Санкт-Петербургский государственный университет

информационных технологий, механики и оптики

Факультет информационных технологий и программирования­­

Кафедра компьютерных технологий

А. C. Тяхти, А. A. Чебатуркин

## Курсовая работа

Программный комплекс для изучения методов глобальной оптимизации

«GlOpt»

2009

# Оглавление

[Введение 3](#_Toc242452257)

[1. Глобальная оптимизация 3](#_Toc242452258)

[2.Алгоритм имитации отжига 3](#_Toc242452259)

[3. Схемы алгоритма отжига 5](#_Toc242452260)

[3.1. Больцмановский отжиг 5](#_Toc242452261)

[3.2. Отжиг Коши (быстрый отжиг) 6](#_Toc242452262)

[3.3. Сверхбыстрый отжиг 6](#_Toc242452263)

[3.4. Другие схемы алгоритма отжига 7](#_Toc242452264)

[4. Исследуемые задачи 7](#_Toc242452265)

[4.1. Задача о расстановке N ферзей 7](#_Toc242452266)

[4.2. Задача об «Умном муравье» 8](#_Toc242452267)

[5. Структура программного комплекса GLOpt 9](#_Toc242452268)

[5.2 Класс Problem 9](#_Toc242452269)

[5.3 Класс Individual 9](#_Toc242452270)

[5.4 Класс SearchOperator 10](#_Toc242452271)

[5.5 Класс Algorithm 10](#_Toc242452272)

[5.4 Интерфейс IIndividualViewer 10](#_Toc242452273)

[7. Заключение 13](#_Toc242452274)

[Источники 14](#_Toc242452275)

[Исходные коды 15](#_Toc242452276)

# Введение

Целью данной работы являлось изучение методов глобальной оптимизации, создание программного комплекса, позволяющего провести количественные сравнения между различными оптимизационными методами на различных задачах оптимизации.

По схожей тематике известна работа *Соколова Д.О и Давыдова А.А* [7], однако реализованная авторами виртуальная лаборатория на наш взгляд обладает рядом недостатков, которые мы постарались устранить в настоящей работе. Одним из главных требований предъявляемых к программному комплексу являлась гибкость, его дальнейшая расширяемость, позволяющая в будущем наращивать не только набор рассматриваемых алгоритмов решения какой-либо конкретной задачи оптимизации, но и добавлять к рассмотрению новые проблемы, наглядно проводить их сравнительный анализ. В работе основное внимание было сосредоточено на алгоритме имитации отжига, сравнении его с генетическими алгоритмами.

# 1. Глобальная оптимизация

Под термином глобальная оптимизация будем понимать процесс поиска экстремума или экстремумов функции, которая, например, в эволюционном моделировании соответствует приспособленности особи, интерпретируемой как ее способность решать поставленную задачу.

Глобальная оптимизация применима к широкому классу задач. Она находит применение в проблемах проектирования, теории управления физическими процессами, распределении ограниченных ресурсов, анализе данных и других областях.

Известные методы глобальной оптимизации можно разделить на детерминированные и стохастические. Детерминированные методы находят глобальное решение посредством поиска на всем допустимом множестве. Поэтому большинство детерминированных алгоритмов теряют эффективность с возрастанием размерности задачи. Стохастические методы позволяют уйти от проблем детерминированных алгоритмов. Стохастический подход присутствует не только в разработке алгоритма, но и, например, при определении условия остановки.

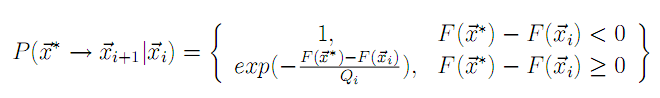
К числу методов глобальной оптимизации относят алгоритм имитации отжига, генетические алгоритмы, метод виртуальных частиц, алгоритм контролируемого случайного поиска.

# 2.Алгоритм имитации отжига

Метод отжига [?], также известен под названиями: метод обжига, метод симуляции отжига, метод модельной закалки, simulated annealing. Этот метод – техника оптимизации, использующая упорядоченный случайный поиск на основе аналогии с процессом образования в веществе кристаллической структуры с минимальной энергией, происходящем при охлаждении этого вещества.

