Лабораторная работа № 1

Алгоритм отжига

Цель работы. Разработка и исследование алгоритма отжига в процессе решения числовой задачи оптимизации.

Домашнее задание

1. Охарактеризуйте основные этапы алгоритма отжига на примере решения классической задачи размещения N ферзей на шахматной доске размером таким образом, чтобы ни один ферзь не угрожал другому.
2. Какими способами в зависимости от сложности решаемой проблемы производится оптимизация алгоритма отжига?
3. Укажите классы задач, в которых использование алгоритма отжига может быть эффективным?

Краткие сведения из теории

Метод оптимизации называемый методом отжига иначе принято называть симуляцией восстановления. Логично полагать, что в данном конкретном случае процесс восстановления моделируется при помощи метода поиска.

Если рассматривать механизм восстановления на примере охлаждения растаявшей субстанции, то основным моментом в данном конкретном случае будет особенность структуры новой замороженной субстанции, которая однозначно будет отличаться от структуры субстанции имевшейся в замороженном виде ранее ввиду того, что спонтанное охлаждение и будет обуславливать таковую структуру. К примеру, субстанция охлажденная скачкообразно имеет слабую структуру. При представлении данного метода *структура представляет собой кодированное решение*

Отжигом называется метод оптимизации, моделирующий физический процесс охлаждения растаявшей субстанции. Алгоритм отжига может быть разделен на пять этапов (рис.1):

- создание начального решения и его кодировка;
- оценка решения;
- случайный поиск решения;
- использование критерия допуска;
- понижение температуры.

Создание начального решения для большинства проблем является случайным. На первом шаге алгоритма начальное решение помещается в текущее решение. Кодировка решения осуществляется набором переменных. Например, в задаче коммивояжера каждый элемент кодировки означает город. Оценка решения состоит из декодировки текущего решения и выполнения нужного действия. Поиск решения начинается с копирования текущего решения в рабочее решение, которое произвольно модифицируется путем перестановки двух элементов кодировки. На этом этапе работы алгоритма есть два решения:

оригинальное, которое называется текущим, а второе – найденное решение, которое называется рабочим.

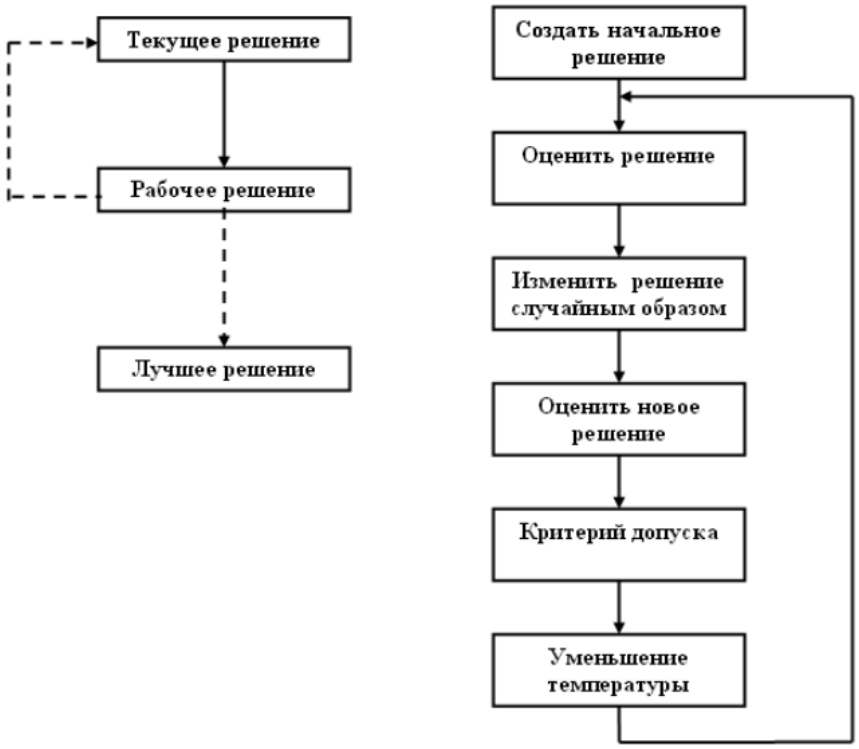


Рис. 1.

С каждым решением связана определенная энергия, представляющая его эффективность. Будем полагать, что чем ниже энергия, тем больше его эффективность. При сравнении двух решений может оказаться, что рабочее решение имеет меньшую энергию, чем текущее решение. Тогда в алгоритме рабочее решение необходимо копировать в текущее решение и осуществить переход к этапу понижения температуры. Если рабочее решение окажется хуже текущего решения, то следует определить критерий допуска. Вероятность допуска основывается на уравнениях теории термодинамики и определяется по формуле:

$$P(\delta E) = \exp(-\delta E/T)$$

где δE - дельта энергии, T – температура.

