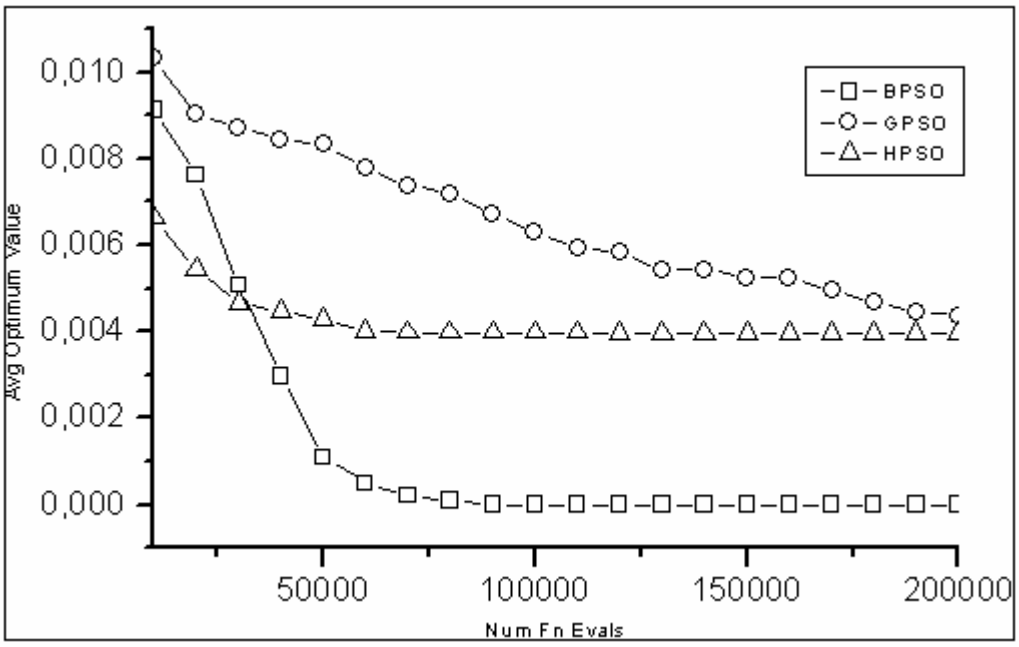
Рисунка 17 : Экспериментальные результаты на Rastrigin



Рисунка 18 : Экспериментальные результаты на Griewank



Рисунка 19: Экспериментальные результаты на Ackley



Рисунка 20 : Экспериментальные результаты на Schaffer's

Рисунки 15–21 показывают экспериментальные результаты на 7 тестах. Для каждого теста, программа работает 100 раз, график иллюстрирует среднее значение gbest в 100 раз выполнении программы. X ось показывает среднее значение gbest, а Y ось показывает число вычисления функции качества.

Экспериментальные результаты на Sphere функции как показано на рисунке 15 показывается, что птицы в HPSO сходятся к решению быстрее, чем алгоритма BPSO, однако в этом простом случае GPSO выполняет лучше, чем HPSO. Аналогичные результаты получаются на Rosenbrock функции, как показано на рисунке 16 , GPSO и HPSO много лучше чем BPSO, тем не менее, GPSO лучше HPSO.

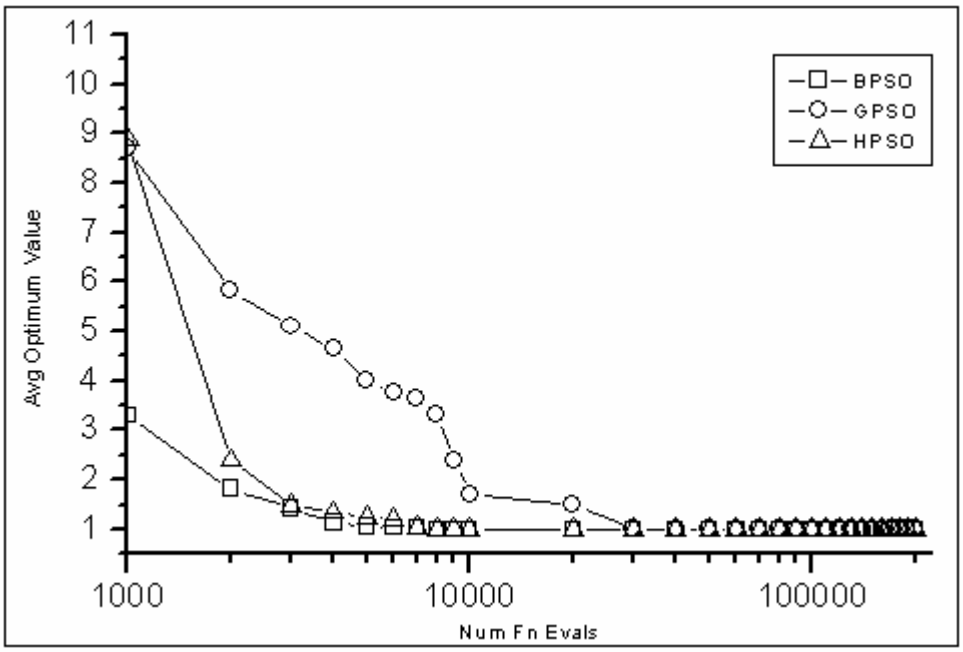
Результаты, показанные на рисунках 17–19, подтверждают то, что HPSO хорошо работает в случаях с мультимодальными функциями. В таких функциях (Rastrigin, Griewank, Ackley), HPSO работает лучше, чем остальные. Отметим важный факт того, что три мультимодальые функции: Rastrigin, Griewank, Ackley являются самыми трудными для

оптимизации среди 7 функций из-за их мультимодальности. В последних случаях (Shekel Foxhole), HPSO, GPSO, BPSO дают аналогичные результаты, однако в тесте с Schaffer функцией, BPSO даёт лучший результат.
Полученные результаты показывают, что алгоритм HPSO даёт лучшие результаты в 6 тестах сравнительно с BPSO. Особенно в тестах с мультимодальными функциями HPSO даёт лучшие результаты относительно и BPSO и GPSO. Чтобы ёще показать хорошие результаты HPSO, в таблице 3 приведены средние gbest значение в 100 раз выполнения программ и среднеквадратическое отклонение. Оно указывается в скобках. Полученные результаты показывают, что HPSO не только даёт лучшие решения, но и является более устойчивым (маленькое среднеквадратическое отклонение) особенно в тестах с мультимодальными функциями. Хотя результаты получены только на маленьком наборе тестов, мы можем заключить, что HPSO работает лучше GPSO и BPSO в почти всех тестах. В следующей части рассмотрим несколько вариантов HPSO путём изменения оценочной функции. Рассмотрим результаты работы этих вариантов на тех же наборах тестов. Обсудим также некоторые недостатки HPSO и направления для их преодоления.

5. Выбор оценочной функции – некоторые варианты HPSO

Улучшение существующих алгоритм является работой учёных компъютерной науки. Этот процесс происходит каждый день и является частью эволюционного процесса алгоритмов. В построение HPSO алгоритма существует несколько недостатков. Например: много параметров. Как выбирать значения для этих параметров не простое дело, так как их выбор зависит от решаемой задачи. В этой части рассмотрим другую проблему, не связанную с выбором параметров, а с тем, что может ухудшать работу HPSO. Хотя в стандартном наборе тестов мы ее не видим, но в прикладной задаче эта проблема может быть серьезной.

В построении HPSO алгоритма лучшая птица в стае получает самую высокую привилегию на обновление. Такая привилегия может привести к риску, когда одна птица берёт на себя полное право на обновление и никогда не отдаёт его остальным. Такая жадность может случиться. Например, на рисунке 22 показывается такой случай. Когда значение Dmax очень маленькое, лучшая птица (в голубой точке) никогда не даёт свою привилегию остальным, и поиск никогда не может найти оптимальное решение в этом случае.