Алгоритм имитации отжига основан на моделировании физического процесса, который происходит при кристаллизации вещества из жидкого состояния в твердое (к примеру, при отжиге металлов). Предполагается, что, во-первых, процесс протекает при понижающейся температуре, а во-вторых, атомы в веществе уже выстроились в кристаллическую решетку, однако переходы отдельных атомов из одной ячейки в другую еще возможны. Вероятность этих переходов в свою очередь обусловлена температурой: чем ниже температура, тем ниже вероятность. Устойчивая кристаллическая структура вещества соответствует минимальному значению энергии. Это значит, что атом либо переходит в состояние с меньшим уровнем энергии, либо остается на месте.

Формализуем данный процесс. Фактически, при моделировании ищется некоторая точка (либо множество точек), на котором достигается минимум некоторой числовой функции *F(x)*. Строится последовательность точек , где  соответствует начальному разделению. При достижении точки  алгоритм завершает свою работу. Пусть рассматривается текущая точка . К ней применяется некоторый оператор *A,* который произвольным образом модифицирует эту точку, в результате чего получается новая точка Вероятность, с которой станет следующей точкой равна , где *P*- распределение Гиббса:



Здесь -элементы произвольной убывающей, сходящейся к нулю последовательности. Она представляет собой аналог понижающейся температуры во время реального физического процесса.

Достоинством метода отжига является свойство избегать "ловушек" в локальных минимумах оптимизируемой функции, и продолжить поиск глобального минимума. Это достигается за счет принятия не только изменений параметров, приводящих к уменьшению значения функции, но и некоторых изменений, увеличивающих ее значение, в зависимости от температуры Т – характеристики моделируемого процесса. Чем выше температура, тем большие "ухудшающие" изменения (аналогичные случайным флуктуациям в нагретом веществе) допустимы, и больше их вероятность. Еще одним достоинством является то, что даже в условиях нехватки вычислительных ресурсов для нахождения глобального минимума, метод отжига, как правило, выдает весьма неплохое решение (один из локальных минимумов). *Л. Ингбером* [?] показано, что метод отжига и его модификации являются одним из наиболее эффективных методов случайного поиска оптимального решения для большого класса задач. Им же проведен сравнительный анализ адаптивного метода отжига (Adaptive Simulated Annealing - ASA) и генетических алгоритмов, из которого следует, что на большинстве задач метод отжига не проигрывает генетическим алгоритмам, а на многих и выигрывает. К настоящему времени разработано множество различных вариантов метода отжига, как общих, так и их специализаций для решения конкретных задач.

Алгоритм имитации отжига можно представить в виде следующей структурной схемы (рис.1).

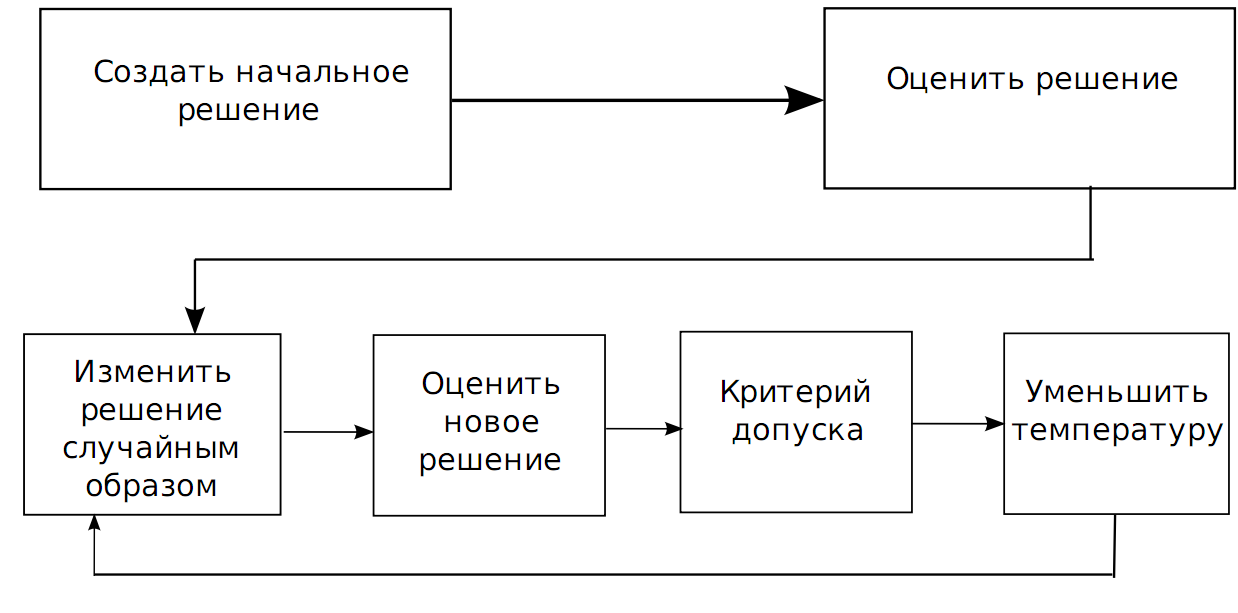
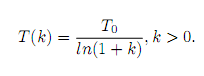