Вычисления вероятности допуска по этой формуле показывают (рис.2), что при температурах свыше 60º С плохие решения принимаются чаще, чем отбрасываются. Однако, если уровень энергии меньше, вероятность принятия лучшего решения выше. При снижении температуры вероятность принятия худшего решения также снижается.

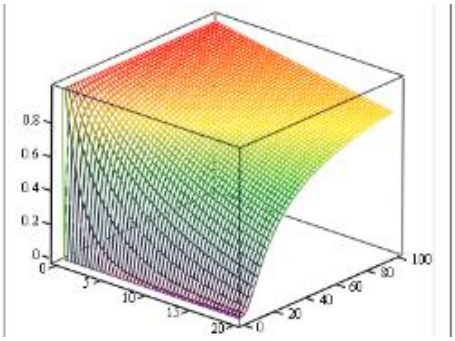


Рис.2.

При снижении температуры вероятность принятия лучшего решения также повышается, при этом диапазон поиска решения сужается до решения проблемы при нулевой температуре. Среди множества вариантов снижения температуры используем простую функцию

$T_{i+1} = \alpha T_i$

где T_{i+1}, T_i - значения температур на предыдущей i-ой и следующей за ней i +1 - ой итерации; α - константа, имеющая значение меньше единицы. Согласно алгоритму отжига при одной температуре выполняется несколько итераций. После завершения итераций температуру следует понизить. Вычислительный процесс продолжается пока температура не достигнет нуля.

Для демонстрации алгоритма отжига используем пример решения задачи размещенияN ферзей на шахматной доске таким образом, чтобы ни один ферзь не угрожал другому. Кодировка задачи о N ферзях стандартна. Так как каждый столбец шахматной доски содержит только одного ферзя, то для отображения решения используется массив из N элементов. В элементах этого массива хранятся строчные индексы положения ферзя. Пример кодировки конечного решения приведен на рис. 3.

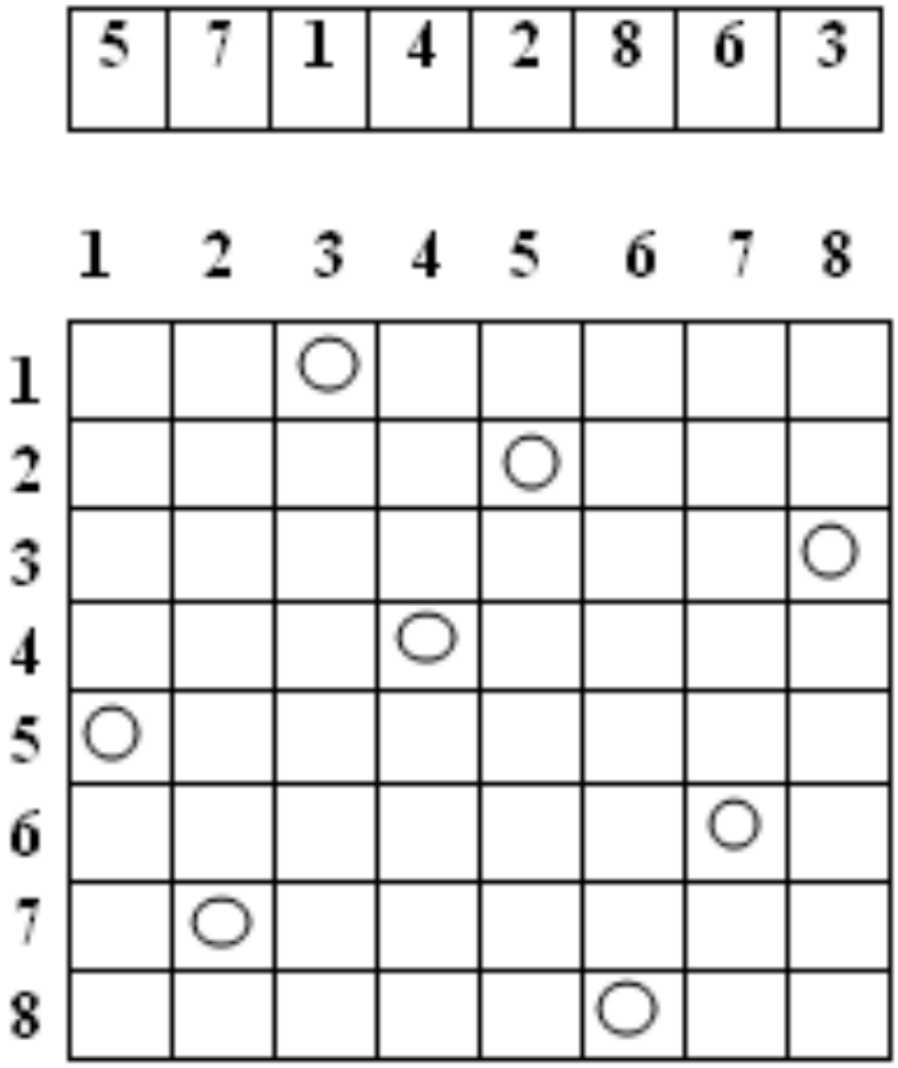


Рис.3.

