Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 5 з дисципліни
«Проектування алгоритмів»

„Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2”

Виконав(ла) *ІП-11 Панченко С. В.* ____________
(шифр, прізвище, ім'я, по батькові)

Перевірив *Головченко М.Н.* ____________
(прізвище, ім'я, по батькові)

Київ 2022

Варіант 23

Мета роботи – вивчити основні підходи розробки метаевристичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

Покроковий алгоритм

```
def Solve()
   StartInitialGeneration()
   SortGeneration()
   bestChromosome = generation[size-1]
   while timer :
      parentOne = Generation[size - 1]
      index = rand() % (size - 1)
      parentTwo = Generation[index]
      chrome = Crossover(parentOne, parentTwo)
      if not isValid(chrome) skip
      chromeCopy = chrome
      Mutate(chrome)
      if not isValid(chrome): chrome = chromeCopy
      chromeCopy = chrome
      Improve(chrome)
      if not isValid(chrome): chrome = chromeCopy
      if CalculatePrice(chrome)>CalculatePrice(BestChromosome):
         BestChromosome = chrome
      add BestChromosome to generation
      SortGeneration()
      erase first element from Generation


def isValid(chrome):
   return CalculateWeight(chrome)<=m_uMaxWeight


def Crossover(one, two)
   index = rand() % Crossovers.size()
   return Crossovers[index](one, two)


def Mutate(chrome)
      if Mutators:
      index = rand() % Mutators.size()
      Mutators[index](this, chrome)


def Improve(chrome):
               if Improvers:
      index = rand() % Improvers.size()
      Improvers[index](chrome)


def StartInitialGeneration():
        for i in m_uInitialPopulationNumber:
           add FindFirstPossibleSolution() to generation



    def SortGeneration():
        sort generation by CalculatePrice(chromeOne)<CalculatePrice(chromeTwo)

    def CalculatePrice(chrome):
        price = 0
        for i in inputSize:
            if chrome[i]:
                price += Input[i].price
        return price


    def CalculateWeight(chrome):
        weight = 0
        for i in inputSize:
            if chrome[i]:
                weight += Input[i].weight
        return weight


    def FindFirstPossibleSolution()
        chromo = Chromosome()
        curWeight = 0
        while true:
            index = rand() % inputSize
            curWeight += Input[index].weight
            if curWeight>m_uMaxWeight: break
            chromo[index] = true
        return chromo

def PointCrossover(one, two)
        chromo = Chromosome()
        if inputSize<pointNumber:
            number = inputSize
        else:
            number = pointNumber
        partition = inputSize / (number + 1)
        for i in number:
            for index in i*partition:
                chromo[index] = (i%2)?one[index]:two[index]
        for index in inputSize:
        if i%2 == 0:
            chromo[index] = one[index]
        else:
            chromo[index] = two[index]
        return chromo
```

```
def SequenceCrossover(one, two)
    chromo = Chromosome()
    for i in inputSize:
        if i%2==0:
            chromo[i] = one[i]
        else
            chromo[i] = two[i]
    return chromo


def SwapMutator(chromo):
    indexOne = rand() % inputSize
    indexTwo = rand() % inputSize
    swap(chromo[indexOne], chromo[indexTwo])


def ChangeSignMutator(chromo)
    index = rand() % inputSize
    chromo[index] = not chromo[index]


def WorstPriceImprover(chromo)
    for i in inputSize:
        if chromo[i] and input[i].price<input[index].price:
            index = i
    weight = CalculateWeight(chromo)
    weight -= m_vInput[index].weight
    isImproved = false
    for i in inputSize and not isImproved:
        if not chromo[i] and input[index].price <= input[i].price:
            posWeight = weight + m_vInput[i].weight
            if posWeight <= m_uMaxWeight:
                chromo[index] = false
                chromo[i] = true
                isImproved = true

def BestPriceImprover(chromo)
    index = 0
    for i in inputSize:
        if chromo[i] and input[i].price > m_vInput[index].price:
            index = i



    weight = CalculateWeight(chromo)
    weight -= m_vInput[index].weight
    isImproved = false
    for i in inputSize and not isImproved:
        if not chromo[i] and input[index].price <= input[i].price:
            posWeight = weight + m_vInput[i].weight
            if posWeight <= m_uMaxWeight:
                chromo[index] = false
                chromo[i] = true
```

```
                        isImproved = true
```

Програмна реалізація алгоритму

```
//@formatter:off
#ifndef LAB5_TGENETICALGORITHM_H
#define LAB5_TGENETICALGORITHM_H
#include <cstdlib>
#include <random>
#include <vector>
#include <functional>
#include <algorithm>
#include <memory>
#include "TTimer.h"

class TTester;

template<unsigned inputSize>
class TGeneticAlgorithm {
    friend class TTester;
    public:
    using Input = std::array<std::pair<unsigned, unsigned>, inputSize>;
    using Chromosome = std::array<bool, inputSize>;
    using Generation = std::vector<Chromosome>;
    using Crossovers = std::vector<std::function<Chromosome(TGeneticAlgorithm<inputSize>*,
const Chromosome&, const Chromosome&)>>;
    using Mutators = std::vector<std::function<void(TGeneticAlgorithm<inputSize>*,
Chromosome&)>>;
    using Improvers = std::vector<std::function<void(TGeneticAlgorithm<inputSize>*,
Chromosome&)>>;

    public:
    TGeneticAlgorithm()=default;
    TGeneticAlgorithm(unsigned initialPopulationsNumber, unsigned milliseconds, unsigned
maxWeight, const Input& input,
    const Crossovers& crossovers, const Mutators& mutators, const Improvers& improvers) {
        m_uInitialPopulationNumber = initialPopulationsNumber;
        m_uMilliseconds = milliseconds;
        m_uMaxWeight = maxWeight;
        m_vInput = input;
        m_vCrossovers = crossovers;
        m_vMutators = mutators;
        m_vImprovers = improvers;
    }

    public:
    void Solve() {
        StartInitialGeneration();
        TTimer timer;
        timer.start();
        SortGeneration();
        m_vBestChromosome = m_vGeneration.back();
        while(timer.elapsedMilliseconds()<m_uMilliseconds) {
```