Рис. 1

# 3. Схемы алгоритма отжига

## 3.1. Больцмановский отжиг

Исторически первой схемой метода отжига является схема Больцмановского отжига. Эта схема использовалась *Н.Метрополисом* для вычисления многомерных интегралов пути в задачах статистической физики, а также *С.Киркпатриком* для решения задачи нахождения оптимальной разводки микросхем. В Больцмановском отжиге изменение температуры задается формулой



Семейство распределений Q(x; T) выбирается как семейство нормальных  
распределений с математическим ожиданием и дисперсией, т.е. задается  
плотностью



где D - размерность пространства состояний. Пространство состояний предполагается метрическим. Для больцмановской схемы доказано, что при достаточно больших **** и общем числе шагов k, выбор такого семейства распределений гарантирует нахождение глобального минимума. Основным недостатком метода является медленное убывание температуры.

## 3.2. Отжиг Коши (быстрый отжиг)

В результате совершенствования больцмановского метода *Цу* и *Хартли* предложили алгоритм, использующий другую схему изменения температуры



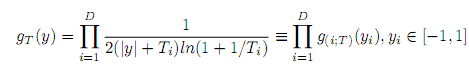
В качестве Q используются нормированные распределения Коши с плотностью



Однако это распределение сложно моделировать в пространствах размерностью больше единицы. Кроме того, в них накладывается ряд ограничений на закон изменения температуры, что является серьезным недостатком данного алгоритма.

## 3.3. Сверхбыстрый отжиг

Недостатки двух предыдущих методов привели к тому, что в 1989 году  
американским исследователем *Л. Ингбером* был разработан метод сверхбыстрого отжига (Very Fast Annealing). В нем пространство S считается состоящим из D-мерных векторов где . Кроме этого,  
температура по каждой из координат может различаться, таким образом,  
T также является вектором размерности D. Семейство распределений  
строится следующим образом. Вводится функция:



В качестве y для получения плотности распределений используется  
.  вычисляется по формуле , где - случайная величина с плотностью *g(i;T)* на [-1;1].Такую случайную величину легко промоделировать

,



где - независимые случайные величины, равномерно распределенные на [0,1]. Доказано, что закон изменения температуры



дает статистическую гарантию нахождения глобального минимума.  как правило управляется двумя параметрами:



Преимущества такого метода очевидны. Экспоненциальное убывание температуры гораздо быстрее достижимого в предыдущих методах. Разделение размерностей может дать большой выигрыш, как благодаря отдельным температурам (т.к. специфика конкретной задачи может сильно различать параметры), так и благодаря ускорению процесса, в случае, если не нужно менять все координаты одновременно. Кроме того, в отличие от отжига Коши, сверхбыстрый отжиг допускает очень быстрое моделирование распределения независимо от размерности пространства. Недостатком данного метода является сложность его настройки для решения конкртеной задачи ввиду большого количества параметром.

## 3.4. Другие схемы алгоритма отжига

Известны и другие алгоритмы отжига. Например, алгоритм *Ксин Яо*, полученный повторением метода сверхбыстрого отжига. Вполне возможны задачи, где этот метод показывает лучшую производительность, чем предыдущий рассмотренный, однако на практике метод Ксин Яо зачастую выдает результат худший, чем при сверхбыстром отжиге.

В тех случаях, когда достаточно не глобального оптимального решения, а лишь близкого к нему, либо при отсутствии достаточных вычислительных ресурсов находят свое применение так называемые методы «тушения». Их идея заключена в комбинировании семейства распределений одного из вышеописанных методов с более быстрым законом изменения температуры.

# 4. Исследуемые задачи

## 4.1. Задача о расстановке N ферзей

В качестве примера использования алгоритма отжига рассмотрим задачу о расстановке N шахматных ферзей на доске размером N\*N. Ферзи должны быть расставлены так, чтобы ни один из них не бил другого. Это означает, что ни на одной вертикали, горизонтали или диагонали не могут находиться два ферзя одновременно. Иными словами, функция коллизий должна принимать минимальное значение. Пример решения задачи для размерности N = 8 показан на рисунке 2.