Из рисунка видно, что первой цифрой в кодировке является число 5, соответствующее строке первого столбца, в которую помещен ферзь, второй цифрой - соответственно число 7, соответствующее строке второго столбца, в которую помещен другой ферзь и т.д. Энергия решения определяется как количество конфликтов, которые возникают в кодировке. Задача решена, когда найдена кодировка, при которой энергия равна нулю, то есть на шахматной доске не существует конфликтов.

Для изучения исходного кода, реализующего алгоритм отжига, используем константы и типы данных, приведенные в листинге 1.

Начало листинга 1. . Типы данных и константы.

```
uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls;

const Max_Length=8;
```

```
INITIAL_TEMPERATURE=30.0;

FINAL_TEMPERATURE=0.5;

ALPHA=0.98;

STEPS_PER_CHANGE=100;

type

SolutionType = array [0..Max_Length-1] of integer;

MemberType=record

Solution:SolutionType;

energy:real;

end;
```

Конец листинга 1

Массив SolutionType – это кодировка задачи о ферзях. Символьная константа Max_Length определяет размер доски (в данном случае решается задача о восьми ферзях). Решение хранится в структуре MemberType, которая также включает энергию, определяемую как количество конфликтов.

Константы INITIAL_TEMPERATURE и FINAL_TEMPERATURE задают границы изменения температуры соответственно 30º C и 0,5º C. В расчете константа ALPHA принимает значение 0,98. Константа STEPS_PER_CHANGE устанавливает количество итераций после каждого изменения температуры. Число итераций в расчете принимается 100.

В листинге 2 содержатся функции инициализации кодировки и поиска нового решения. Они используются для создания начального решения и его произвольного изменения.

Начало листинга 2 Инициализация и функции поиска.

```
procedure tweakSolution(var Member:MemberType);

var temp,x,y:integer;

begin

x:=random(Max_Length);

repeat

y:=random(Max_Length);

Until x<>y;

temp:=Member.Solution[x];
```

```
Member.Solution[x]:=Member.Solution[y];

Member.Solution[y]:=temp;

end;

procedure InitializeSolution(var Member:MemberType);

var i:integer;

begin

// Начальная инициализация решения

for i:=0 to Max_Length-1 do

Member.Solution[i]:=i;

// Изменение решения случайным образом

for i:=1 to Max_Length-1 do

TweakSolution(Member);

end;
```

Конец листинга 2

Функция InitializeSolution создает решение, при котором все ферзи помещаются на доску. Для каждого ферзя задаются идентичные индексы строки и столбца. Это обозначает отсутствие конфликтов по горизонтали и вертикали. Затем решение изменяется при помощи функции TweakSolution. Позднее функция TweakSolution используется в алгоритме, чтобы изменить рабочее решение, выведенное из текущего решения. Оценка решения выполняется с помощью функции ComputeEnergy. Эта функция, как показано на листинге 3, идентифицирует все конфликты, которые существуют для текущего решения

Начало листинга 3. Оценка решения.

```
procedure ComputeEnergy(Member:MemberType);

const dx:array [0..3] of integer=(-1,1,-1,1);

dy:array [0..3] of integer=(-1,1,1,-1);

var i,j,x,y,tempx,tempy:integer;

board:array[0..Max_Length-1,0..Max_Length-1] of char;

conflicts:integer;

begin

for i:=0 to Max_length-1 do

board[i,Member.Solution[i]]:='Q';
```

```
//Считает количество конфликтов для каждого ферзя

conflicts:=0;

for i:=0 to Max_Length-1 do begin

x:=i;

y:=Member.Solution[i];

//Замечание: по условию кодировки конфликты по вертикали и горизонтали исключены

//Проверяем диагонали

for j:=0 to 3 do begin

Tempx:=x;

Tempy:=y;

While True do begin

Tempx:=Tempx+dx[j];

Tempy:=Tempy+dy[j];

if (TempX<0)or(TempX>=Max_Length)or

(TempY<0)or(TempY>=Max_Length) then break;

if board[TempX,TempY]='Q' then inc(conflicts);

end;

end;

end;

Member.energy:=conflicts;

end;
```

Конец листинга

Из данного листинга видно, что положение ферзей на доске определяется значениями x и y. Диапазоны dx и dy при этом используются для расчета положения на доске каждого ферзя. Начальные значения dx = -1 и dy= -1 соответствуют перемещению ферзя на северо-запад, а конечные значения dx = 1 и dy = - 1 отвечают перемещению на северо-восток. При этом расчет количества конфликтов для каждого ферзя на доске по всем четырем диагоналям производится поочередно. Если другой ферзь найден, значение переменной конфликта увеличивается. После завершения поиска конфликты загружаются в структуру с решением в качестве значения энергии.