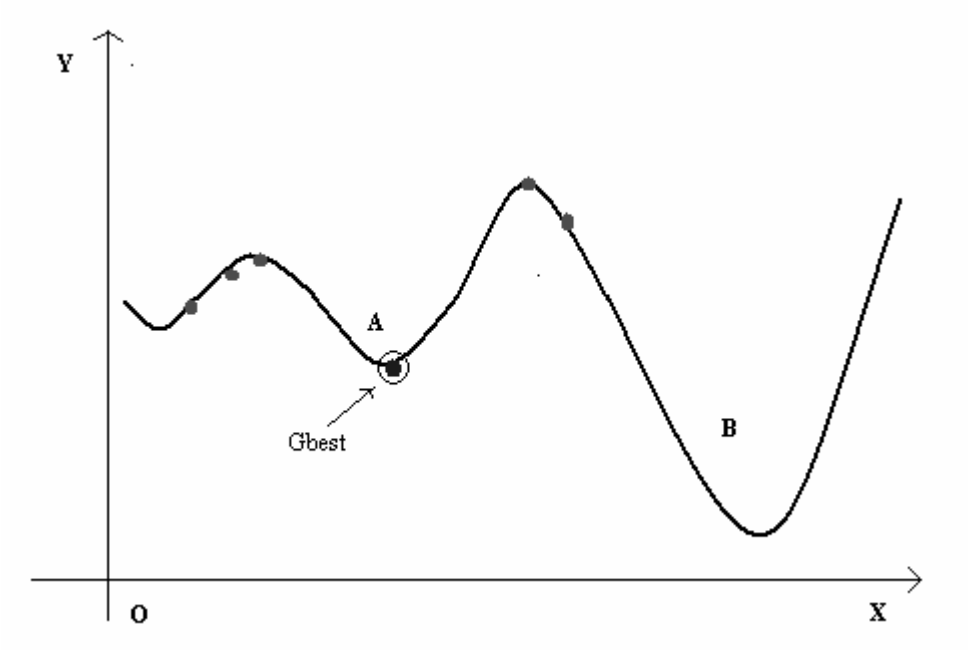


Рисунка 21 : Экспериментальные результаты на Shekel's Foxhole

Table 3: Average minimum values and standard deviation after 200000 function evaluations:

Function	HPSO	GPSO	BPSO
f_1 Sphere	0 (0)	0 (0)	0 (0)
f_2 Rosenbrock	30.63 (29.46)	2.21 (6.79)	119.69 (261.31)
f_3 Rastrigin	0 (0)	0.059 (0.23)	29.41 7.46
f_4 Griewank	0.007 (0.008)	0.059 (0.05)	0.017 (0.019)
f_5 Ackley	1E-7 (1E-7)	0.21 (1.69)	3.22 (7.38)
f_6 Schaffer's	0.004 (0.004)	0.004 (0.004)	0 (0)
f_7 Shekel's Foxhole	0.998004 (0)	0.998004 (0)	0.998004 (0)

Таблица 3: средний минимум и среднеквадратическое отклонение (в скобках) после 200000 вычислении функции качества



Рисунка 22 : одна частица берёт за собой полное право на обновление

Как видно на рисунке 22, лучшая птица (в голубой точке) берёт на себя право на обновление, когда она находится в области "A", и она никогда не отдаст его остальным, которые в это время ждут в высших позициях (красные точки), право на обновление. Из за этого стая не может найти более перспективные решения в области "B".

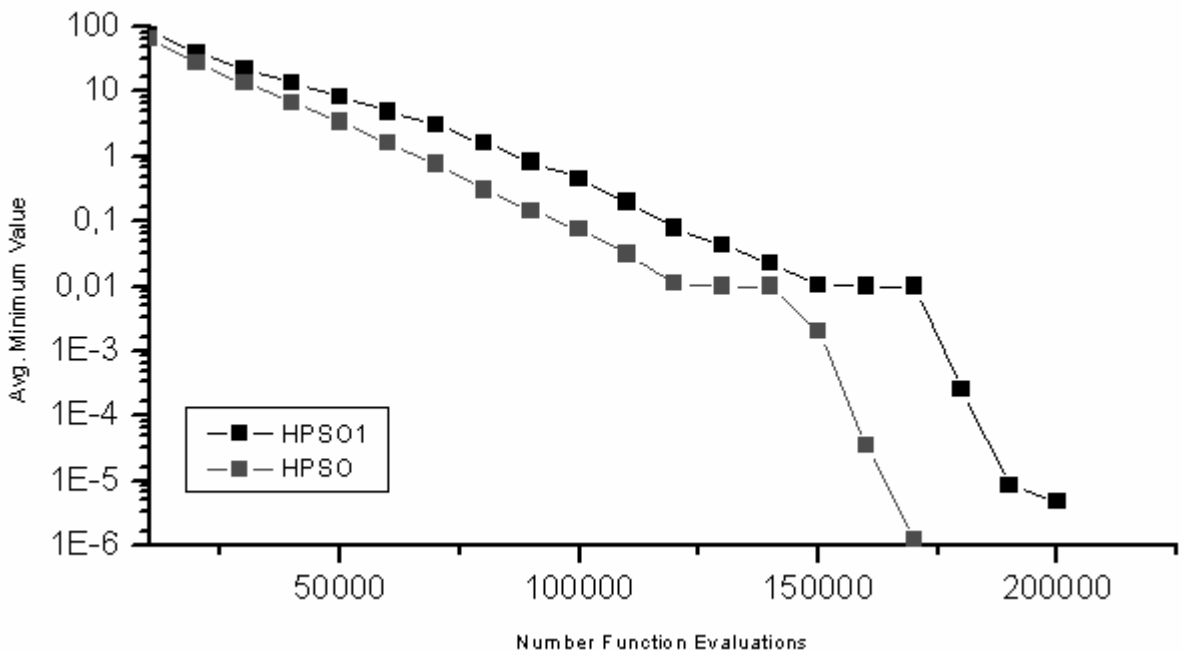
Для решения этой проблемы существует много подходов. В работе предлагается подход, который использовался в традиционном эвристическом поисковом алгоритме, а именно, - добавление к оценочной функции значения "глубина". Как и для традиционного эвристического поискового алгоритма, мы получаем новую формулу вычисления оценочной функции:

$$F_{Evaluation} = F_{fitness} + \alpha * Depth$$

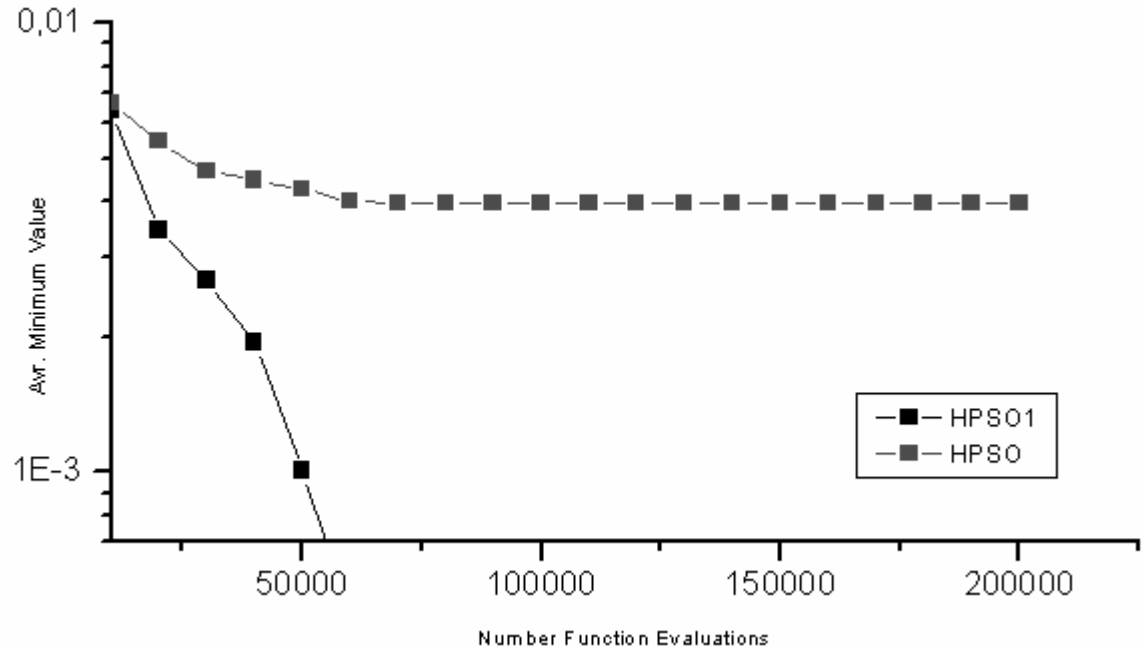
Где *Depth* является глубиной, т.е. сколько раз эта птица имеет права на обновление, α является постоянной.