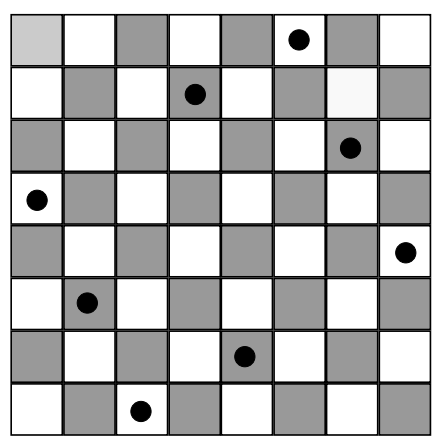


Рис. 2

Данная задача в течение многих лет решалась известными математиками, такими как Гаусс, однако оптимальное решение найдено не было. Задача может решаться непосредственно - алгоритмом полного перебора с возвратом, однако это не позволяет рассматривать данную задачу при достаточно больших N. На обычном персональном компьютере за разумное время может быть получен результат лишь для N менее 15. В то же время с помощью метода имитации отжига задача может достаточно эффективно решаться для значительно большей размерности, однако при достаточно больших N решение может содержать коллизии.

## 4.2. Задача об «Умном муравье»

Действие происходит на поверхности тора размером 32x32 клетки. В некоторых клетках находится еда. Муравей начинает движение из клетки, помеченной «Start».

За ход муравей может выполнить следующие действия:

* повернуть налево;
* повернуть направо;
* сделать шаг вперед и, если в новой клетке есть еда, съесть ее;
* ничего не делать.

Игра длится 200 ходов. Цель игры - создать муравья «с минимальным числом состояний», который за минимальное число ходов съест как можно больше яблок.