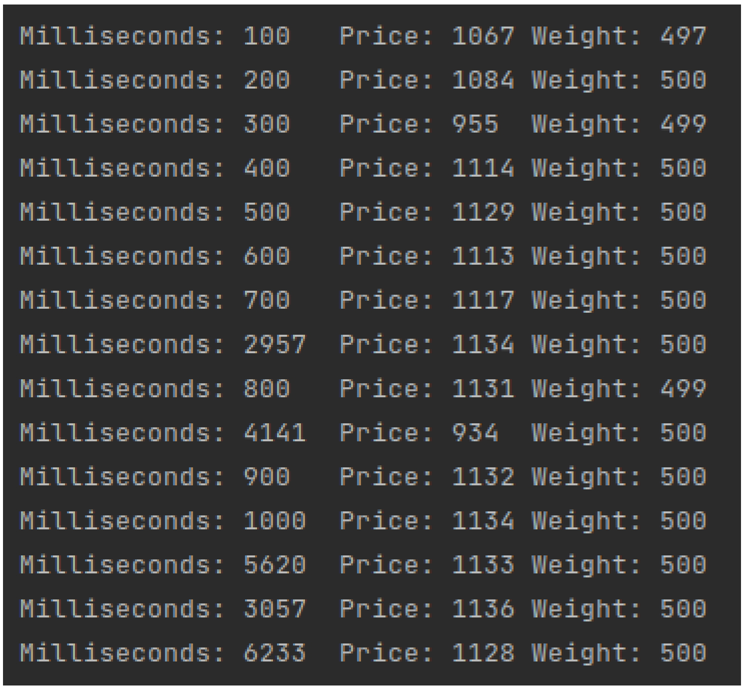
```
        // pick parent
        auto size = m_vGeneration.size();
        auto& parentOne = m_vGeneration[size - 1];
        auto index = rand() % (size - 1);
        auto& parentTwo = m_vGeneration[index];
        // crossover
        auto chrome = Crossover(parentOne, parentTwo);
        if(!isValid(chrome)) continue;
        // mutate
        auto chromeCopy = chrome;
        Mutate(chrome);
        if(!isValid(chrome)) chrome = chromeCopy;
        // improve
        chromeCopy = chrome;
        Improve(chrome);
        if(!isValid(chrome)) chrome = chromeCopy;
        if(CalculatePrice(chrome)>CalculatePrice(m_vBestChromosome)) {
            m_vBestChromosome = chrome;
        }
        m_vGeneration.push_back(m_vBestChromosome);
        SortGeneration();
        m_vGeneration.erase(m_vGeneration.begin());
    }
}

protected:
bool isValid(const Chromosome& chrome) {
    return CalculateWeight(chrome)<=m_uMaxWeight;
}

Chromosome Crossover(const Chromosome& one, const Chromosome& two) {
    // randomly choose crossover
    auto index = rand() % m_vCrossovers.size();
    return m_vCrossovers[index](this, one, two);
}

void Mutate(Chromosome& chrome) {
        if(m_vMutators.empty()) return;
    auto index = rand() % m_vMutators.size();
    m_vMutators[index](this, chrome);
}

void Improve(Chromosome& chrome) {
                    if(m_vImprovers.empty()) return;
    auto index = rand() % m_vImprovers.size();
    m_vImprovers[index](this, chrome);
}

void StartInitialGeneration() {
    m_vGeneration.reserve(m_uInitialPopulationNumber);
    for(unsigned i=0;i<m_uInitialPopulationNumber;++i) {
       m_vGeneration.push_back(FindFirstPossibleSolution());
    }
}

void SortGeneration() {
    std::sort(m_vGeneration.begin(), m_vGeneration.end(),
        [this](auto& chromeOne, auto& chromeTwo) {
            return CalculatePrice(chromeOne)<CalculatePrice(chromeTwo);
        }
    );
}

unsigned CalculatePrice(const Chromosome& chrome) {
    auto price = 0;
    for(unsigned i=0;i<inputSize;++i) {
        if(chrome[i]) {
            price += m_vInput[i].first;
        }
    }
    return price;
}

unsigned CalculateWeight(const Chromosome& chrome) {
    auto weight = 0;
    for(unsigned i=0;i<inputSize;++i) {
        if(chrome[i]) {
            weight += m_vInput[i].second;
        }
    }
    return weight;
}

Chromosome FindFirstPossibleSolution() {
    srand(time(NULL));
    auto chromo = Chromosome();
    auto curWeight = 0;
    while(true) {
        auto index = rand() % inputSize;
        curWeight += m_vInput[index].second;
        if(curWeight>m_uMaxWeight) break;
        chromo[index] = true;
    }
    return chromo;
}


protected:
unsigned m_uInitialPopulationNumber = 0;
unsigned m_uMilliseconds = 0;
unsigned m_uMaxWeight = 0;

protected:
Input m_vInput;
```

```
    Generation m_vGeneration;
    Crossovers m_vCrossovers;
    Mutators m_vMutators;
    Improvers m_vImprovers;
    Chromosome m_vBestChromosome;
};

#endif //LAB5_TGENETICALGORITHM_H
//@formatter:on


#ifndef LAB5_TTESTER_H
#define LAB5_TTESTER_H
#include <iostream>
#include <cassert>
#include <future>
#include <thread>
#include <mutex>
#include <filesystem>
#include "include/pbPlots.hpp"
#include "include/supportLib.hpp"
#include "TGeneticAlgorithm.h"

class TTester {
    public:
    TTester()=default;
    TTester(const TTester&)=default;
    TTester(TTester&&)=default;
    virtual ~TTester()=default;

            protected:
            static constexpr char s_sBasePath[] =
"/home/choleraplague/university/Algorithms_KPI_2_Year/lab5/results/";
    static constexpr unsigned s_uInputSize = 100;
            static constexpr unsigned s_uMaxWeight = 500;
            static constexpr unsigned s_uLPrice= 2;
            static constexpr unsigned s_uMPrice= 30;
            static constexpr unsigned s_uLWeight = 1;
            static constexpr unsigned s_uMWeight= 20;
            static constexpr unsigned s_uInitialPopulation = 10;
            static constexpr unsigned s_uThreads = 12;
            static constexpr unsigned s_uStartTime = 100;
            static constexpr unsigned s_uEndTime = 10000;
            static constexpr unsigned s_uAddTime = 100;

            protected:
            using Chromosome = typename TGeneticAlgorithm<s_uInputSize>::Chromosome;
            using Input = typename TGeneticAlgorithm<s_uInputSize>::Input;
            using Crossovers = typename TGeneticAlgorithm<s_uInputSize>::Crossovers;
    using Mutators = typename TGeneticAlgorithm<s_uInputSize>::Mutators;
    using Improvers = typename TGeneticAlgorithm<s_uInputSize>::Improvers;
            using AlgData = std::pair<std::vector<double>, std::vector<double>>;
            using Results = std::vector<AlgData>;
            using Tasks = std::vector<std::future<AlgData>>;

            protected:
            Input m_vInput;
            Crossovers m_vCrossovers;
            Mutators m_vMutators;
    Improvers m_vImprovers;

            protected:

            #define VAR_TO_STR(x) \
                ""#x
            #define ADD_FUNC_TO_VECTOR(func, vec, str) \
                vec.push_back(func);           \
                AddFuncToStr(str, #func)

            static void AddAttributeToStr(std::string& str, std::string&& attrStr, unsigned attr) {
                str += attrStr + "_" + std::to_string(attr) + " ";
            }
            static void AddFuncToStr(std::string& str, std::string&& funcStr) {
                str += funcStr + " ";
            }

            public:
            void Test() {
              auto legend = std::string();
                AddAttributeToStr(legend, VAR_TO_STR(s_uInputSize), s_uInputSize);
                AddAttributeToStr(legend, VAR_TO_STR(s_uMaxWeight), s_uMaxWeight);
                AddAttributeToStr(legend, VAR_TO_STR(s_uInitialPopulation),
s_uInitialPopulation);
                AddAttributeToStr(legend, VAR_TO_STR(s_uStartTime), s_uStartTime);
                AddAttributeToStr(legend, VAR_TO_STR(s_uEndTime), s_uEndTime);
                AddAttributeToStr(legend, VAR_TO_STR(s_uAddTime), s_uAddTime);

              auto directory = std::string();
                ADD_FUNC_TO_VECTOR(PointCrossover<2>, m_vCrossovers, directory);
//      ADD_FUNC_TO_VECTOR(PointCrossover<4>, m_vCrossovers, directory);
                ADD_FUNC_TO_VECTOR(PointCrossover<1>, m_vCrossovers, directory);
                ADD_FUNC_TO_VECTOR(SequenceCrossover, m_vCrossovers, directory);


                ADD_FUNC_TO_VECTOR(ChangeSignMutator, m_vMutators, directory);
                ADD_FUNC_TO_VECTOR(SwapMutator, m_vMutators, directory);
//
                ADD_FUNC_TO_VECTOR(WorstPriceImprover, m_vImprovers, directory);
                ADD_FUNC_TO_VECTOR(BestPriceImprover, m_vImprovers, directory);

                auto funcs = s_sBasePath + directory;
                auto params = funcs + "/" + legend;
                std::filesystem::create_directory(funcs);
                std::filesystem::create_directory(params);
                std::filesystem::current_path(params);

                GenerateInput();

                auto results = Results();
                auto tasks = Tasks();
                DistributeTasks(tasks);
                auto e = std::vector<wchar_t>();
                for(auto& t : tasks) {
                    results.push_back(t.get());
                }

                auto x = std::vector<double>();
                auto y = std::vector<double>();
                for(const auto& res : results) {
                    x.insert(x.end(), res.first.begin(), res.first.end());
                    y.insert(y.end(), res.second.begin(), res.second.end());
                }

                auto series = GetDefaultScatterPlotSeriesSettings();
                series->xs = &x;
                series->ys = &y;
                auto vs = std::vector<ScatterPlotSeries*>();