В листинге 4 используется функция для копирования одного решения в другое.

Начало листинга 4. Копирование одного решения в другое

```
procedure CopySolution(var dest,src:MemberType);

var i:integer;

begin

for i:=0 to Max_Length-1 do

dest.Solution[i]:=src.Solution[i];

dest.energy:=src.energy;

end;
```

Конец листинга

Листинг 4. Копирование одного решения в другое.

Здесь следует напомнить, что решение кодируется и сохраняется в структуре MemberType. Функция CopySolution копирует содержимое одной структуры MemberType в другую. Последняя вспомогательная функция приведена в листинге 5. Это функция emitSolution, которая распечатывает представление доски из закодированного решения и выдает его.

Начало листинга 5. Отображение решения в виде шахматной доски.

```
procedure emitSolution(var Member:MemberType);

var board:array[0..Max_Length-1,0..Max_Length-1] of char;

x,y:integer;

begin

For x:=0 to Max_Length-1 do

board[x,Member.Solution[x]]:='Q';

Form1.Label1.Caption:='board:';

//SetLength(Form1.Memo1.Lines,Max_Length);

// for y:=0 to Max_Length-1 do

//SetLength(Form1.Memo1.Lines[y],Max_Length);

for y:=0 to Max_Length-1 do begin

for x:=0 to Max_Length-1 do

if board[x,y]='Q' then Form1.Memo1.Text:=Form1.Memo1.Text+'Q'
```

```
else Form1.Memo1.Text:=Form1.Memo1.Text+'*';

Form1.Memo1.Text:=Form1.Memo1.Text+#13+#10;

end;

end;
```

Конец листинга

При помощи функции emitSolution шахматная доска печатается на основе кодировки (строка – это индекс, а содержимое – это столбец). Затем доска отображается, причем Q соответствует ферзю, а «#» - пустой клетке доски.

После изучения всех вспомогательных функций в листинге 6 представлен непосредственно алгоритм отжига с помощью функции TForm1.Button1Click.

Начало листинга 6. Алгоритм отжига.

```
TForm1 = class(TForm)

Memo1: TMemo;

Label1: TLabel;

Button1: TButton;

Memo2: TMemo;

procedure Button1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);

var Timer:integer;

Solution:Integer;
```

```
Step,useNew,accepted:integer;

temperature,test,delta,calc:real;

current,working,best:memberType;

fp:textFile;

begin

Timer:=0;

Solution:=0;

temperature:=INITIAL_TEMPERATURE;

AssignFile(fp,'stats.txt');

rewrite(fp);

randomize;

InitializeSolution(current);

computeEnergy(current);

best.energy:=100.0;

CopySolution(working,current);

while Temperature>FINAL_TEMPERATURE do begin

Form1.Memo2.Lines.Add('Temperature:'+FloatToStr(Temperature));

accepted:=0;

//Изменены решения случайным образом

for step:=0 to STEPS_PER_CHANGE-1 do begin

useNew:=0;

tweakSolution(working);

ComputeEnergy(working);

if working.energy<=current.energy then useNew:=1

else begin

test:=random; //?

delta:=working.energy-current.energy;
```

```
calc:=exp(-delta/Temperature);

if calc>test then begin inc(accepted); useNew:=1; end;

end;

if useNew=1 then begin

useNew:=0;

CopySolution(current,working);

if Current.energy<best.energy then begin

CopySolution(best,current);

Solution:=1;

end;

end

else CopySolution(working,current);

end;

timer:=timer+1;

writeln(fp,timer,Temperature:8:3,best.energy,accepted:10);

Memo2.Lines.Add('Best Energy ='+FloatToStr(best.energy));

temperature:=temperature*ALPHA;

end;

CloseFile(fp);

if Solution=1 Then emitSolution(best);

end;

end.
```

Конец листинга

Данный листинг программы практически повторяет основные этапы алгоритма отжига. После открытия файла для вывода лога и инициализации генератора случайных чисел происходит инициализация текущего решения InitializeSolution с переменной current и оценка решения с помощью функции ComputeEnergy. Текущее решение копируется в рабочее решение, и алгоритм запускается. Внешний цикл алгоритма выполняется до тех пор, пока текущая температура не станет меньше конечной температуры или сравняется с ней. Это позволяет избежать использования нулевой температуры при расчете вероятности допуска. Внутренний цикл алгоритма работает по методу Монте-Карло. Он выполняет ряд итераций при текущей температуре с целью полного изучения