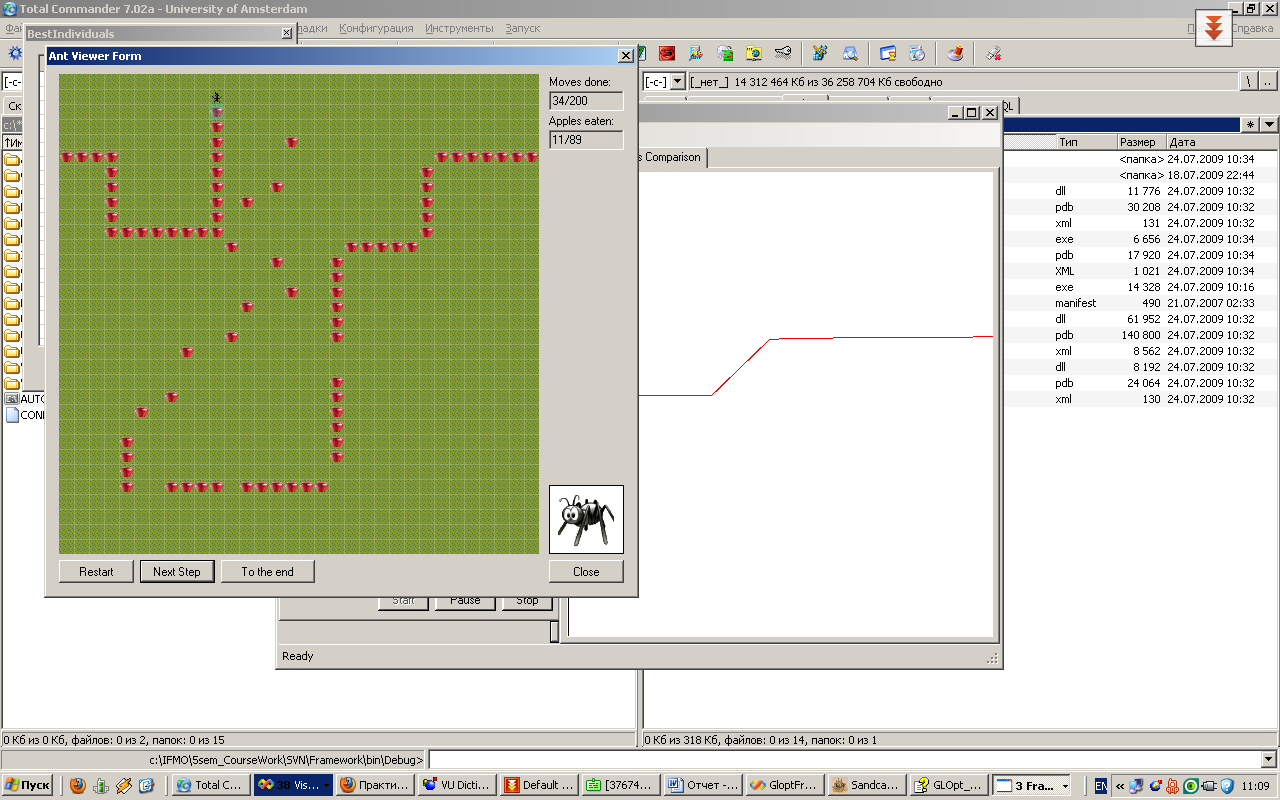


Рис. 3 Визуализатор особи в задаче об умном муравье комплекса GLOpt

# 5. Структура программного комплекса GLOpt

Важнейшим критерием при создании программного комплекса являлось удобство и легкость его расширения на максимальное количество задач оптимизации. Реализация этой задачи потребовала разработки специальной программной структуры, которая бы полностью соответствовала предъявляемым требованиям к масштабируемости, а также была бы довольно простой для понимания, позволяя желающим внедрять и анализировать методы оптимизации, не вошедшие в рамки данной работы.

Основные классы фреймворка GLOpt, реализованного на языке С# под платформой Microsoft.NET, представлены на рисунке 4. Управляющий класс Brain ответственен за загрузку основных сущностей программного комплекса: Problem, Optimizer, ISearchOperator, а также осуществляет контроль над текущими процессами. Сама же задача глобальной оптимизации полностью описывается абстрактными классами Problem, Algorithm, Individual, SearchOperator, от которых в свою очередь наследуются классы, реализующие конкретные задачи, например, задачу «умного муравья» или же метод имитации отжига.

## 5.2 Класс Problem

Абстрактный класс Problem содержит метод OptimizationDirection, возвращающий информацию о направления оптимизации (min, max) и метод EvaluateIndividual, служащий для вычисления целевой fitness-функции у конкретной особи. Наследники этого класс должны реализовывать эти методы, а также методы обеспечивающие доступ к базовой информации о задаче: названии, ее кратком описании.

public abstract class Problem : Plugin

{

public abstract OptimizationDirection OptimizationDirection { get; }

public abstract double EvaluateIndividual(Individual individual);

}

## 5.3 Класс Individual

Абстрактный класс Individual реализует представление индивидуальной для каждой задачи оценочной характеристики Fitness, описывает метод сравнения индивидов между собой.

public abstract class Individual : IComparable<Individual>

{

public double Fitness

{

…

}

#region IComparable<Individual> Members

public int CompareTo(Individual other)

{

return Fitness.CompareTo(other.Fitness);

}

#endregion

}

## 5.4 Класс SearchOperator

Класс SearchOperator необходим для организации требуемых в задаче операций над объектами класса Individual.Так, например, в задаче «умного муравья» это методы мутации индивида и скрещивания индивидов между собой (кроссовер), а в методе имитации отжига – изменение уже найденного решения в соответствии с выбранным вероятностным распределением. Классы наследники должны определять следующие методы.

* Create – создание новой особи.
* Modify – изменение особи каким-либо образом (например случайно), с учетом интерфейса, который требует Algorithm.

public abstract class SearchOperator : Plugin, ICreateOperator

{

#region ICreateOperator Members

public abstract Individual Create(Random r);

#endregion

}

## 5.5 Класс Algorithm

Класс Algorithm содержит следующие методы:

* OptimizationDirection – направление оптимизации, характеризаующее сам алгоритм (направление по умолчанию).
* SearchOperator – набор методов для работы с особями, которые использует алгоритм.
* Initialize – установка необходимых для работы алгоритма параметров.
* NextIteration – метод, описывающий шаг итерации алгоритма.
* Terminated – метод, позволяющий определить - закончил ли алгоритм работу. Если метод возвращает true, то алгоритм автоматически перезапускается.