                vs.push_back(series);

                auto titleVec = std::vector<wchar_t>(legend.begin(), legend.end());
                auto xlabelVec = std::vector<wchar_t>(directory.begin(), directory.end());

                auto settings = GetDefaultScatterPlotSettings();
                settings->width = 1280;
                settings->height = 720;
                settings->autoBoundaries = true;
                settings->autoPadding = true;
                settings->title = &titleVec;
                settings->xLabel = &xlabelVec;
                settings->scatterPlotSeries = &vs;

                RGBABitmapImageReference* imageRef = CreateRGBABitmapImageReference();
                StringReference* errorMessage = CreateStringReference(&e);
                DrawScatterPlotFromSettings(imageRef, settings, errorMessage);

                auto pngData = ConvertToPNG(imageRef->image);

                WriteToFile(pngData, GenerateFileName());
                DeleteImage(imageRef->image);
    }

            protected:
            static std::basic_string<char> GenerateFileName() {
                auto path = std::filesystem::current_path();
                auto p = std::filesystem::current_path().string();
                auto it = std::filesystem::directory_iterator(std::filesystem::current_path());
```

```
		auto paths = std::vector<std::string>();
		while(it!=std::filesystem::end(it)) {
			paths.push_back(it->path().string());
			++it;
		}
		auto index = 0;
		while(std::find(paths.begin(), paths.end(), p + "/" + std::to_string(index) + ".png")!
=paths.end()) {
			++index;
		}
		return std::to_string(index) + ".png";
	}

	void DistributeTasks(Tasks& tasks) {
		auto timePoints = std::array<std::pair<unsigned, unsigned>, s_uThreads>();
		DistributeTimeEvenly(timePoints);
		for(const auto& t : timePoints) {
			tasks.push_back(std::async(std::launch::async, DoTest, t.first, t.second,
s_uAddTime, m_vCrossovers, m_vMutators, m_vImprovers, m_vInput));
		}
	}

	static void DistributeTimeEvenly(std::array<std::pair<unsigned, unsigned>, s_uThreads>&
timePoints) {
		auto initial = std::make_pair(s_uStartTime, s_uEndTime);
		auto splitOne = SplitArea(initial, 2);

		auto splitTwo = SplitArea(splitOne[0], 3);
		auto sixthOne = splitTwo[0];
		auto [sixthTwo, sixthThree] = SplitArea(splitTwo[1], 2);

		auto splitThree = SplitArea(splitOne[1], 3);
		auto sixthFour = splitThree[0];
		auto [sixthFive, sixthSix] = SplitArea(splitThree[1], 2);

		auto [one, two] = SplitArea(sixthOne, 2);
		timePoints[0] = one;
		timePoints[1] = two;
		auto [three, four] = SplitArea(sixthTwo, 2);
		timePoints[2] = three;
		timePoints[3] = four;
		auto [five, six] = SplitArea(sixthThree, 2);
		timePoints[4] = five;
		timePoints[5] = six;
		auto [seven, eight] = SplitArea(sixthFour, 2);
		timePoints[6] = seven;
		timePoints[7] = eight;
		auto [nine, ten] = SplitArea(sixthFive, 2);
		timePoints[8] = nine;
		timePoints[9] = ten;
		auto [eleven, twelve] = SplitArea(sixthSix, 2);
		timePoints[10] = eleven;
```

```
        timePoints[11] = twelve;
    }

    static std::array<std::pair<unsigned, unsigned>, 2> SplitArea(const std::pair<unsigned,
unsigned>& p, unsigned ratio) {
        auto pairs = std::array<std::pair<unsigned , unsigned>, 2>();
        auto len = unsigned(double(p.second - p.first) / sqrt(ratio)) + p.first;
        pairs[0] = std::make_pair(p.first, len);
        pairs[1] = std::make_pair(len, p.second);
        return pairs;
    }

    protected:
    static AlgData DoTest(double start, double end, double add,
    const Crossovers& crossovers, const Mutators& mutators, const Improvers& improvers,
const Input& input) {

        auto x = std::vector<double>();
        auto y = std::vector<double>();
        for(auto i=start;i<end;i+=add) {
            x.push_back(i);
            auto alg = TGeneticAlgorithm<s_uInputSize>(s_uInitialPopulation, i,
s_uMaxWeight, input,
            crossovers, mutators, improvers);
            alg.Solve();
            auto price = alg.CalculatePrice(alg.m_vBestChromosome);
            auto weight = alg.CalculateWeight(alg.m_vBestChromosome);
            std::cout<<"Milliseconds: "<<i<<"\t"<<"Price: "<<price<<"\t"<<"Weight:
"<<weight<<"\n";
            y.push_back(price);
        }
        return std::make_pair(x, y);

    }

    protected:
    void GenerateInput() {
        for(unsigned i=0;i<s_uInputSize;++i) {
            auto weight = s_uLWeight + rand() % (s_uMWeight - s_uLWeight - 1);
            auto price = s_uLPrice + rand() % (s_uMPrice - s_uLPrice - 1);
            m_vInput[i] = std::make_pair(price, weight);
        }
    }

   template<unsigned pointNumber>
   static Chromosome PointCrossover(TGeneticAlgorithm<s_uInputSize>* alg, const
Chromosome& one, const Chromosome& two) {
        assert(pointNumber>0);

    auto chromo = Chromosome();
    auto number = (s_uInputSize<pointNumber)?s_uInputSize:pointNumber;
    auto partition = s_uInputSize / (number + 1);
    auto i = 1;
    auto index = 0;
    for(;i<=number;++i) {
        for(;index<i*partition;++index) {
            chromo[index] = (i%2)?one[index]:two[index];
        }
    }
    for(;index<s_uInputSize;++index) {
            chromo[index] = (i%2)?one[index]:two[index];
    }
    return chromo;
  }

  static Chromosome SequenceCrossover(TGeneticAlgorithm<s_uInputSize>* alg, const
Chromosome& one, const Chromosome& two) {
    auto chromo = Chromosome();
    for(unsigned i=0;i<s_uInputSize;++i) {
            chromo[i] = (i%2)?one[i]:two[i];
        }
        return chromo;
    }

    static void SwapMutator(TGeneticAlgorithm<s_uInputSize>* alg, Chromosome& chromo)
{
        auto indexOne = rand() % s_uInputSize;
        auto indexTwo = rand() % s_uInputSize;
        std::swap(chromo[indexOne], chromo[indexTwo]);
    }

    static void ChangeSignMutator(TGeneticAlgorithm<s_uInputSize>* alg, Chromosome&
chromo) {
        auto index = rand() % s_uInputSize;
        chromo[index] = !chromo[index];
    }

    static void WorstPriceImprover(TGeneticAlgorithm<s_uInputSize>* alg, Chromosome&
chromo) {
        unsigned index = 0;
        for(unsigned i=0;i<s_uInputSize;++i) {
            if(chromo[i] and alg->m_vInput[i].first<alg->m_vInput[index].first) {
                index = i;
            }
        }
        auto weight = alg->CalculateWeight(chromo);
        weight -= alg->m_vInput[index].second;
        auto isImproved = false;
        for(unsigned i=0;i<s_uInputSize and !isImproved;++i) {
            if(!chromo[i] and alg->m_vInput[index].first <= alg->m_vInput[i].first) {
                auto posWeight = weight + alg->m_vInput[i].second;
                if(posWeight<=alg->m_uMaxWeight) {
                    chromo[index] = false;
                    chromo[i] = true;
                    isImproved = true;
                }
            }
        }
    }

    static void BestPriceImprover(TGeneticAlgorithm<s_uInputSize>* alg, Chromosome& chromo) {
        unsigned index = 0;
        for(unsigned i=0;i<s_uInputSize;++i) {
            if(chromo[i] and alg->m_vInput[i].first>alg->m_vInput[index].first) {
                index = i;
            }
        }
        auto weight = alg->CalculateWeight(chromo);
        weight -= alg->m_vInput[index].second;
        auto isImproved = false;
        for(unsigned i=0;i<s_uInputSize and !isImproved;++i) {
            if(!chromo[i] and alg->m_vInput[index].first <= alg->m_vInput[i].first) {
                auto posWeight = weight + alg->m_vInput[i].second;
                if(posWeight<=alg->m_uMaxWeight) {
                    chromo[index] = false;
                    chromo[i] = true;
                    isImproved = true;
                }
            }
        }
    }
};

#endif //LAB5_TTESTER_H


#include <iostream>
#include "TTester.h"
int main() {
    srand(time(NULL));

    for(int i=0;i<5;++i) {
        TTester test;
        test.Test();
    }
}
```