возможностей поиска при данной температуре. На первом шаге происходит изменение рабочего решения с помощью функции TweakSolution. Затем рассчитывается энергия рабочего решения, которая сравнивается с текущим решением. Если энергия нового рабочего решения меньше или равна энергии текущего решения, рабочее решение принимается по умолчанию. В противном случае рассчитывается критерий допуска, и определяется будет ли выбрано худшее решение. Дельта энергии рассчитывается как разность между рабочей энергией и текущей. Это означает, что энергия рабочего решения больше, чем энергия текущего решения. В данном примере просто генерируется случайное число в интервале от нуля до единицы, которое затем сравнивается со значением критерия допуска. Если в результате сравнения критерий допуска больше случайного значения, то рабочее решение принимается. Затем рабочее решение необходимо скопировать в текущее решение, так как переменная working, в которой на данный момент хранится рабочее решение, повторно будет изменена при следующей итерации внутреннего цикла. Если рабочее решение не было принято, текущее решение копируется поверх рабочего. При следующей итерации старое рабочее решение удаляется, программа изменяет текущее решение и пробует снова. После вывода статистической информации в лог-файл, выполнив требуемое количество итераций внутреннего цикла, температуру необходимо снизить. Понижение температуры происходит умножением текущей температуры на константу ALPHA и внешний цикл повторяется. Если решение было найдено, то оно хранится в переменной best. Об этом в конце алгоритма сигнализирует переменная Solution. Она устанавливается во внутреннем цикле, после того как было определено, что обнаружено решение, энергия которого меньше энергии текущего решения best.

области применения:

Метод отжига может быть эффективным при решении задач различных классов, требующих оптимизации. Ниже приводится их краткий список:

• создание пути;

а реконструкция изображения;

D назначение задач и планирование;

Q размещение сети

• глобальная маршрутизация;

• обнаружение и распознавание визуальных объектов;

• разработка специальных цифровых фильтров.

Поскольку метод отжига представляет собой процесс генерации случайных чисел, поиск решения с использованием данного алгоритма может занять значительное время. В некоторых случаях алгоритм вообще не находит решение или выбирает не самое оптимальное.

Рабочее задание

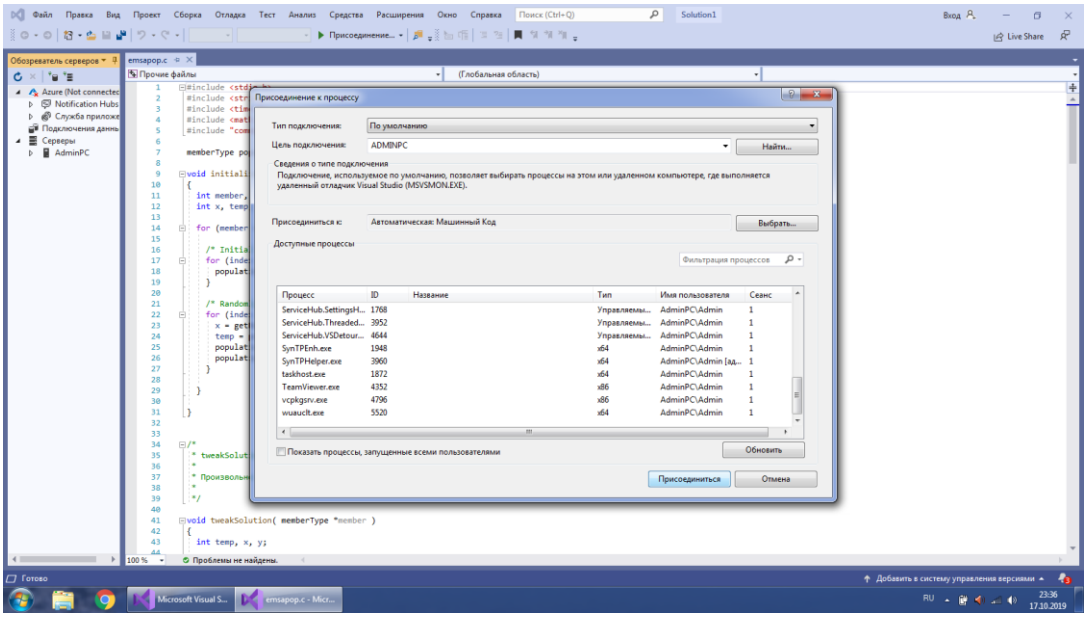
1. Рассмотреть исходный код алгоритма отжига.
2. Составить полный листинг программы решения задачи N ферзей, используя листинги 1- 6.
3. На основе пункта 2 рабочего задания создать программу - исполняемый файл, который может загружаться и выполняться под управлением Windows.
4. Привести пример решения задачи о 8 ферзях.
5. Выполнить оптимизацию алгоритма отжига путем изменения начальной температуры, конечной температуры, числа итераций при одном значении температуры, значения коэффициента .