public abstract class Algorithm : Plugin

{

protected internal abstract OptimizationDirection OptimizationDirection { get; }

protected internal SearchOperator SearchOperator {get; internal set; }

protected internal virtual List<Individual> CurrentIndividuals { get;

protected set; }

protected internal virtual Individual BestIndividual { get; set; }

protected internal virtual void Initialize(){}

protected internal virtual void NextIteration(){}

protected internal virtual bool Terminated() {}

}

## 5.4 Интерфейс IIndividualViewer

Данный интерфейс необходим для создания возможности отображать решения конкретной задачи визуально. Для визуализации решения также должен быть подготовлен необходимый набор форм.

public interface IIndividualViewer

{

void ViewIndividual(Individual individual);

}

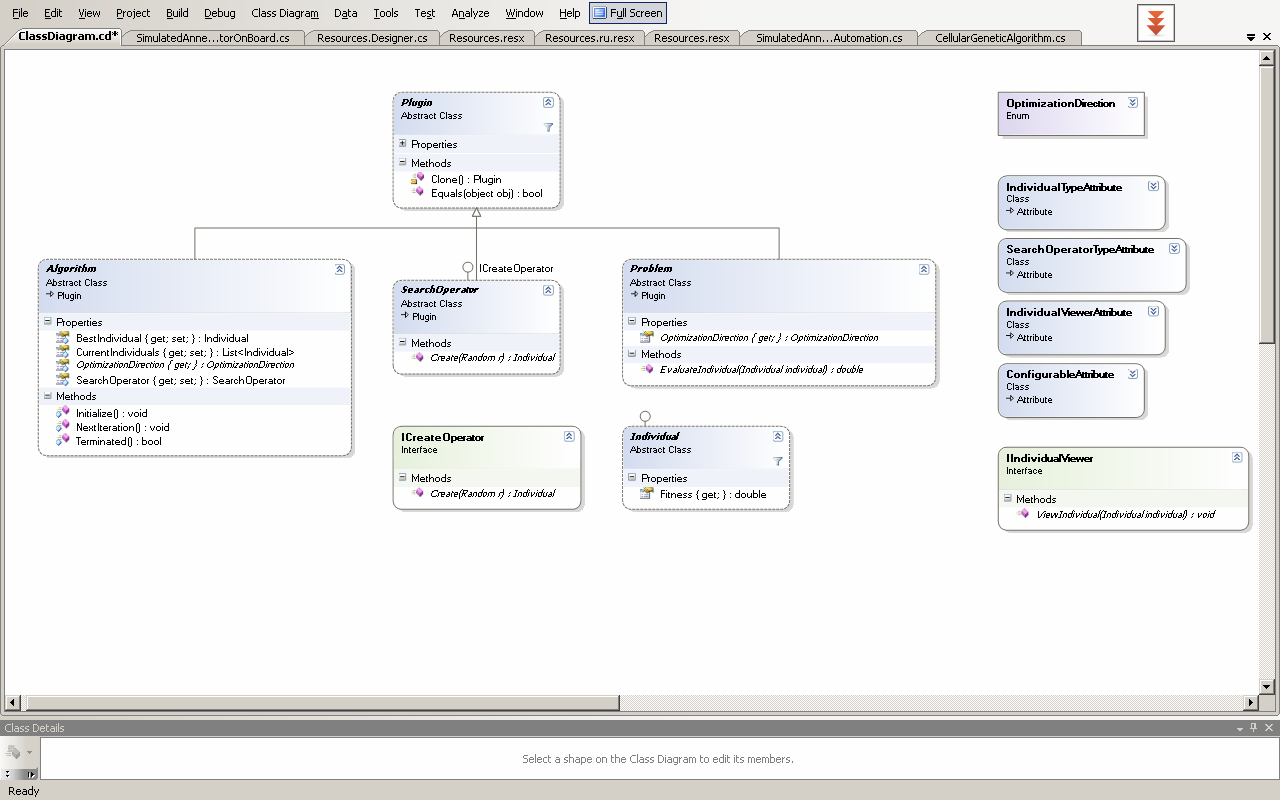


Рис.4

**6. Инструкция по разработке модулей GLOpt**

Программный комплекс GLOpt разрабатывался как средство анализа и сравнения различных методов оптимизации на конкретных задачах. В систему закладывалась возможность добавления новых алгоритмов и рассматриваемых задач как самими авторами на различных этапах разработки, так и другими студентами – с помощью подключения отдельных модулей (плагинов).

Проектирование модулей предполагает изучение структуры уже реализованных методов оптимизации, и постарении на их основе собственных. Ниже приведены базовые рекомендации, которым необходимо следовать при разработке собственных модулей, определяющих алгоритм решения.