Function	**Average minimum and standard deviation**	
	α **=0.1**	α **=0.0**
Sphere	0.0(0.0)	0.0(0.0)
Rosenbrock	31.10(31.10)	30.63(29.46)
Rastrigin	0.0000047(0.000046)	0.0(0.0)
Griewank	0.017(0.026)	0.007(0.008)
Ackley	11.41(9.66)	1E-7(1E-7)
Schaffer	0.0(0.0)	0.004(0.004)
Shekel's Foxhole	0.99804(0.0)	0.98804(0.0)

Таблица 4: средний минимум и среднеквадратическое отклонение (в скобках) после 200000 вычислении функции качества.



Рисунка 23 : экспериментальные результаты на Rastrigin



Рисунка 24 : экспериментальные результаты на Schaffer

В таблице 4 показано среднее gbest значение при 100 раз выполнения программ HPSO и среднеквадратическое отклонение (в скобках) в случае α=0.1 и α=0.0 соответственно.
На рисунке 23-24 показывается среднее gbest значение при 100 раз выполнения программ HPSO в случае α=0.1 в задаче Rastrigin и Schaffer соответственно. Как мы видим, результаты HPSO лучше в задаче Schaffer но хуже в задаче Rastrigin сравнительно с α=0.1. В следующей главе эффективность выбора оценочной функции будет исследована более подробно и будут рассмотрены более трудные задачи с очень большими размерностями, в которых выбор оценочной функции сильно влияет на результат работы алгоритмов. Отметим, что результаты на рисунке 23-24 и в таблице 4 получается таким же путём, как эксперименты в разделе 4, т.е. после 100 раз выполнения алгоритмов, и после 200000 вычислении функции качества в каждом случае.

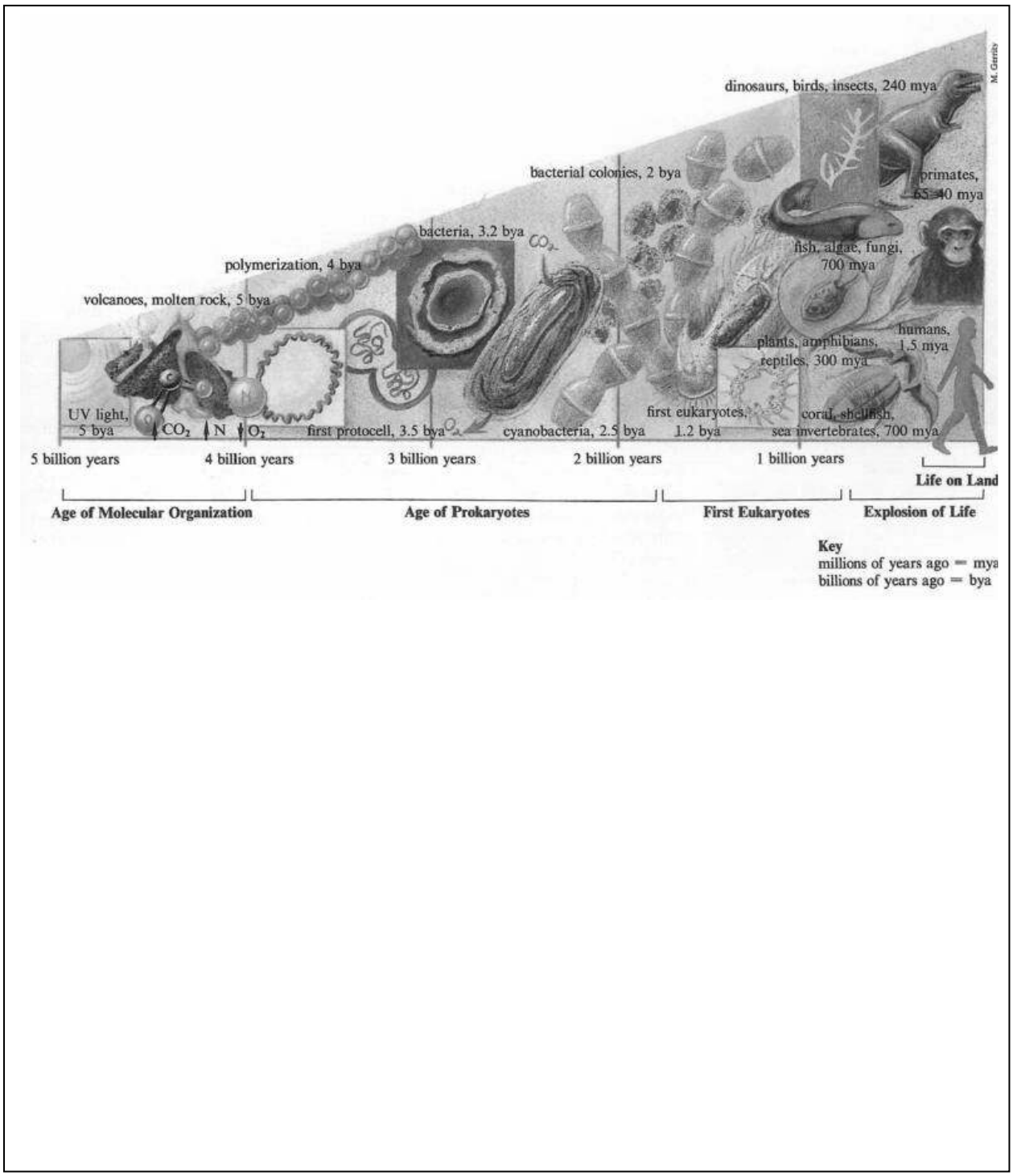
6. Описание программной реализации

Программа была написана на языке C++. Программа запускалась на IBM Regatta P690 Series. Для первого эксперимента было написано три программы, реализующие соответственно BPSO, GPSO и HPSO. Совокупность source code для первого эксперимента находится в папке “Experiment 1”, там находится 3 файла: hpso.cpp, bpso.cpp, gpso.cpp написаны на языке C++, параметры алгоритмов описаны в начале каждой программы.

Глава II

Разработка и исследование HPSO алгоритма для решений задачи кластеризации документов

"*Am I trying to disclaim myself ?*
No, it is a process of evolution"



1. Задача кластеризации документов

1.1 Постановка задачи кластеризации документов

Задача кластеризации документов является одной из рядов задач в области информационного поиска (Information Retrieval), постановка этой задачи аналогична постановкой задачи кластеризации в общем случае. Формулировка задачи кластеризации документов такая:

- Набор документов $D=\{D_1,D_2,\ldots,D_n\}$ $n \in N$
- Заданное число кластеров K
- Найти разделение множества D на K подмножеств (кластеров) $C=\{C_1,C_2,\ldots,C_K\}$ такое что:

+ $C_i \cap C_j = \Theta \quad \forall i \neq j$ i, j=1,2,…,K

+ $\bigcup_{i=1}^{K} C_i = D$

Множество кластеров C называется решением, такая формулировка не говорит о цели кластеризации. Цель кластеризации является теми, что увеличивает различие (т.е. расстояние) между кластерами и увеличивает компактность в каждом кластере. Формально можно оценить качества решения по следующим принципам:

- Максимизировать $\mathrm{Min}_{i,j} \|C_i, C_j\|$
- Минимизировать $\mathrm{Max}_i \|C_i\|$

Где $\|C_i, C_j\|$ является расстоянием между центрами кластеров C_i и C_j, а $\|C_i\|$ является средним расстоянием между каждым документом в кластере C_i до его центра.