Приклад роботи:



```
Milliseconds: 100   Price: 1067 Weight: 497
Milliseconds: 200   Price: 1084 Weight: 500
Milliseconds: 300   Price: 955  Weight: 499
Milliseconds: 400   Price: 1114 Weight: 500
Milliseconds: 500   Price: 1129 Weight: 500
Milliseconds: 600   Price: 1113 Weight: 500
Milliseconds: 700   Price: 1117 Weight: 500
Milliseconds: 2957  Price: 1134 Weight: 500
Milliseconds: 800   Price: 1131 Weight: 499
Milliseconds: 4141  Price: 934  Weight: 500
Milliseconds: 900   Price: 1132 Weight: 500
Milliseconds: 1000  Price: 1134 Weight: 500
Milliseconds: 5620  Price: 1133 Weight: 500
Milliseconds: 3057  Price: 1136 Weight: 500
Milliseconds: 6233  Price: 1128 Weight: 500
```

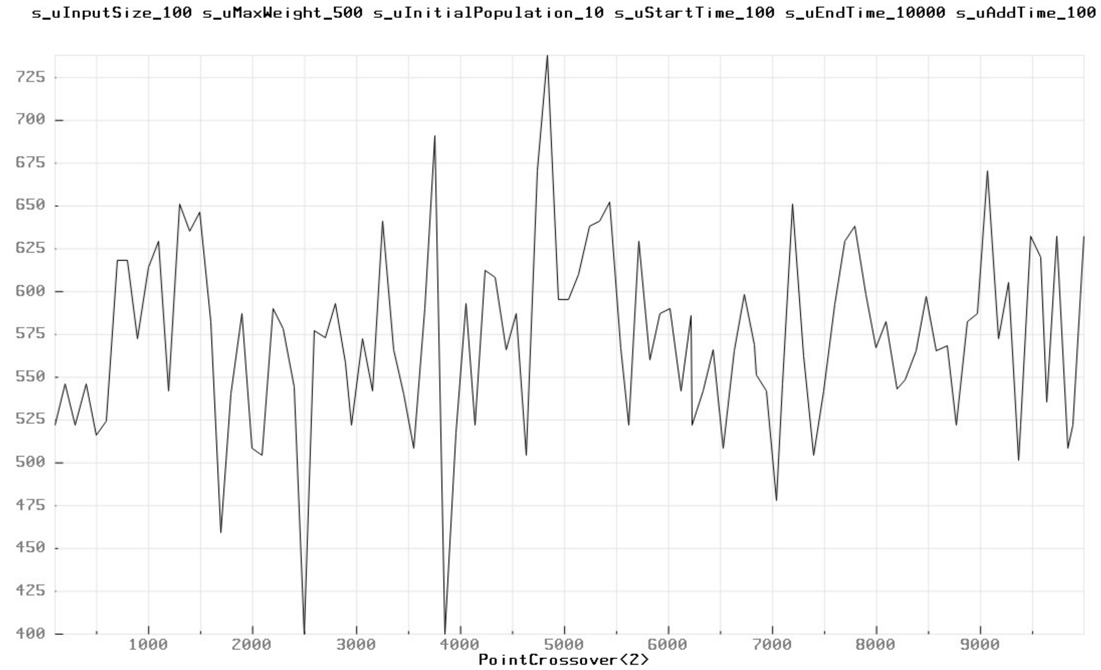


Рисунок 3.1 – PointCrossover<2>

Milliseconds	Price
100	522
200	546
300	522
400	546
500	516
600	524
700	618

800	618
900	572
1000	614
1100	629
1200	542
1300	651
1400	635
1500	646
1600	582
1700	459
1800	540
1900	587
2000	508
2100	504
2200	590
2300	578
2400	544
2500	400
2600	577
2700	573
2800	593
2900	558
2957	522
3057	572
3157	542
3257	641
3357	566
3457	540
3557	508
3657	590
3757	691
3857	400
3957	517
4057	593
4141	522
4241	612
4341	608
4441	566
4541	587
4641	504
4741	670
4841	738
4941	595
5041	595
5141	610
5241	638
5341	641
5441	652
5541	567
5620	522
5720	629
5820	560

5920	587
6020	590
6120	542
6220	586
6233	522
6333	542
6433	566
6533	508
6633	565
6733	598
6833	569
6846	551
6946	542
7046	478
7100	546
7200	651
7300	562
7400	504
7500	542
7600	592
7700	629
7800	638
7900	598
8000	567
8100	582
8200	543
8283	548
8383	565
8483	597
8583	565
8683	568
8774	522
8874	582
8974	587
9074	670
9174	572
9274	605
9374	501
9386	522
9486	632
9586	620
9640	535
9740	632
9840	508
9894	522
9994	632

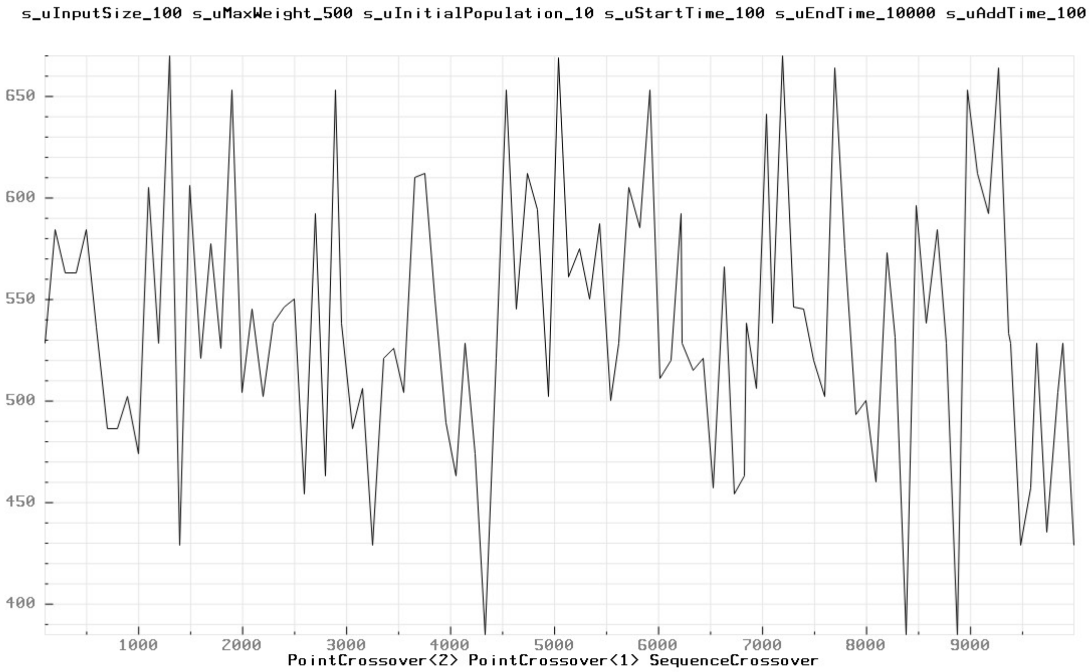


Рисунок 3.2 - PointCrossover<2> PointCrossover<1> SequenceCrossover

Milliseconds	Price
100	528
200	584
300	563
400	563
500	584
600	536
700	486
800	486
900	502
1000	474
1100	605
1200	528
1300	670
1400	429
1500	606
1600	521
1700	577
1800	526
1900	653
2000	504
2100	545
2200	502
2300	538
2400	546
2500	550
2600	454
2700	592
2800	463
2900	653

2957	538
3057	486
3157	506
3257	429
3357	521
3457	526
3557	504
3657	610
3757	612
3857	550
3957	489
4057	463
4141	528
4241	474
4341	385
4441	521
4541	653
4641	545
4741	612
4841	594
4941	502
5041	669
5141	561
5241	575
5341	550
5441	587
5541	500
5620	528
5720	605
5820	585
5920	653
6020	511
6120	520
6220	592
6233	528
6333	515
6433	521
6533	457
6633	566
6733	454
6833	463
6846	538
6946	506
7046	641
7100	538
7200	670
7300	546
7400	545
7500	520
7600	502
7700	664
7800	575

7900	493
8000	500
8100	460
8200	573
8283	531
8383	385
8483	596
8583	538
8683	584
8774	528
8874	385
8974	653
9074	612
9174	592
9274	664
9374	533
9386	528
9486	429
9586	457
9640	528
9740	435
9840	504
9894	528
9994	429

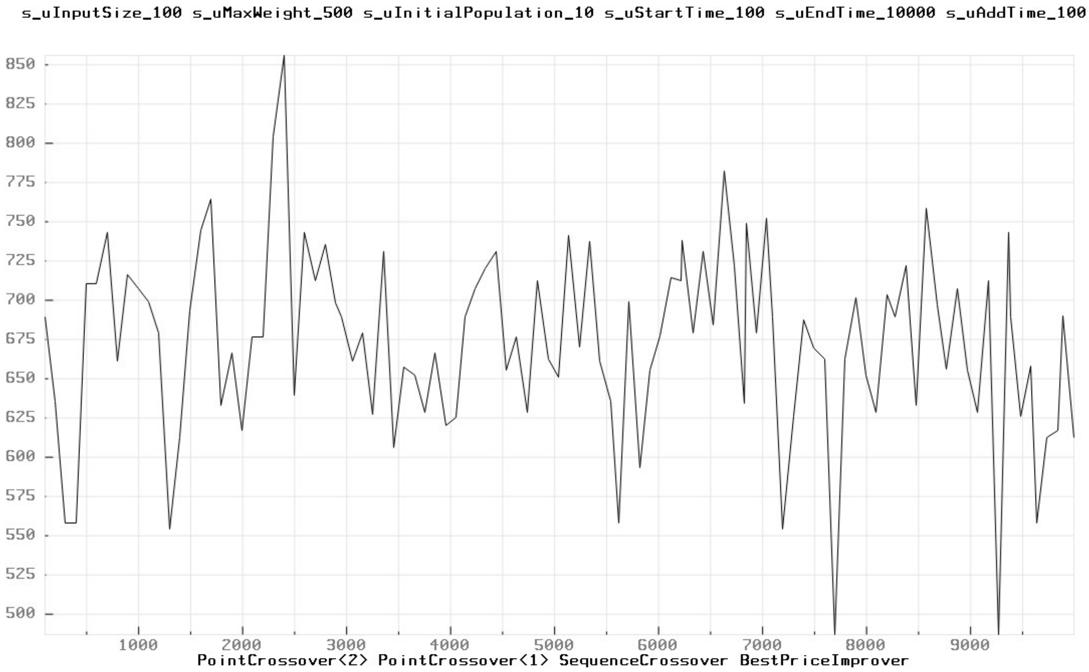


Рисунок 3.3 - PointCrossover<2> PointCrossover<1> SequenceCrossover BestPriceImprover