1. Реализуйте алгоритм поиска оптимального решения – класс, наследованный от абстрактного класса Algorithm. Данный класс должен определять следующие методы.
   * Title – возвращает стоку с названием реализуемого алгоритма.
   * Description – возвращает строку с кратким описанием алгоритма.
   * OptimizationDirection – возвращает одну из двух именованных констант: OptimizationDirection.Minimize или OptimizationDirection.Maximize.
   * Initialize - описывает начальное состояние в алгоритме поиска решения.
   * NextIteration – описывает действия, происходящие на очередной итерации алгоритма.
   * В переменной BestIndividual типа Individual должна содержаться актуальная информация об объекте типа Individual c лучшей на данной итерации алгоритма целевой функцией.

Методы Title и Description в свою очередь являются абстрактными методами класса Plugin, от которого наследуется класс Algorithm.

1. Реализуйте интерфейс SearchOperator, наследованный от интерфейса ICreateOperator. В данном интерфейсе должен быть определен метод Create, возвращающий объект типа Individual (абстрактный класс, определяется для конкретной задачи). Также в SearchOperator’е реализуются все необходимые методы для работы с объектами Individual – например, операции мутации и кроссовера.
2. Необходимо удостовериться что библиотека \*.dll откомпилированного модуля находится в папке plugins проекта Framework.

Для реализации новой задачи в комплексе GLOpt необходимо сделать следующее.

1. Реализовать класс - наследник от абстрактного класса Individual, определяющего объект к которому в дальнейшем применяется алгоритм оптимизации. Так, в задачи о ферзях таким объектом выступает шахматная доска с расставленными на ней ферзями.
2. Реализовать класс – наследник от абстрактного класса Problem. В данном классе должны быть определены методы Title, Description, OptimizationDirecrtion, аналогичные методам, описанным в разделе рекомендаций по реализации алгоритма. Также необходимо определить метод EvaluateIndividual, возвращающий значение типа double – значение целевой функции для данного индивида.
3. Для реализации компоненты визуализации текущего решения требуется определить класс Viewer, наследуемый от класса IndividualViewer, и в частности определить метод ViewIndividual, вызывающий графическую форму, либо представляющий данные о решении в любом другом удобном для пользователя виде.
4. Необходимо удостовериться что библиотека \*.dll откомпилированного модуля находится в папке plugins проекта Framework.

К программному комплексу GLOpt прилагается пакет технической документации, призванный упростить разработку собственных модулей задач и алгоритмов, а также понимание архитектурных принципов самого модуля. Документация получена с помощью генератора документации компании Microsoft – Sandcastle.

# 7. Заключение

Реализованный программный комплекс позволяет анализировать эффективность тех или иных методов глобальной оптимизации на любых задачах, к которым применима оптимизация.

Возможным является добавление в среду как собственных задач для рассмотрения, алгоритмов их решения, так и реализация визуализаторов для соответствующих задач и методов.

В ходе сравнения эффективности различных методов оптимизации для решения уже существующих в среде задач были получены следующие результаты. Для решения задачи об умном муравье наиболее эффективным оказался простой клеточный алгоритм, показавший чуть более высокую фитнесс-функцию, чем клеточный. Метод отжига показал на этой задаче существенно более низкие результаты. Для решения же задачи о расстановке ферзей при разных установках параметров наиболее эффективными оказались метод отжига и клеточный генетический алгоритм. Простой генетический алгоритм при любых настройках показывал худшие результаты. Однако следует понимать, что в данном случае сильна зависимость результатов от конкретной реализации алгоритмов.

В будущих версиях планируется проработать систему документации и подсказок, облегчающая понимание пользователями уже существующих решений и упрощающая реализацию собственных плагинов.

# Источники

1. *Ingber A.L.* Simulating Annealing: Practice versus theory ­-

Mathl. Comput. Modelling, 1993

1. *Елкин Д. И. Тяхти А. C.* Метод отжига - СПбГУ ИТМО, 2008, http://rain.ifmo.ru/cat/view.php/theory/unsorted/ai-annealing-2008/article.pdf
2. *Бедный Ю.Д., Шалыто А.А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». http://is.ifmo.ru/works/\_ant.pdf
3. *Джонс М. Т.* Программирование искусственного интеллекта в приложениях – М.: ДМК-Пресс, 2004
4. *Лопатин А.* Методы отжига. Электронный конспект – *Крысталь Б.*

www.cs-seminar.spb.ru/reports/52.pdf

1. *Орлянская И.В* Современные подходы к построению методов глобальной оптимизации. http://zhurnal.ape.relarn.ru/articles/2002/189.pdf
2. *Соколов Д.О., Давыдов А.А., Царев Ф.Н., Шалыто А.А.* Виртуальная лаборатория обучения генетическому программированию для генерации управляющих конечных автоматов. – МК МГУ. М.: МАКС Пресс, 2008, с. 179 – 183. http://is.ifmo.ru/works/\_2\_93\_davidov\_sokolov.pdf