Можно классифицировать алгоритмы кластеризации на два типа, алгоритм опирается на оптимизация и не опирается на оптимизация, первой тип например эволюционные алгоритмы [12-16] ищут решение путём оптимизации какой то функции качества, в этом случае можно выбрать функция качества как пара ($\mathrm{Min}_{i,j} \|C_i, C_j\|$, $\mathrm{Max}_i \|C_i\|$) и приводить задача кластеризации к задаче мульти-объектной оптимизации “ Максимизировать $\mathrm{Min}_{i,j} \|C_i, C_j\|$ и Минимизировать $\mathrm{Max}_i \|C_i\|$ ” . Второй тип например K-mean [17], Hierachical [18] ищут решение не путём оптимизации, в этом случае функция качества в первом случае станёт средством для оценивания качества решения. Мы будём обращаться к этой проблеме в следующих частях.

1.2 Векторная презентация документов и модель TF-IDF

Существует несколько моделей презентация документов в виде векторов, в этой части мы посмотрим одна из них TF-IDF модель. Такая модель популярно используется во многих литературах. В соответственно с [19-20] TF-IDF (Term Frequency with Inverse Document Frequency) описается таким образом:

- Каждый j-ой документ в наборе D соответственно вектор $(w_{j1}, w_{2j}, .., w_{j,Nterm})$, где w_i является весей соответственно i-ному терму (слову) в наборе D а *Nterm* является числом различных термов в наборе D.
- Веса w_{ji} вычисляется по формуле:

$$w_{ji} = tf_{ji} * idf_{ji} = tf_{ji} * \log(n / df_{ji})$$

Где tf_{ji} является частотой появления i-ого терма в j-ом документе, а df_{ji} является частотой появления i-ого терма в наборе D, т.е. число документов в наборе D , в которых содержит i-ой терм а n является числом документов в наборе D. Видно что в формуле вычисления весы имеет две части, первая часть понятно а появление второй части (*idf* - Inverse Document Frequency) для того чтобы уменьшать влияние того что когда один терм

очень популярный и появляется на почти всех документах. Чтобы более понятно о TF-IDF модели посмотрим следующий простой пример.

TERM VECTOR MODEL BASED ON $w_i = tf_i * IDF_i$											
Query, Q: "gold silver truck" D_1: "Shipment of gold damaged in a fire" D_2: "Delivery of silver arrived in a silver truck" D_3: "Shipment of gold arrived in a truck" D = 3; IDF = log(D/df_i)											
		Counts, tf_i						Weights, $w_i = tf_i * IDF_i$			
Terms	Q	D_1	D_2	D_3	df_i	D/df_i	IDF_i	Q	D_1	D_2	D_3
a	0	1	1	1	3	3/3 = 1	0	0	0	0	0
arrived	0	0	1	1	2	3/2 = 1.5	0.1761	0	0	0.1761	0.1761
damaged	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
delivery	0	0	1	0	1	3/1 = 3	0.4771	0	0	0.4771	0
fire	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
gold	1	1	0	1	2	3/2 = 1.5	0.1761	0.1761	0.1761	0	0.1761
in	0	1	1	1	3	3/3 = 1	0	0	0	0	0
of	0	1	1	1	3	3/3 = 1	0	0	0	0	0
silver	1	0	2	0	1	3/1 = 3	0.4771	0.4771	0	0.9542	0
shipment	0	1	0	1	2	3/2 = 1.5	0.1761	0	0.1761	0	0.1761
truck	1	0	1	1	2	3/2 = 1.5	0.1761	0.1761	0	0.1761	0.1761

Таблица 5: вычисление весовых векторов документов по схеме TF-IDF

В таблице 5 показывается 4 документа (1 запрос Q и 3 обычных документов D_1, D_2, D_3), в нижней таблице показывается, как вычислить весы векторов по TF-IDF модели, последние 4 столбцы является результатами. Предполагаем, что нам надо оценить какой документ самый подходящий для запросов. Тогда надо вычислить расстояния между запросом и обычными документами. Пусть расстояние между двумя векторами вычисляется по косинусу угла между векторами тогда расстояния между запросом и обычными документами показывается ниже:

$$\text{Cosine } \theta_{D_1} = \frac{Q \bullet D_1}{|Q| * |D_1|} = \frac{0.0310}{0.5382 * 0.7192} = 0.0801$$

$$\text{Cosine } \theta_{D_2} = \frac{Q \bullet D_2}{|Q| * |D_2|} = \frac{0.4862}{0.5382 * 1.0955} = 0.8246$$

$$\text{Cosine } \theta_{D_3} = \frac{Q \bullet D_3}{|Q| * |D_3|} = \frac{0.0620}{0.5382 * 0.3522} = 0.3271$$

$$\therefore \text{Cosine } \theta_{D_i} = \text{Sim}(Q, D_i)$$

$$\therefore \text{Sim}(Q, D_i) = \frac{\sum_i w_{Q,j} w_{i,j}}{\sqrt{\sum_j w_{Q,j}^2} \sqrt{\sum_i w_{i,j}^2}}$$

Тогда документ D_2 является самым походящим для запроса Q.

1.3 Предварительная обработка документов, алгоритм Porter

Документы с начало содержит множество термов, предварительная обработка документов является необходимым, для того чтобы сократить лишние термы. Она и является важным процессом, так как лишние термы отрицательно влияет на результаты кластеризации.

Популярные термы (a, and, the, …) является лишними, поэтому надо удалить такие термы с начало. Термы, у которых однокорни как “action”, “active”, “acting”, “actor”, “actress”, “act” должны приводиться к единственному терму “act” такой процесс называется морфологическим поиском (stemming) и может делать с помощью алгоритма Porter [21]. Краткое описание алгоритма Porter показывается в этой части, чтобы узнать подробно обращаться в [21].

В английском языке “vowel” является буквами “a, o ,i, u, e” и “y” (если перед ней одна “consonant”), остальными является “consonant”. Пусть C обозначается “consonant” а V обозначается “vowel” тогда любое слово в английском языке можно представить в виде:

CVCV...C
CVCV...V
VCVC...C
VCVC...V

В общем случае можно представить в виде: [C](VC){m}[V] , где является натуральным числом. Например:

m=0 TR, EE, TREE, Y, BY.
m=1 TROUBLE, OATS, TREES, IVY.
m=2 TROUBLES, PRIVATE, OATEN, ORRERY.

Для процесса морфологического поиска (stemming), основное поведение является удалением суффиксов слов. Для этого алгоритм определяет несколько правил, в общём случае одно правило имеет вид :

(condition) S1 -> S2

Это правило обозначается что, если слово имеет суффикс S1 и часть слова перед этим суффиксом удовлетворяется условию condition, тогда слово преобразуется в S2, например:

(m > 1) EMENT ->

В этом случае S1 совпадает с ”EMENT” и S2 пустое. В выражении условия condition может содержать ещё:

*S слово оканчивается суффиксом S
v слово содержит “vowel”
*d слово оканчивается двоим количеством “consonant”, пример “-TT” “-SS”
*o слово оканчивается суффиксом CVC, где второй C отличается от W, X, Y пример “-WIL”, “-HOP”

В выражении условия condition может содержать ещё: AND, OR, NOT например следующее правило (m>1 and (*S or *T)) обозначается словом с (m>1) и оканчивается с “T” или “S”. Следующие правило привестит множественные существительное в однокорни:

SSES -> SS
IES -> I
SS -> SS
S ->

В алгоритме Porter определяет наборы правил, с помощью которых процесс морфологического поиска хорошо работает. В этой части мы уже посмотрели основные идеи процесса морфологического поиска так же как алгоритм Porter. Чтобы подробнее об алгоритме Porter и его наборах правил преобразования обращается в [21].