Milliseconds	Price
100	689
200	635
300	558
400	558

500	710
600	710
700	743
800	661
900	716
1000	707
1100	699
1200	679
1300	554
1400	612
1500	693
1600	744
1700	764
1800	633
1900	666
2000	617
2100	676
2200	676
2300	804
2400	856
2500	639
2600	743
2700	712
2800	735
2900	698
2957	689
3057	661
3157	679
3257	627
3357	731
3457	606
3557	657
3657	652
3757	628
3857	666
3957	620
4057	625
4141	689
4241	707
4341	720
4441	731
4541	655
4641	676
4741	628
4841	712
4941	662
5041	651
5141	741
5241	670
5341	737
5441	661
5541	635

5620	558
5720	699
5820	593
5920	655
6020	677
6120	714
6220	712
6233	738
6333	679
6433	731
6533	684
6633	782
6733	722
6833	634
6846	749
6946	679
7046	752
7100	689
7200	554
7300	626
7400	687
7500	669
7600	662
7700	487
7800	662
7900	701
8000	652
8100	628
8200	703
8283	689
8383	722
8483	633
8583	758
8683	697
8774	656
8874	707
8974	655
9074	628
9174	712
9274	487
9374	743
9386	689
9486	626
9586	658
9640	558
9740	612
9840	617
9894	690
9994	612

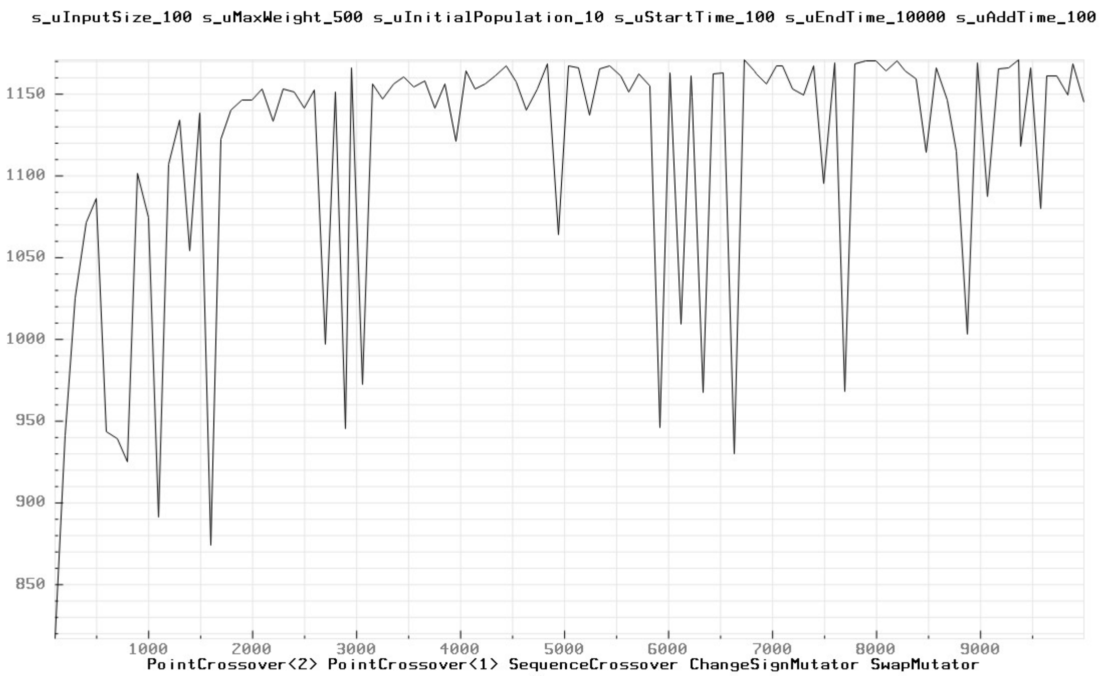


Рисунок 3.4 - PointCrossover<2> PointCrossover<1> SequenceCrossover ChangeSignMutator SwapMutator

Milliseconds	Price
100	817
200	941
300	1025
400	1071
500	1086
600	943
700	939
800	925
900	1101
1000	1074
1100	891
1200	1107
1300	1134
1400	1054
1500	1138
1600	874
1700	1122
1800	1140
1900	1146
2000	1146
2100	1153
2200	1133
2300	1153
2400	1151
2500	1141
2600	1152

2700	997
2800	1151
2900	945
2957	1166
3057	972
3157	1156
3257	1147
3357	1156
3457	1160
3557	1154
3657	1158
3757	1141
3857	1156
3957	1121
4057	1164
4141	1153
4241	1156
4341	1161
4441	1167
4541	1157
4641	1140
4741	1153
4841	1168
4941	1064
5041	1167
5141	1166
5241	1137
5341	1165
5441	1167
5541	1161
5620	1151
5720	1162
5820	1155
5920	946
6020	1163
6120	1009
6220	1161
6233	1148
6333	967
6433	1162
6533	1163
6633	930
6733	1171
6833	1164
6846	1162
6946	1156
7046	1167
7100	1167
7200	1153
7300	1149
7400	1167
7500	1095

7600	1169
7700	968
7800	1168
7900	1170
8000	1170
8100	1164
8200	1170
8283	1164
8383	1159
8483	1114
8583	1166
8683	1146
8774	1115
8874	1003
8974	1169
9074	1087
9174	1165
9274	1166
9374	1171
9386	1118
9486	1166
9586	1080
9640	1161
9740	1161
9840	1149
9894	1168
9994	1145

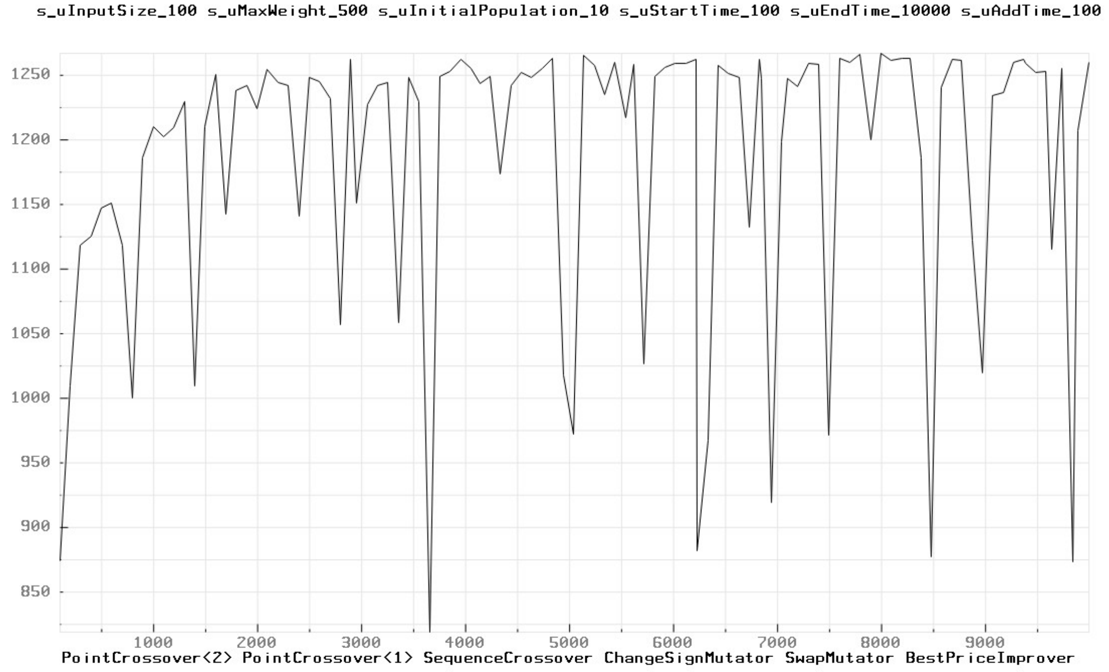


Рисунок 3.5 - PointCrossover<2> PointCrossover<1> SequenceCrossover ChangeSignMutator SwapMutator BestPriceImprover

Milliseconds	Price

100	874
200	1009
300	1118
400	1125
500	1147
600	1151
700	1118
800	1000
900	1186
1000	1210
1100	1202
1200	1209
1300	1229
1400	1009
1500	1210
1600	1250
1700	1142
1800	1238
1900	1242
2000	1224
2100	1254
2200	1244
2300	1242
2400	1141
2500	1248
2600	1245
2700	1232
2800	1057
2900	1262
2957	1151
3057	1227
3157	1242
3257	1244
3357	1058
3457	1248
3557	1229
3657	819
3757	1249
3857	1253
3957	1262
4057	1255
4141	1243
4241	1249
4341	1173
4441	1242
4541	1252
4641	1248
4741	1255
4841	1263
4941	1018
5041	972
5141	1265

5241	1257
5341	1235
5441	1260
5541	1217
5620	1258
5720	1026
5820	1249
5920	1256
6020	1259
6120	1259
6220	1262
6233	882
6333	967
6433	1257
6533	1251
6633	1248
6733	1132
6833	1262
6846	1247
6946	919
7046	1198
7100	1247
7200	1241
7300	1259
7400	1258
7500	971
7600	1263
7700	1260
7800	1266
7900	1200
8000	1267
8100	1261
8200	1263
8283	1263
8383	1186
8483	877
8583	1240
8683	1262
8774	1261
8874	1122
8974	1019
9074	1234
9174	1236
9274	1260
9374	1262
9386	1259
9486	1252
9586	1253
9640	1115
9740	1255
9840	873
9894	1207