# Исходные коды

Ознакомиться с исходными кодами можно в архиве, прилагаемом к данной работе. Файловая структура программного комплекса выглядит следующим образом:

C:.

│ GloptFramework.sln

│

├───ArtificialAnt – задача об «умном муравье»

│ ├───AntViewer

│ │ AntViewerForm.cs

│ │ AntViewerForm.Designer.cs

│ │ FieldControl.cs

│ │ FieldControl.Designer.cs

│ │

│ ├───IndividualInfo – описание индивида в задаче

│ │ Automation.cs

│ │

│ ├───ProblemInfo – представление задачи

│ │ SimpleAntProblem.cs

│ │

│ └───\_Internal

│ ├───Mover

│ │ IMover.cs

│ │ SimpleMover.cs

│ │

│ └───Phenotype

│ AbstractAnt.cs

│ Ant.cs

│ SimpleAnt.cs

│

├───CellularGenetic – клеточный генетический алгоритм

│ CellularGeneticAlgorithm.cs

│

├───Charts

│ ChartControl2D.cs

│ IChart.cs

│ SimpleChart.cs

│ SurfaceChart.cs

│

├───**Framework**

│ Program.cs

│

├───Genetic – генетический алгоритм

│ │ GeneticOptimizationAlgorithm.cs

│ │

│ └───Operators

│ GeneticSearchOperator.cs

│ GeneticSearchOperatorOnAutomation.cs – реализация |алгоритма с использованием особи на базе автомата

│ GeneticSearchOperatorOnBoard.cs – реализация |алгоритма с использованием особи на базе шахматной доски

│ IGeneticSearchOperator.cs

│ SearchOperatorOnBoard.cs

│

├───NewBrain – управляющие модули программного комплекса

│ │ AlgorithmHistory.cs

│ │ Brain.cs

│ │ ClassDiagram1.cd

│ │ ConfigurationManager.cs

│ │

│ ├───Algorithms

│ │ AlgorithmRunner.cs

│ │ ASOPair.cs

│ │ ASOPairEditor.cs

│ │

│ ├───Attributes

│ │ ConfigurableAttribute.cs

│ │ IndividualTypeAttribute.cs

│ │ SearchOperatorTypeAttribute.cs

│ │

│ ├───ComponentModel

│ │ IndividualViewer.cs

│ │

│ ├───Exceptions

│ │ TypeAttributeException.cs

│ │

│ ├───Plugins

│ │ Algorithm.cs

│ │ CustomProperties.cs

│ │ ICreateOperator.cs

│ │ Individual.cs

│ │ OptimizationDirection.cs

│ │ Plugin.cs

│ │ PluginManager.cs

│ │ Problem.cs

│ │ SearchOperator.cs

│ │

│ └───UI

│ ├───Controls

│ └───Forms

├───QueensPlacement – задача о расстановке N ферзей

│ ├───IndividualInfo – представление индивида в задаче

│ │ Board.cs

│ │

│ ├───ProblemInfo

│ │ QueensPlacementProblem.cs

│ │

│ └───QueenViewer – визуализатор индивида на базе шахматной доски

│ FieldControl.cs

│ FieldControl.Designer.cs

│ QueenViewerForm.cs

│ QueenViewerForm.Designer.cs

│

├───SimpleGeneticOptimizer – простой генетический алгоритм

│ SimpleGeneticOptimizationAlgorithm.cs

│

├───SimulatedAnnealing – метод имитации отжига

│ GaussianRandom.cs

│ ISimulatedAnnealingOperator.cs

│ SimulatedAnnealingAlgorithm.cs

│ SimulatedAnnealingOperatorOnAutomation.cs

│ SimulatedAnnealingOperatorOnBoard.cs

│

└───Utils

├───ComponentModel

│ EditorHelper.cs

│ SimpleTypeDescriptorContext.cs

│

└───Drawing

MatrixHelper.cs

RectangleHelper.cs