1.4 Метрическое подобие, методы оценивания алгоритма кластеризации

Выше в 1.2 мы посмотрели пример, в котором подобие между документами вычисляется по косинусу угла между двумя векторами презентации этих документов. В этой части мы подробнее определяется подобие двух документов. Подобие двух документов вычисляется по их расстоянию, мы уже знаем про векторную презентацию документов, а как определяет расстояние между ними пока не посмотрели. Самые популярные методы вычисления используется расстояние по Евклиде и косинуса.
Расстояние по Евклиде вычисляется по формуле:

$$\|D_i, D_j\|_{Euclid} = \sqrt{\sum_{k=1}^{n}(D_{ik} - D_{jk})^2}$$

Расстояние по косинусу вычисляется по формуле:

$$Cos(D_i, D_j) = \frac{D_i * D_j}{\| D_i, D_i \|_{Euclid} . \| D_j, D_j \|_{Euclid}}$$

где $\|.\|_{Euclid}$ обозначает расстояние по Евклиде, $D_i * D_j$ является скалярным умножением векторов. Отметим что для расстояния по Евклиде, чем меньше расстояние, чем подобнее документы, а для расстояния по косинусу чем меньше расстояние чем подобнее документы. Таким образом можно вычислять подобие двух документов.
Сейчас мы посмотрим в чём хорошо одного решения и методы оценивания решения. Для оценивания одного решение C (множество кластеров), используется функция критерии (Criterion Function) F(C). Существует много вариантов функции критерии [22], простой случай мы посмотрели в 1.1:

- Максимизировать $\mathrm{Min}_{i,j}\|C_i, C_j\|$
- Минимизировать $\mathrm{Max}_i\|C_i\|$

В этом случае F как мульти-объектная функция. Ещё один простой вариант функции критерии является средним отклонением до центров Je, пусть $C=\{C_1, C_2, \ldots, C_K\}$ является решением, а $\mathrm{Centroid}_i$ соответственно центру i–ого кластера, n_i есть число документов в i–ом кластере, тогда среднее отклонение до центров Je вычисляется по формуле:

$$Je = \frac{\sum_{i=1}^{K}((\sum_{D \in C_i}\| D, Centroid_i \|)/n_i)}{K}$$

Где ||.|| обозначает расстояние между векторами. Чем меньше значение Je чем лучше решение. Кроме этих функции критерии, имеет и другие [22], но в моей работе, поскольку исследовать работы алгоритма PSO в задаче кластеризации, я использовал только функции критерии, которые описаны в этом разделе как в [23-24]. На следующей части мы будем смотреть некоторые алгоритмы решений задач кластеризации.

2. Обзор существующих решений рассматриваемых задач

Алгоритмы кластеризации в общём случае бывает много, в [24] описаны достаточно большой объём таких алгоритмов:

- K-mean 1965
- Fuzzy C-mean 1980-1981
- The Gaussian Expectation Maximization Algorithm 1997
- K-harmonic mean 1999

Чтобы не повторять то, что существует во многих литературах, в этом разделе описано только самый популярный алгоритм кластеризации K-mean, и ещё описываются существенные походы, опирающиеся на PSO. Алгоритм K-mean как обычно является стандартным для сравнения с другими методами, а в нашей работе мы так же сравним работы HPSO с K-mean. Посмотрим описание K-mean на следующем разделе.

2.1 Алгоритм K-mean

K-mean популярно используется во многих работах. Основной идеей алгоритма K-mean является минимизацией квадратичной ошибки (square error function):

$$J=\sum_{l=1}^{K}\sum_{i=1}^{n_i}\| D - Centroid_i \|^2$$

Опирается на необходимые условия для того, чтобы J минимум, строится алгоритма K-mean. Краткое описание алгоритма K-mean для задачи кластеризации документов показывается ниже:

1. Выбирает любые К документы, чтобы составляет К центров кластеров $C_1, C_2, .., C_K$
2. Для каждого документа, вычислим расстояние между ним и центрами, вложит его в кластере, до которого расстояние наименее.
3. Обновляет центры кластеров по формуле:

$$Centroid_{ij} = \frac{\sum_{k=1}^{m_i} D_{kj}}{m_i}$$

где $Centroid_i$ обозначает вектор центра i–ого кластера, m_i обозначает число документов в i–ом кластере.

4. Повторяет 2-3 до того как условие остановки истинно

Алгоритм K-mean описано здесь является локальным поиском (Local Search), он даёт локальный оптимум и быстро сходится, если выбирается хорошие центры в начале. Тем не менее, K-mean иногда не сходится к глобальному оптимуму, результат работы алгоритма сильно зависит от начальной инициализации центров. На рисунке 25 показывается, что алгоритм K-mean не может найти глобальной оптимум (К=3).

Рисунка 25 : Начальные позиции центров сильно влияют на результат работы алгоритма Kmean

На рисунке видно что, если начальные позиции центры кластеров выбираются не точно тогда три кластера всегда как на рисунке 25, даже после миллион раз работы алгоритм K-mean.

2.2 Hybrid K-mean PSO

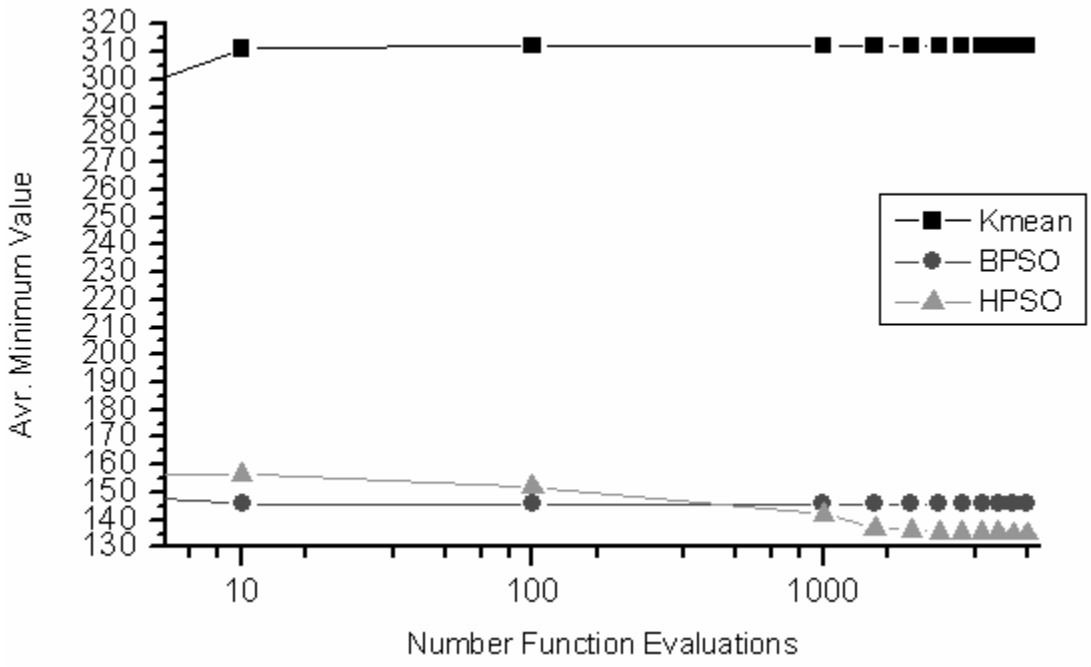
В отличие от других методов, алгоритм PSO для кластеризации является алгоритмом оптимизации. Т.е. алгоритм старается оптимизировать какую ту функцию качества. В качестве функции качества, можно выбирает среднее отклонение до центров Je, которое описано в разделе 1.4. Хотя цель алгоритма K-mean так же является минимизацией квадратичной ошибки, но процесс оптимизации не явно и не гарантирует, что решение алгоритма K-mean является оптимальным решением. Для алгоритмов кластеризации, базирующихся на эволюционные алгоритмы (Генетически алгоритм, Ant colony, PSO...), функция качества явна и не гарантирует, что решение алгоритма оптимально по функции качества.

До настоящего момента, количеств алгоритмов кластеризации, которые опирается на алгоритм PSO, не много. В этом разделе мы посмотрим некоторые из них. В соответственно с [25], алгоритм PSO работает не эффективно в задаче кластеризации. Хотя алгоритм PSO найдёт лучшие решения сравнительно с K-mean алгоритмом, но сходится медленнее. В эксперименте в работе [25], результаты показывается, что K-mean сходится после только 10 итерации, а для сходимости алгоритм PSO должен работает до ста итерации.