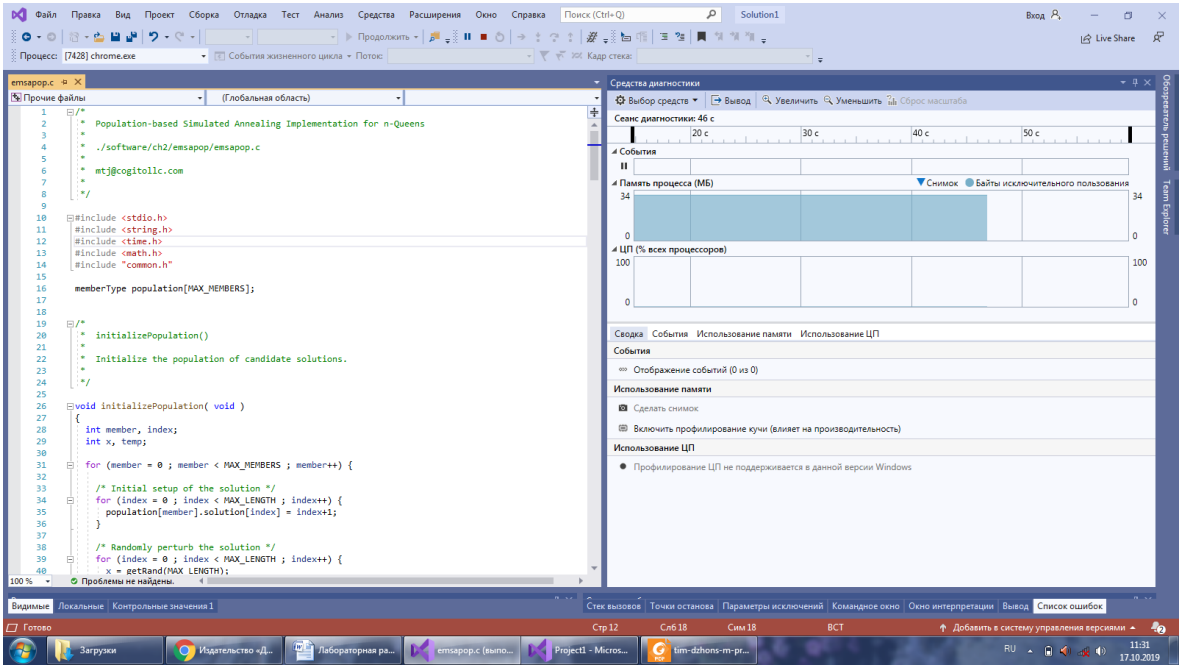
Вопросы к защите

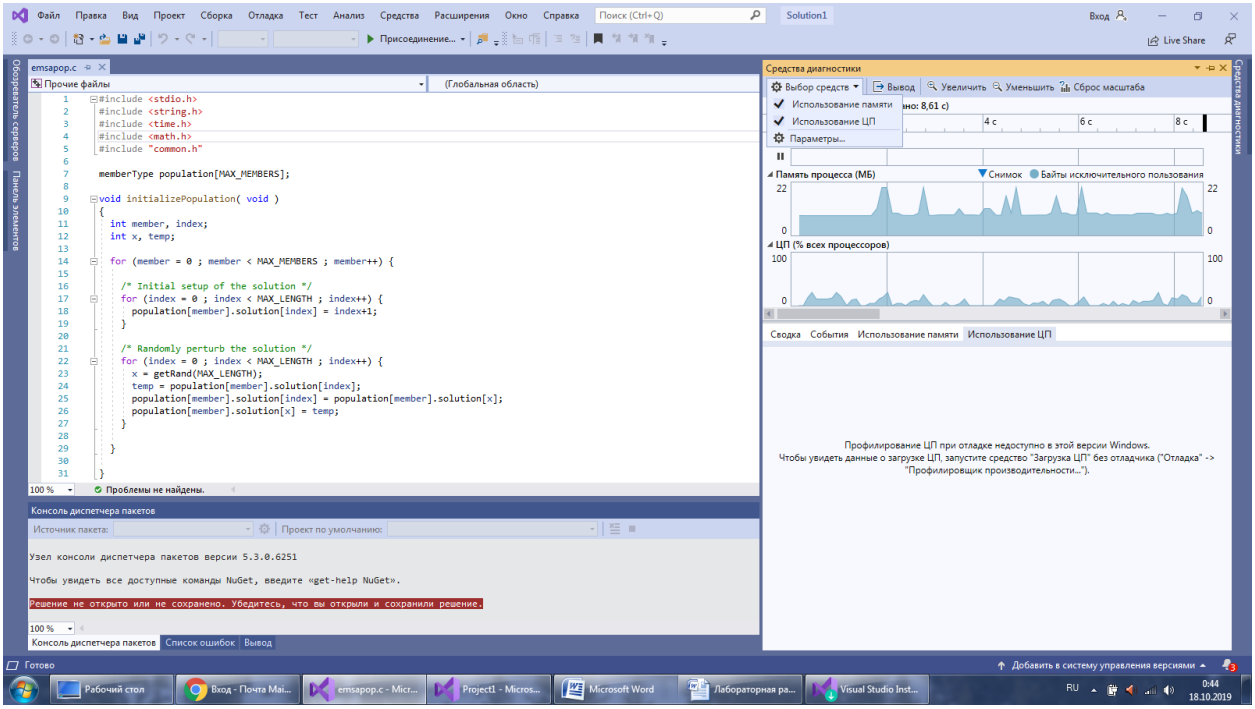
1. Почему алгоритм отжига представляет собой процесс генерации случайных чисел?
2. Какие причины обуславливают необходимость принимать в алгоритме отжига конечную температуру, отличную от нуля?
3. Опишите функцию вероятности допуска и ее роль в алгоритме отжига?
4. Укажите различия между понятиями начальное решение, текущее решение и рабочее решение, а также способы их представления в задаче о N ферзях.
5. В чем смысл понятия энергии, и ее значений в алгоритме отжига?
6. Приведите комментарий к основным частям программы решения задачи о N ферзях.