9994	1260

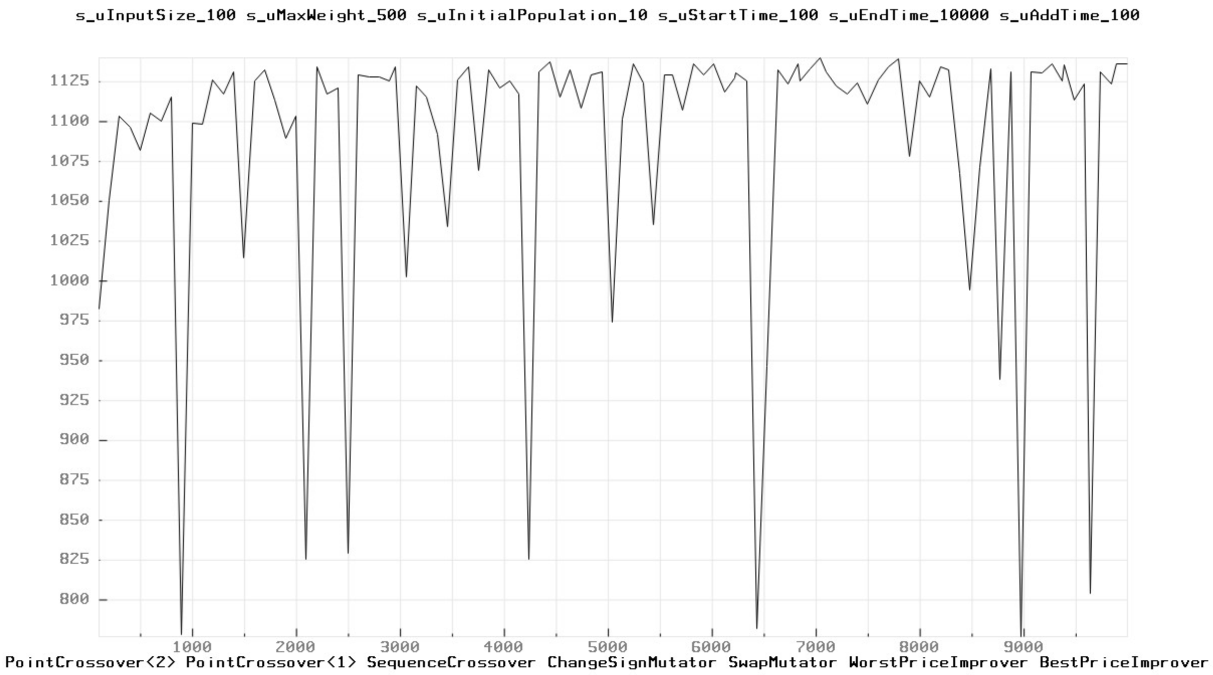


Рисунок 3.6 - PointCrossover<2> PointCrossover<1> SequenceCrossover ChangeSignMutator SwapMutator WorstPriceImprover BestPriceImprover

100	982
200	1050
300	1103
400	1096
500	1082
600	1105
700	1100
800	1115
900	778
1000	1099
1100	1098
1200	1126
1300	1117
1400	1131
1500	1014
1600	1125
1700	1132
1800	1113
1900	1089
2000	1103
2100	825
2200	1134
2300	1117
2400	1121
2500	829
2600	1129

2700	1128
2800	1128
2900	1125
2957	1134
3057	1002
3157	1122
3257	1115
3357	1092
3457	1034
3557	1126
3657	1134
3757	1069
3857	1132
3957	1121
4057	1125
4141	1117
4241	825
4341	1131
4441	1137
4541	1115
4641	1132
4741	1108
4841	1129
4941	1131
5041	974
5141	1101
5241	1136
5341	1124
5441	1035
5541	1129
5620	1129
5720	1107
5820	1136
5920	1129
6020	1136
6120	1118
6220	1127
6233	1130
6333	1125
6433	782
6533	946
6633	1132
6733	1123
6833	1136
6846	1125
6946	1133
7046	1140
7100	1131
7200	1122
7300	1117
7400	1124
7500	1111

7600	1126
7700	1134
7800	1139
7900	1078
8000	1125
8100	1115
8200	1134
8283	1132
8383	1067
8483	994
8583	1071
8683	1133
8774	938
8874	1131
8974	777
9074	1131
9174	1130
9274	1136
9374	1125
9386	1135
9486	1113
9586	1123
9640	804
9740	1131
9840	1123
9894	1136
9994	1136

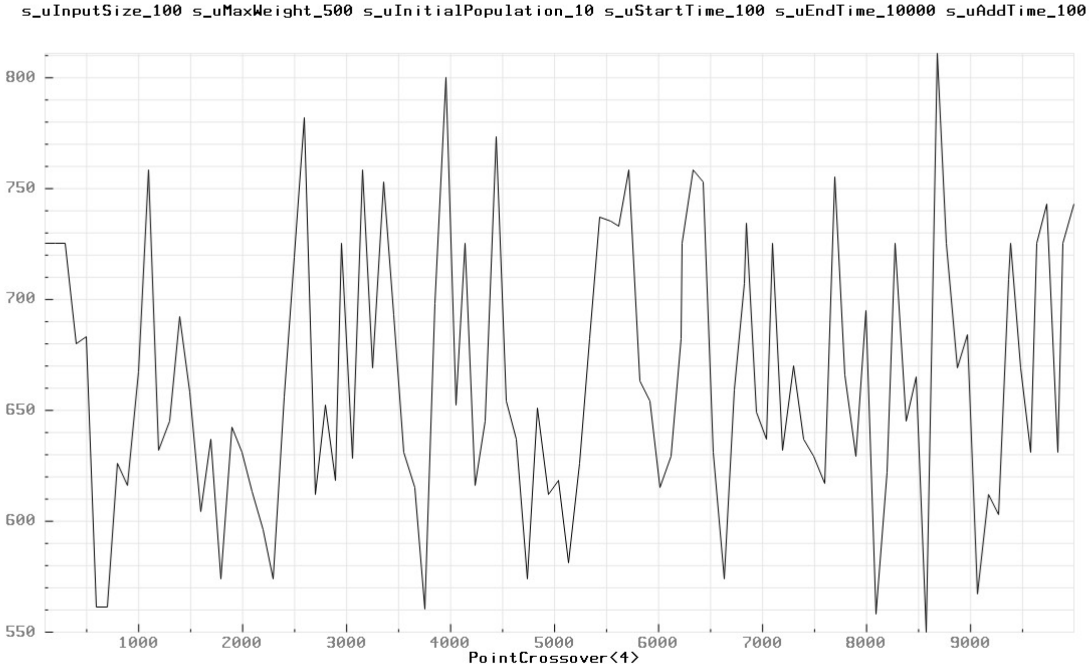


Рисунок 3.7 - PointCrossover<4>

100	725
200	725

300	725
400	680
500	683
600	561
700	561
800	626
900	616
1000	667
1100	758
1200	632
1300	645
1400	692
1500	658
1600	604
1700	637
1800	574
1900	642
2000	631
2100	613
2200	596
2300	574
2400	656
2500	719
2600	782
2700	612
2800	652
2900	618
2957	725
3057	628
3157	758
3257	669
3357	753
3457	693
3557	631
3657	615
3757	560
3857	698
3957	800
4057	652
4141	725
4241	616
4341	645
4441	773
4541	654
4641	637
4741	574
4841	651
4941	612
5041	618
5141	581
5241	626
5341	682

5441	737
5541	735
5620	733
5720	758
5820	663
5920	654
6020	615
6120	629
6220	682
6233	725
6333	758
6433	753
6533	631
6633	574
6733	659
6833	707
6846	734
6946	649
7046	637
7100	725
7200	632
7300	670
7400	637
7500	629
7600	617
7700	755
7800	666
7900	629
8000	695
8100	558
8200	622
8283	725
8383	645
8483	665
8583	550
8683	811
8774	725
8874	669
8974	684
9074	567
9174	612
9274	603
9374	709
9386	725
9486	669
9586	631
9640	725
9740	743
9840	631
9894	725
9994	743