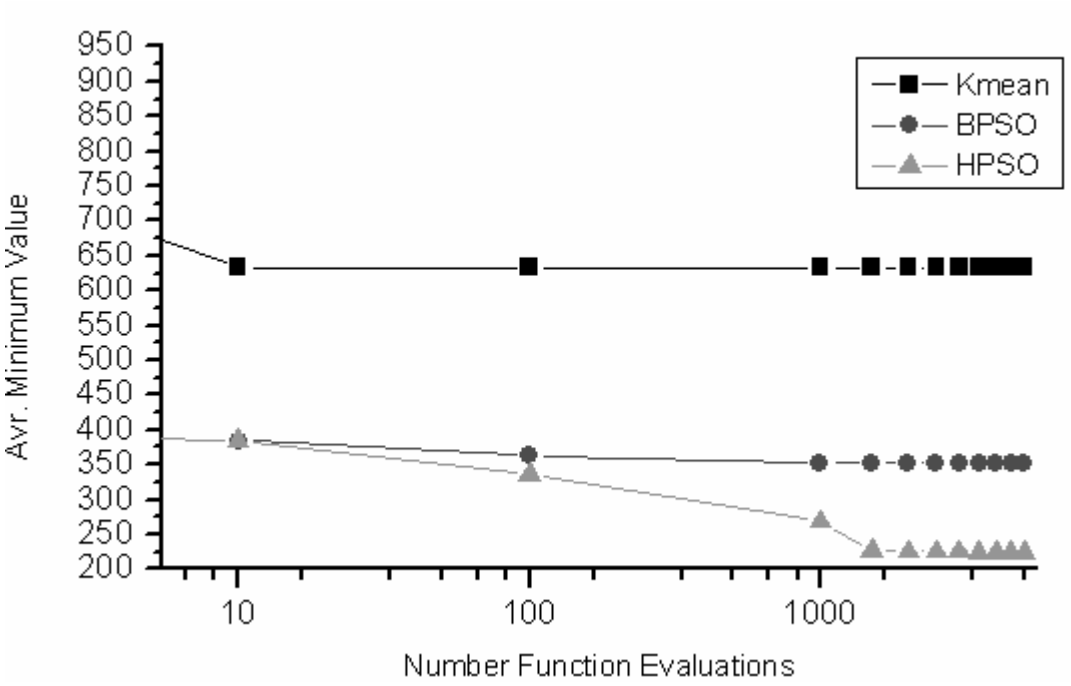
Такой результат не странно, потому что K-mean является локальным поисковым алгоритмом, для него найти локальный оптимум не трудно и очень быстро. Тем не менее, K-mean никогда не может найти глобальные оптимальные решения, если при инициализации, начальные позиции центров не попадают в области вокруг глобального оптимального решения. Отсюда вытекает идея, что составляет иерархическую схему PSO-Kmean. Авторы предполагали, что начало PSO алгоритм работает до определённого числа итерации и потом работает K-mean. Начальные позиции центров для алгоритма в этом случае являются результатом работы алгоритма PSO в предыдущем шаге. В этой иерархической схеме, алгоритм PSO играет роль гайдера, т.е. он ищет области оптимума а процесс точной настройки выполняется K-mean. По [25], иерархическая схема сдаёт лучшие результаты сравнительно с K-mean или PSO.

Data	Source	Num. of Documents	Num. of Term	Num. of Cluster
Fbis	FBIS (TREC)	2463	12674	17
Tr31	TREC	927	10128	7
Tr41	TREC	878	7454	10
Re0	Reuter-21578	1504	2886	13
Ohscal	OHSUMED	11162	11465	4

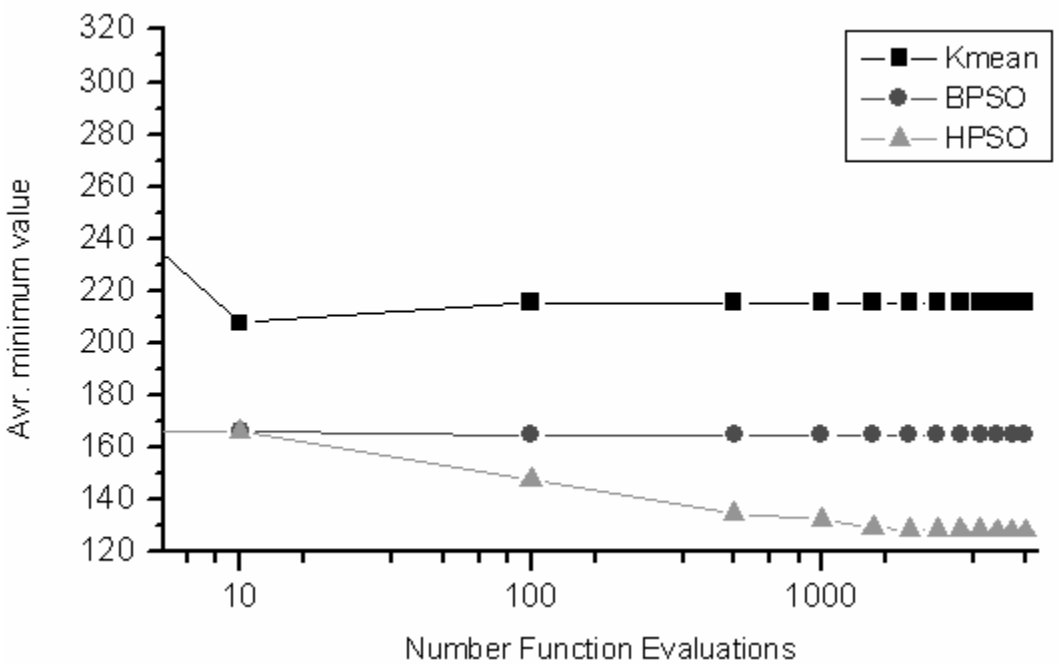
Таблица 6: наборы документов для тестирования



Рисунка 26 : экспериментальные результаты на Fbis



Рисунка 27 : экспериментальные результаты на Tr31



Рисунка 28 : экспериментальные результаты на tr41

3. Разработка и Исследование HPSO для решений задачи кластеризации документов

3.1 Наборы тестов и параметры алгоритма

Наборы тестов описан в таблице 6, число документов в каждом наборе изменяется от 878 до 11162. Число термов, т.е. длина весового вектора изменяется от 7454 до 12674. Эти тесты достаточно труднее тестов в главе 1 для решений, так как размерности поискового пространства очень велики. Все наборы были предтварительно обработаны с помощью алгоритма Porter и удалены все публичные термы.

Набор извлекает с Foreign Broadcast Information Service (TREC-5 [27]). Наборы Tr31, Tr41 извлекает с TREC-6, TREC-7 [27]. Набор Re0 извлекает с Reuters-21578 [28]. Набор Ohscal извлекает с OHSUMED [29] .

Алгоритм HPSO используется в этом разделе почти похож на описанный алгоритм в разделе 3, кроме изменения некоторых параметров. Все параметры описаны на следующем:

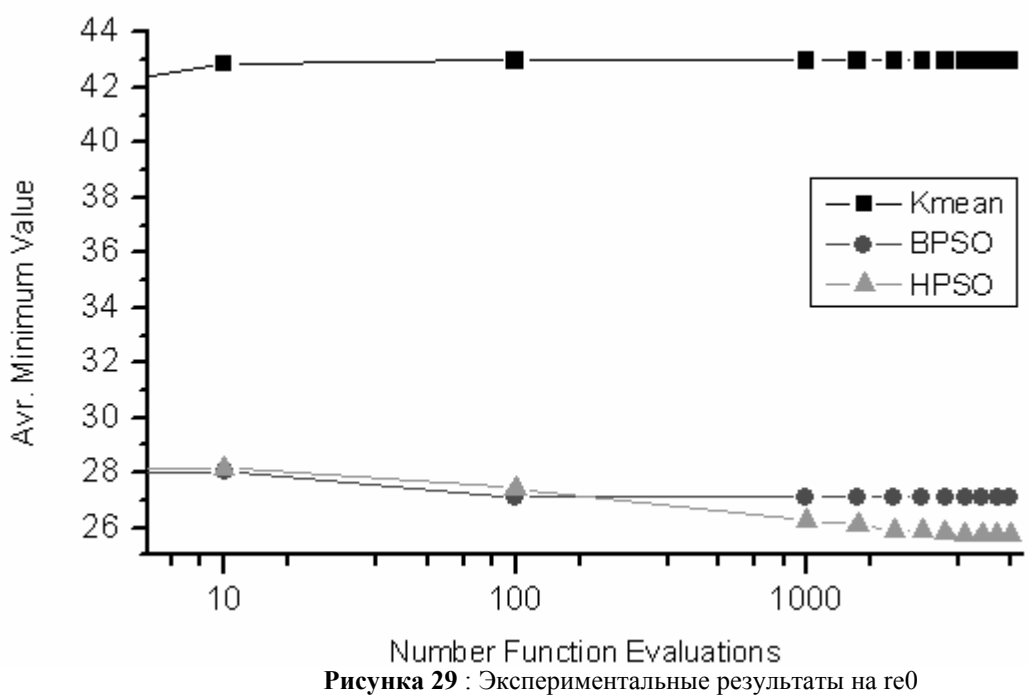
- Npop=50 в место 40, так как размерность поискового пространства велика
- $\lambda = 0.6$ в место 10^{-8}, так как размерность поискового пространства велика
- $D_{\max} = 0.1, D_{\min} = -0.1$
- $c_2 = 2.25$ с начала, и уменьшается до 2.0 в конце
- c_1=1.25 с начала, и уменьшается до 1.0 в конце, так как размерность поискового пространства велика, функция качества имеет много локальных минимумов, использовние большего социального значения может привести к рано сходимости.
- $F_{Evaluation} = F_{Fitness}$
- $F_{fitness}$ совпадается с средним отклонением до центров, т.е. в качестве функция кфчествамы выбираем функцию Je.
- MaxIter=100.

3.2 Экспериментальные результаты и дискуссия

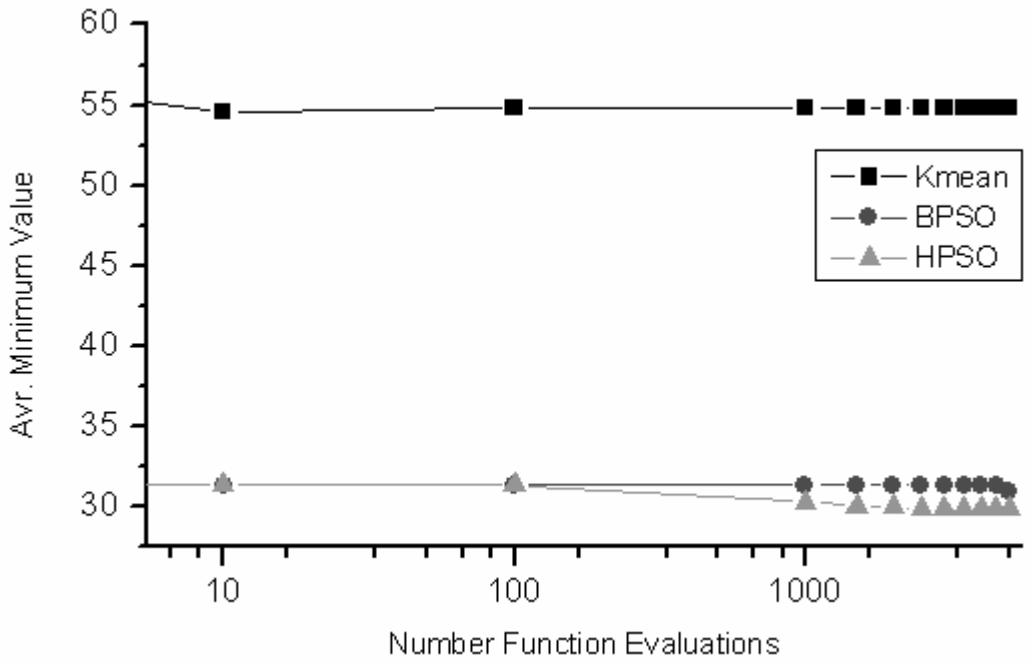
Программа работает 10 раз с случайными начальными позициами частиц. Средние минимумы в каждом итерации вычисляется. Мы выбираем K-mean и стандартный вариант PSO (BPSO) для сравнения результаты работы HPSO. Средние минимумы в 10 раз выполнения программы показаны в рисунке 26-30

Рисунки 26-30 показываются, что HPSO выполняет много лучше чем K-mean и BPSO. Алгоритм K-mean работает очень плохо и иногда значение минимума увеличивает (в tr41, re0). Тем не менее, K-mean сходится очень быстро, как мы видим обычно после 10-20 итерации, K-mean не может работать дальше.

Как мы уже говорили в главе 1, недостаток алгоритма состоит в том, что когда одна частица берёт за собой полное право на обновление и никогда не отдаст остальным это право. Этот случай может случиться в таком задаче, в котором функция качества имеет великая размерность. Для выяснения, как изменяются результаты при использовании другой оценочной функции, в таблице 7 показывается значения минимума, которые получается путём выполнения алгоритма HSPO с разными функциями оценивания.



Рисунка 29 : Экспериментальные результаты на re0



Рисунка 30: Экспериментальные результаты на ohscal

Data	α **= 0.0**	α **= 1.0**
Fbis	134.9	**121.5**
Tr31	224.1	**210.9**
Tr41	128.3	**114.5**
Re0	**25.7**	26.2
Ohscal	29.94	**29.65**

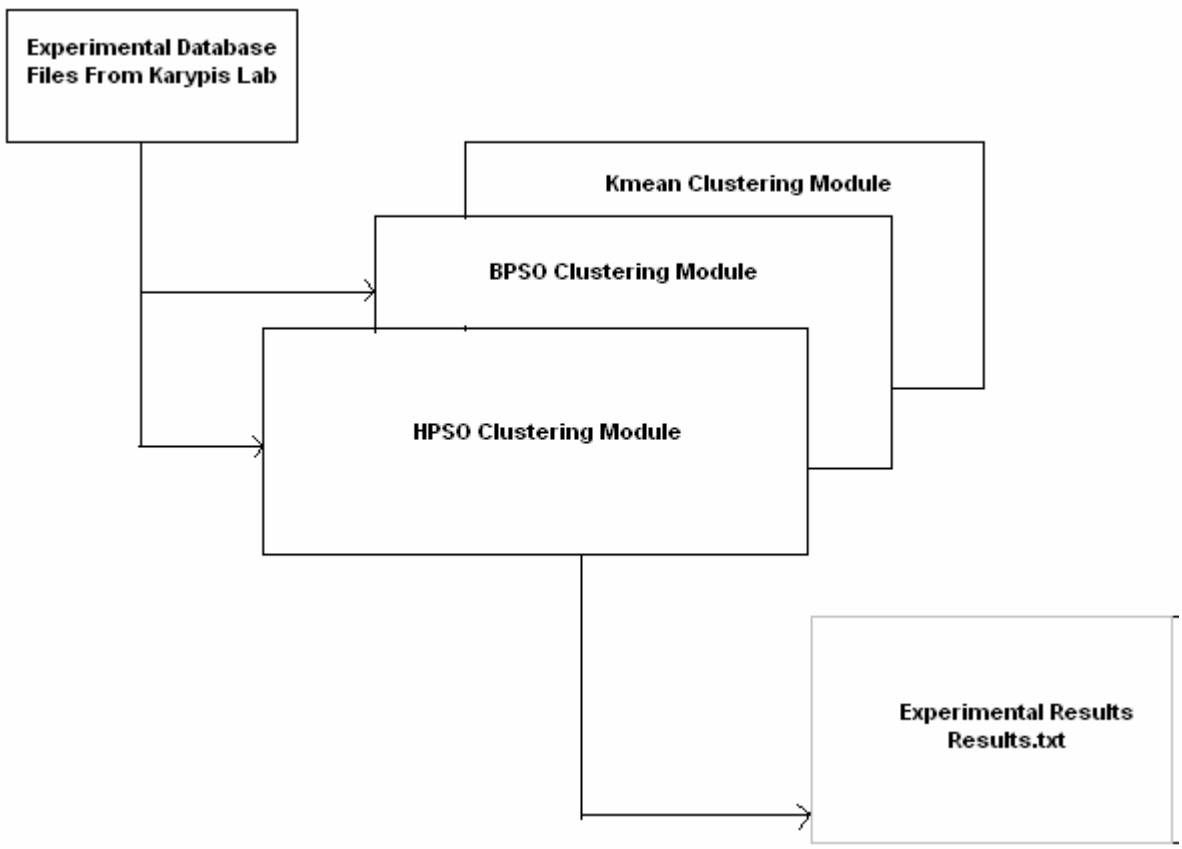
Таблица 7: средние минимума после 100 итерации. Результаты получается на алгоритм HSPO с разными функциями оценивания

В 4/5 тестовых задачах, вариант HPSO с дополнительной глубиной в формуле вычисления оценочной функции ($\alpha = 1.0$) выполняется лучше, чем вариант HPSO, который не использует глубину для оценивания частицы. Такой результат нормальный и мы уже не только раз видели в работах традиционного эвристического поискового алгоритма.