Литература

1. Андрейчиков О.В., Андрейчикова О.Н. Интеллектуальные информационные системы: Учебник.- М.: Финансы и статистика, 2004. – С. 32 - 41.
2. Т. Джонс. Программирование искусственного интеллекта в приложениях./ Пер. с англ. Осипов А.И. – М.: ДМК Пресс. 2004.- С. 25-42.
3. Задача N ферзей. Исходный код программы с использованием алгоритма отжига содержится в папке software/ch2. Загрузка с сайта издательства «ДМК Пресс» www.dmk.ru.






Файл Правка Вид Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
Solution1
Вход
Live Share
Присоединение...
emsapop.c
Прочие файлы
(Глобальная область)
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include "common.h"
memberType population[MAX_MEMBERS];
void initializePopulation(void)
{
int member, index;
int x, temp;
for (member = 0 ; member < MAX_MEMBERS ; member++) {
/* Initial setup of the solution */
for (index = 0 ; index < MAX_LENGTH ; index++) {
population[member].solution[index] = index+1;
}
/* Randomly perturb the solution */
for (index = 0 ; index < MAX_LENGTH ; index++) {
x = getRand(MAX_LENGTH);
temp = population[member].solution[index];
population[member].solution[index] = population[member].solution[x];
population[member].solution[x] = temp;
}
}
}
100 %
Проблемы не найдены.
Консоль диспетчера пакетов
Источник пакета:
Проект по умолчанию:
Узел консоли диспетчера пакетов версии 5.3.0.6251
Чтобы увидеть все доступные команды NuGet, введите «get-help NuGet».
Решение не открыто или не сохранено. Убедитесь, что вы открыли и сохранили решение.
Консоль диспетчера пакетов
Список ошибок
Вывод
Средства диагностики
Выбор средств
Вывод
Увеличить
Уменьшить
Сброс масштаба
Использование памяти
Использование ЦП
Параметры...
Память процесса (МБ)
Снимок
Байты исключительного пользования
ЦП (% всех процессоров)
Сводка
События
Использование памяти
Использование ЦП
Профилирование ЦП при отладке недоступно в этой версии Windows.
Чтобы увидеть данные о загрузке ЦП, запустите средство "Загрузка ЦП" без отладчика ("Отладка" -> "Профилировщик производительности...").
Готово
Добавить в систему управления версиями
Рабочий стол
Вход - Почта Mai...
emsapop.c - Micr...
Project1 - Micros...
Microsoft Word
Лабораторная ра...
Visual Studio Inst...
RU
0:44
18.10.2019

Приложение

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include "common.h"

memberType population[MAX_MEMBERS];

void initializePopulation( void )
{
  int member, index;
  int x, temp;

  for (member = 0 ; member < MAX_MEMBERS ; member++) {

    /* Initial setup of the solution */
    for (index = 0 ; index < MAX_LENGTH ; index++) {
      population[member].solution[index] = index+1;
    }

    /* Randomly perturb the solution */
    for (index = 0 ; index < MAX_LENGTH ; index++) {
      x = getRand(MAX_LENGTH);
      temp = population[member].solution[index];
      population[member].solution[index] = population[member].solution[x];
      population[member].solution[x] = temp;
    }