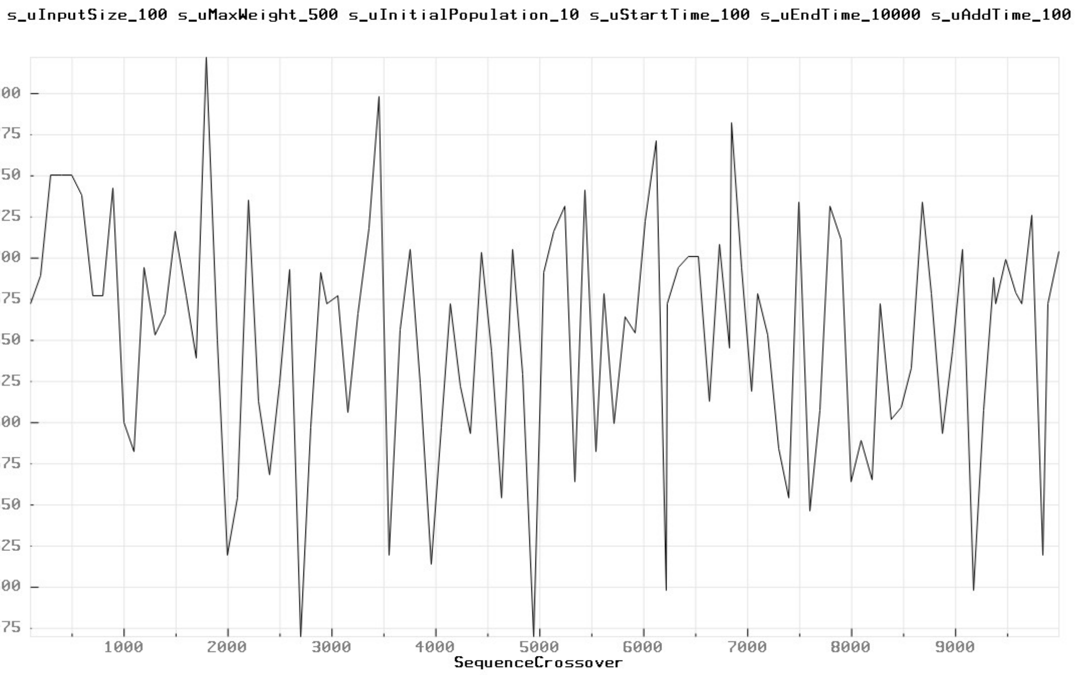


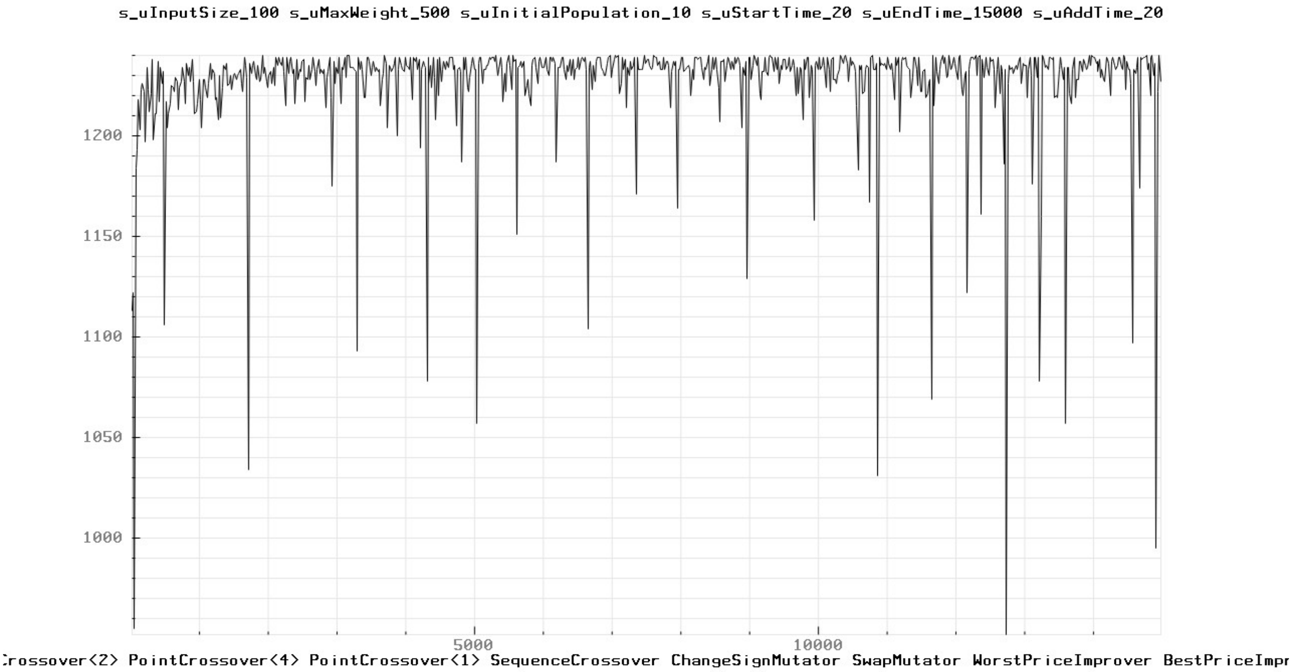
Рисунок 3.8 - SequenceCrossover

100	672
200	689
300	750
400	750
500	750
600	738
700	677
800	677
900	742
1000	600
1100	582
1200	694
1300	653
1400	666
1500	716
1600	677
1700	639
1800	822
1900	650
2000	519
2100	554
2200	735
2300	613
2400	568
2500	624
2600	693
2700	470
2800	597
2900	691
2957	672

3057	677
3157	606
3257	666
3357	718
3457	798
3557	519
3657	656
3757	705
3857	624
3957	514
4057	597
4141	672
4241	622
4341	593
4441	703
4541	643
4641	554
4741	705
4841	629
4941	470
5041	691
5141	716
5241	731
5341	564
5441	741
5541	582
5620	678
5720	599
5820	664
5920	654
6020	722
6120	771
6220	498
6233	672
6333	694
6433	701
6533	701
6633	613
6733	708
6833	645
6846	782
6946	694
7046	619
7100	678
7200	653
7300	584
7400	554
7500	734
7600	546
7700	607
7800	731
7900	711

8000	564
8100	589
8200	565
8283	672
8383	602
8483	609
8583	633
8683	734
8774	678
8874	593
8974	643
9074	705
9174	498
9274	607
9374	688
9386	672
9486	699
9586	679
9640	672
9740	726
9840	519
9894	672
9994	704

Фінальний тест:



20	1113
40	1122
60	955
80	1184

100	1193
120	1218
140	1203
160	1223
180	1226
200	1222
220	1197
240	1234
260	1223
280	1212
300	1221
320	1238
340	1198
360	1211
380	1211
400	1237
420	1217
440	1234
460	1226
480	1232
500	1106
520	1217
540	1204
560	1211
580	1216
600	1225
620	1224
640	1222
660	1229
680	1227
700	1213
720	1224
740	1227
760	1234
780	1227
800	1216
820	1236
840	1230
860	1234
880	1227
900	1238
920	1227
940	1211
960	1212
980	1216
1000	1228
1020	1218
1040	1204
1060	1228
1080	1228
1100	1224
1120	1219

1140	1226
1160	1236
1180	1228
1200	1233
1220	1231
1240	1218
1260	1219
1280	1208
1300	1230
1320	1209
1340	1224
1360	1235
1380	1233
1400	1236
1420	1225
1440	1226
1460	1230
1480	1223
1500	1230
1520	1231
1540	1228
1560	1230
1580	1231
1600	1233
1620	1226
1640	1222
1660	1239
1680	1235
1700	1229
1720	1034
1740	1236
1760	1231
1780	1235
1800	1237
1820	1230
1840	1228
1860	1235
1880	1227
1900	1228
1920	1240
1940	1231
1960	1234
1980	1227
2000	1224
2020	1230
2040	1232
2060	1226
2080	1234
2100	1225
2120	1239
2140	1235
2160	1239

2180	1237
2200	1235
2220	1239
2240	1232
2260	1215
2280	1239
2300	1231
2320	1237
2340	1239
2360	1231
2380	1233
2400	1216
2420	1238
2440	1228
2460	1233
2480	1232
2500	1237
2520	1238
2540	1217
2560	1229
2580	1228
2600	1235
2620	1228
2640	1233
2660	1239
2680	1232
2700	1225
2720	1233
2740	1238
2760	1231
2780	1233
2800	1231
2820	1233
2840	1214
2860	1229
2880	1232
2900	1239
2920	1219
2940	1175
2960	1237
2980	1226
3000	1234
3020	1231
3040	1239
3060	1216
3080	1234
3100	1233
3120	1239
3140	1229
3160	1240
3180	1240
3200	1234

3220	1234
3240	1234
3260	1233
3280	1232
3300	1093
3320	1239
3340	1233
3360	1237
3380	1233
3400	1219
3420	1219
3440	1239
3460	1239
3480	1237
3500	1230
3520	1234
3540	1237
3560	1232
3580	1234
3600	1232
3620	1233
3640	1215
3660	1228
3680	1231
3700	1240
3720	1236
3740	1204
3760	1235
3780	1234
3800	1232
3820	1234
3840	1236
3860	1239
3880	1200
3900	1233
3920	1239
3940	1235
3960	1233
3980	1232
4000	1239
4020	1235
4040	1234
4060	1233
4080	1239
4100	1218
4120	1239
4140	1237
4160	1239
4180	1233
4200	1238
4220	1194
4240	1233