4. Описание программной реализации

Source code содержит 4 файла: hpsoClus.c, psoClus.c, kmean.c, write.c. Все параметры алгоритма описаны в начале каждой программы, для изменения значения должно изменить вручную. Программа write.c используется для генерации начальные позиции частиц для двух программ hpsoClus.c, psoClus.c.

Когда выполняет программу write, нужно добавить параметры “write NumDoc NumClus RunTime Npop”. Где NumDoc количество документов , NumClus количество кластеров, RunTime число повторения работы программы, Npop количество частиц в стае. Результат работы программы write является файлом input.txt, из которых программы hpsoClus.c, psoClus.c. читают начальные позиции частиц. Все программы запускалась в IBM Regatta.



5. Заключение

В рамках дипломной работы исследован вопрос применения алгоритма PSO для решения задачи кластеризации документов. Разработанный алгоритм, опирающийся на эвристический оператор выбора следующих частиц для обновления, представляется перспективным не только для теоретических тестов, но и для решения прикладных задач. Недостатком нового алгоритма является то, что в алгоритме используется много параметров, выбор их значений - не простое дело. В дипломной работе не предлагаются методы выбора параметров, все параметры выбирались эмпирически. В работе исследована основная идея и показано, насколько хорошо работает эта идея, а адаптивный выбор параметров может быть дальнейшим продолжением исследований.

Основные результаты, полученные в дипломной работе.

1. Проведено исследование метода PSO для решения оптимизационных задач. Определены основные параметры метода и возможности дальнейшего его улучшения.

Для выбора параметров предложено использовать:

- эвристический выбор следующих частиц для обновления
- механизм переинициализации векторов скорости для предупреждения ранней сходимости метода

Разработанный алгоритм HPSO работает лучше, чем некоторые варианты PSO на наборах тестов с мультимодальными функциями качества.

2. Разработана программная система, включающая различные варианты алгорита PSO и позволяющая проводить вычислительные эксперименты.
3. Разработан и исследован алгоритм HPSO для задачи кластеризации документов.
4. Исследована оценочная функция для улучшения алгоритма HPSO в задачах с большой размерностью.

Разработанный алгоритм HPSO на тестовых наборах работает лучше, чем базовый вариант PSO и алгоритма K-mean .

Список литература

[1] J. Kennedy and R. C. Eberhart. Particle Swarm Optimization. IEEE Int. Conf. Neural Networks, pages 1942-1948, 1995.

[2] Beville Jean-Iuc Particle Swarm Optimization. Technical report, Miki Lab, Doshisha University Japan 2006

[3] Jakob Vesterstrøm and Jacques Riget, Particle Swarm-Extensions for improved local, multi-modal, and dynamic search in numerical optimization, Master thesis 2002

[4] M. Clerc, and J. Kennedy, The Particle Swarm Explosion, Stability and Convergence in a Multidimensional Complex Space, IEEE Transactions on Evolutionary Computation, Vol. 6, No. 1, February, (2002): 58-73.

[5] T. Krink, J. Vesterstrøm, and J. Riget, Particle Swarm Optimization with Spatial Particle Extension, To appear in: Proceedings of the Congress on Evolutionary Computation, 2002 (CEC-2002).

[6] J. Kennedy. The behavior of particles. 7th Annual Conference on Evolutionary Programming, pages 581-590, 1998

[7] A. Carlisle and G. Dozier. An off-the-shelf PSO. In Proc. Workshop on Particle Swarm Optimization, Indianapolis, IN, 2001.

[8] Srinivas Pasupuleti, Roberto Battiti: The gregarious particle swarm optimizer (G-PSO). GECCO 2006: 67-74

[9] Ratnaweera, A. Halgamuge, S.K. Watson, H.C. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients IEEE transaction on Evolutionary Computation 2004

[10] Michie Heuristic search *The Computer Journal.* 1971; 14: 96-102

[11] Rania Hassan A comparison of Particle Swarm Optimization and Genetic Algorithm 46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference 18 - 21 April 2005, Austin, Texas

[12] Chun-Hung Cheng; Wing-Kin Lee; Kam-Fai Wong A genetic algorithm-based clustering approach for database partitioning Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on Volume 32, Issue 3, Aug 2002 Page(s): 215 - 230

[13] Giosu Lo Bosco, "PGAC: A Parallel Genetic Algorithm for Data Clustering," *camp*, pp. 283-287, Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05), 2005.

[14] Shu-Chuan Chu Constrained Ant Colony Optimization for Data Clustering PRICAI 2004: Trends in Artificial Intelligence

[15] Cheng-Fa Tsai; Han-Chang Wu; Chun-Wei Tsai. A new data clustering approach for data mining in large databases Parallel Architectures, Algorithms and Networks, 2002. I-SPAN apos 02. Proceedings. International Symposium on Volume , Issue , 2002 Page(s):278 – 283

[16] Paterlini, S.; Krink, T. High performance clustering with differential evolution Evolutionary Computation, 2004. CEC2004. Congress on Volume 2, Issue , 19-23 June 2004 Page(s): 2004 - 2011 Vol.2

[17] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", Berkeley, University of California Press, 1:281-297

[18] S. C. Johnson (1967): "Hierarchical Clustering Schemes" Psychometrik*a*, 2:241-254

[19] Salton, G. and McGill, M. J. 1983 *Introduction to modern information retrieval*. McGraw-Hill, ISBN 0070544840.

[20] Salton, G. and Buckley, C. 1988 Term-weighting approaches in automatic text retrieval. *Information Processing & Management* 24(5): 513–523

[21] C.J. van Rijsbergen, S.E. Robertson and M.F. Porter, 1980. *New models in probabilistic information retrieval.* London: British Library. (British Library Research and Development Report, no. 5587).
[22] Ying Zhao and George Karypis. Criterion Functions for Document Clustering: Experiments and Analysis. UMN CS 01-040, 2001.
[23] A Comparison of Document Clustering Techniques. Michael Steinbach, George Karypis and Vipin Kumar. KDD Workshop on Text Mining, 2000.
[24] Mohamed G. H. Orman Particle swarm methods for pattern recognition and image processing PhD Thesis 2005 University of Pretoria.
[25] Xiaohui Cui, Thomas E. Potok Document Clustering using Particle Swarm Optimization IEEE SIS 2005
[26] DW van der Merwe Data Clustering using Particle Swarm Optimization IEEE CEC 2003
[27] TREC. Text REtrieval conference. *http://trec.nist.gov*, 1999.
[28] D. D. Lewis. Reuters-21578 text categorization test collection distribution 1.0. *http://www.research.att.com/_lewis*, 1999.
[29] W. Hersh, C. Buckley, T.J. Leone, and D. Hickam. OHSUMED: An interactive retrieval evaluation and new large test collection for research. In *SIGIR-94*, pages 192–201, 1994.
[30] Hoang Thanh Lam, Popova Nina Nikolaevna, Nguyen Thoi Minh Quan "A Heuristic Particle Swarm Optimization" to-be appeared in the proceedings of SIGEVO GECCO 2007 University of College London