  }

}


/*
 * tweakSolution()
 *
 * Произвольно возмущать закодированное решение.
 *
 */

void tweakSolution( memberType *member )
{
  int temp, x, y;

  x = getRand(MAX_LENGTH);
  do {
    y = getRand(MAX_LENGTH);
  } while (x == y);

  temp = member->solution[x];
  member->solution[x] = member->solution[y];
  member->solution[y] = temp;
}


/*
 * эмитсолюция()
 *
 * Выделите раствор в шахматном виде.
 *
 */

void emitSolution( memberType *member )
{
```

```
  char board[MAX_LENGTH][MAX_LENGTH];
  int x, y;

  bzero( (void *)board, MAX_LENGTH * MAX_LENGTH );

  for (x = 0 ; x < MAX_LENGTH ; x++) {
    board[x][member->solution[x]-1] = 'Q';
  }

  printf("board:\n");
  for (y = 0 ; y < MAX_LENGTH ; y++) {
    for (x = 0 ; x < MAX_LENGTH ; x++) {
      if (board[x][y] == 'Q') printf("Q ");
      else printf(". ");
    }
    printf("\n");
  }
  printf("\n\n");
}


/*
 * computeEnergy()
 *
 * Рассчитайте энергию пройденного решения. Энергия
 * количество конфликтов на доске. Обратите внимание, что только диагонали
 * проверяются. Кодирование гарантирует, что нет верифицированных или горизонтальных
 * возможны конфликты.
 *
 */

float computeEnergy( memberType *member )
{
  int  i, j, x, y, tempx, tempy;
  char board[MAX_LENGTH][MAX_LENGTH];
  int conflicts;
  const int dx[4] = {-1,  1, -1,  1};
  const int dy[4] = {-1,  1,  1, -1};

  bzero( (void *)board, MAX_LENGTH * MAX_LENGTH );

  for (i = 0 ; i < MAX_LENGTH ; i++) {
    board[i][member->solution[i]-1] = 'Q';
  }

  / *Пройдите через каждую из ферзей и вычислите количество конфликтов * /
  conflicts = 0;

  for (i = 0 ; i < MAX_LENGTH ; i++) {

    x = i; y = member->solution[i]-1;


       / *Примечание: На основе кодирования, горизонтальные и вертикальные конфликты не
будут происходить!!!
             */
 *Проверьте диагонали * /

    for (j = 0 ; j < 4 ; j++) {

      tempx = x ; tempy = y;
      while(1) {
        tempx += dx[j]; tempy += dy[j];
        if ((tempx < 0) || (tempx >= MAX_LENGTH) ||
               (tempy < 0) || (tempy >= MAX_LENGTH)) break;
```

```
        if (board[tempx][tempy] == 'Q') conflicts++;
      }

    }

  }

  return (float)conflicts;
}


/*
 * имитированный отжиг()
 *
 * Выполните алгоритм имитированного отжига.
 *
 */

int simulateAnnealing( float curTemp )
{
  int member, i;
  memberType tempMember;
  float energy;
  int useNew = 0;
  int solution = -1;

  for (member = 0 ; member < MAX_MEMBERS ; member++) {

    for (i = 0 ; i < MAX_LENGTH ; i++) {
      tempMember.solution[i] = population[member].solution[i];
    }

    tweakSolution( &tempMember );

    energy = computeEnergy( &tempMember );

    useNew = 0;

    if (energy < population[member].energy) {
      useNew = 1;
    } else {
      float test = getSRand();
      float delta = energy - population[member].energy;

      if (exp(-delta/curTemp) > test) {
        useNew = 1;
      }
    }

    if (useNew) {

      for (i = 0 ; i < MAX_LENGTH ; i++) {
        population[member].solution[i] = tempMember.solution[i];
        population[member].energy = energy;
      }

    }

    if (population[member].energy == 0) solution = member;

  }

  return solution;
}
```

```c
/*
 *
 * Запустите через население и вычислить энергию для каждого
 * решение кандидата
 */

void computeAllEnergy( void )
{
  int member;

  for (member = 0 ; member < MAX_MEMBERS ; member++) {
    population[member].energy = computeEnergy( &population[member] );
  }

}


/* * Главным образом функция для сымитированной демонстрации отжига.
 */

int main()
{
  int  step, solution = -1;
  float temperature = INITIAL_TEMPERATURE;

  srand(time(NULL));

  initializePopulation();

  computeAllEnergy();

  while (temperature > FINAL_TEMPERATURE) {

    printf("temperature %f (solution %d)\n", temperature, solution);

    for (step = 0 ; step < STEPS_PER_CHANGE ; step++) {
      solution = simulateAnnealing( temperature );
    }

    temperature -= RATE;
  }

  if (solution == -1) {
    printf("No solution found\n");
  } else {
    emitSolution( &population[solution] );
  }

  return 0;
}
```