4260	1237
4280	1235
4300	1234
4320	1078
4340	1239
4343	1226
4363	1235
4383	1224
4403	1239
4423	1229
4443	1208
4463	1238
4483	1220
4503	1239
4523	1227
4543	1233
4563	1237
4583	1233
4603	1237
4623	1234
4643	1239
4663	1239
4683	1239
4703	1233
4723	1235
4743	1205
4763	1233
4783	1234
4803	1232
4823	1187
4843	1239
4863	1233
4883	1239
4903	1224
4923	1222
4943	1229
4963	1236
4983	1239
5003	1233
5023	1233
5043	1057
5063	1236
5083	1239
5103	1239
5123	1226
5143	1233
5163	1233
5183	1235
5203	1235
5223	1239
5243	1238
5263	1239

5283	1239
5303	1237
5323	1240
5343	1230
5363	1233
5383	1239
5403	1234
5423	1217
5443	1231
5463	1222
5483	1236
5503	1240
5523	1235
5543	1223
5563	1240
5583	1237
5603	1238
5623	1151
5643	1234
5663	1235
5683	1237
5703	1234
5723	1237
5743	1220
5763	1225
5783	1228
5803	1221
5823	1215
5843	1235
5863	1238
5883	1239
5903	1234
5923	1226
5943	1237
5963	1239
5983	1234
6003	1234
6023	1234
6043	1239
6063	1233
6083	1230
6103	1239
6123	1239
6135	1237
6155	1239
6175	1236
6195	1187
6215	1234
6235	1234
6255	1234
6275	1239
6295	1239

6315	1233
6335	1228
6355	1235
6375	1232
6395	1238
6415	1230
6435	1238
6455	1228
6475	1233
6495	1234
6515	1239
6535	1233
6555	1238
6575	1239
6595	1237
6615	1239
6635	1233
6655	1104
6675	1234
6695	1237
6715	1223
6735	1235
6755	1227
6775	1237
6795	1239
6815	1239
6835	1239
6855	1234
6875	1232
6895	1230
6915	1230
6935	1236
6955	1239
6975	1240
6995	1229
7015	1233
7035	1238
7055	1240
7075	1239
7095	1240
7115	1221
7135	1226
7155	1239
7175	1233
7195	1238
7215	1214
7235	1234
7255	1230
7275	1237
7295	1233
7315	1235
7335	1239

7355	1171
7375	1239
7395	1233
7415	1233
7435	1233
7455	1233
7475	1240
7495	1236
7515	1238
7535	1238
7555	1235
7575	1233
7595	1234
7615	1240
7635	1239
7655	1239
7675	1235
7695	1237
7715	1233
7735	1233
7755	1239
7775	1233
7795	1237
7815	1237
7835	1235
7855	1214
7875	1232
7895	1237
7915	1235
7935	1237
7955	1164
7975	1233
7995	1233
8015	1239
8035	1239
8055	1232
8075	1233
8095	1234
8115	1235
8135	1233
8155	1220
8175	1233
8195	1228
8215	1239
8235	1239
8255	1238
8275	1239
8295	1239
8315	1234
8335	1228
8355	1234
8372	1233

8392	1237
8412	1240
8432	1225
8452	1233
8472	1239
8492	1233
8512	1239
8532	1233
8552	1224
8572	1207
8592	1237
8612	1237
8632	1238
8652	1227
8672	1235
8692	1239
8712	1238
8732	1233
8752	1237
8772	1233
8792	1239
8812	1236
8832	1239
8852	1238
8872	1235
8892	1204
8912	1234
8932	1230
8952	1237
8972	1129
8992	1239
9012	1239
9032	1228
9052	1235
9072	1240
9092	1239
9112	1235
9132	1239
9152	1221
9172	1218
9192	1232
9212	1232
9232	1239
9252	1232
9272	1230
9292	1239
9300	1240
9320	1239
9340	1239
9360	1232
9380	1239
9400	1233

9420	1239
9440	1226
9460	1235
9480	1233
9500	1238
9520	1238
9540	1233
9560	1235
9580	1239
9600	1232
9620	1235
9640	1238
9660	1235
9680	1220
9700	1234
9720	1221
9740	1234
9760	1233
9780	1208
9800	1238
9820	1233
9840	1239
9860	1239
9880	1219
9900	1235
9920	1233
9940	1158
9960	1235
9980	1237
10000	1233
10020	1239
10040	1233
10060	1235
10080	1239
10100	1235
10120	1224
10140	1239
10160	1233
10180	1222
10200	1238
10220	1239
10227	1228
10247	1228
10267	1226
10287	1230
10307	1233
10327	1240
10347	1239
10367	1236
10387	1239
10407	1240
10427	1236

10447	1227
10467	1239
10487	1235
10507	1237
10527	1235
10547	1232
10567	1215
10587	1183
10607	1235
10612	1233
10632	1235
10652	1221
10672	1222
10692	1233
10712	1235
10732	1234
10752	1167
10772	1239
10792	1240
10812	1233
10832	1233
10852	1236
10872	1031
10892	1239
10912	1235
10932	1236
10952	1235
10972	1236
10992	1233
11012	1239
11032	1236
11052	1233
11072	1236
11092	1232
11112	1218
11132	1239
11152	1235
11172	1239
11192	1202
11212	1237
11232	1239
11252	1238
11272	1239
11292	1235
11312	1234
11332	1233
11352	1219
11372	1229
11392	1239
11412	1239
11432	1233
11452	1225

11472	1239
11492	1222
11512	1222
11532	1232
11552	1233
11572	1219
11592	1221
11612	1226
11632	1228
11652	1069
11672	1240
11692	1215
11712	1239
11732	1232
11752	1226
11772	1233
11792	1237
11812	1240
11832	1236
11852	1239
11872	1229
11892	1230
11912	1240
11932	1233
11952	1234
11972	1239
11992	1236
12012	1233
12032	1228
12052	1239
12072	1240
12092	1223
12112	1220
12132	1230
12152	1232
12172	1122
12192	1233
12212	1237
12232	1239
12252	1239
12272	1238
12292	1240
12312	1230
12332	1239
12352	1240
12372	1161
12392	1238
12403	1234
12423	1229
12443	1239
12463	1234
12483	1230

12503	1239
12523	1237
12543	1238
12563	1233
12583	1214
12603	1233
12623	1239
12643	1221
12663	1235
12683	1232
12703	1186
12723	1233
12743	952
12763	1226
12783	1235
12803	1233
12823	1234
12843	1231
12863	1234
12883	1235
12903	1233
12923	1238
12943	1239
12963	1226
12983	1236
13003	1235
13023	1235
13043	1219
13063	1233
13083	1239
13103	1230
13123	1176
13143	1239
13145	1233
13165	1239
13185	1229
13205	1239
13225	1078
13245	1160
13265	1240
13285	1239
13305	1234
13325	1235
13345	1233
13365	1239
13385	1239
13405	1238
13425	1239
13445	1219
13465	1220
13485	1219
13505	1233

13525	1239
13545	1235
13565	1228
13585	1234
13605	1057
13625	1234
13645	1234
13665	1221
13685	1216
13705	1233
13725	1239
13745	1219
13765	1228
13785	1228
13805	1237
13825	1237
13845	1232
13865	1238
13885	1235
13905	1233
13925	1234
13945	1240
13965	1235
13985	1240
14005	1239
14025	1239
14045	1236
14065	1237
14072	1232
14092	1238
14112	1229
14132	1233
14152	1228
14172	1227
14192	1239
14212	1238
14232	1236
14252	1220
14272	1237
14292	1239
14312	1235
14332	1233
14352	1235
14372	1239
14392	1239
14412	1234
14432	1233
14452	1240
14456	1239
14476	1223
14496	1237
14516	1233

14536	1233
14556	1226
14576	1097
14596	1233
14616	1231
14636	1234
14656	1239
14676	1174
14696	1239
14716	1238
14736	1239
14756	1239
14776	1240
14796	1233
14816	1236
14836	1220
14840	1239
14860	1240
14880	1230
14900	1236
14920	995
14940	1222
14960	1240
14980	1227

Висновок

Під час лабораторної роботи вивчив основні підходи розробки метаевристичних алгоритмів для типових прикладних задач на прикладі задачі про рюкзак реалізованою за допомогою генетичного алгоритму. Опрацювати методологію підбору прийнятних параметрів алгоритму, де результатами є графіки та таблиці , що генеруються програмою. Як можна побачити за допомогою лише кросоверу високі результати не досягаються, але якщо додати ще й мутації, то зміни видно відразу і дають приріст у 2, а то і в 3 рази. Отже, важливим еволюційним фактором поколінь є не тільки їх змішування між собою, а й мутації, що мають виникати внаслідок пристосування до навколишнього середовища. Також додано локальне покращення, але приріст воно дає незначне, якщо і взагалі дає.