

# Operational Research Library 1.0

manuál pro použití knihovny operačního výzkumu

## OBSAH

<b>1</b>	<b>KNIHOVNA OPERAČNÍHO VÝZKUMU .....</b>	<b>3</b>
1.1	ÚVOD .....	3
1.2	OBSAH KNIHOVNY .....	3
1.2.1	Hlavičkové soubory (C++ Header files) .....	3
1.2.2	Zdrojové soubory (C++ Source files) .....	4
<b>2</b>	<b>POUŽITÍ PROBLÉMŮ KNIHOVNY OPERAČNÍHO VÝZKUMU .....</b>	<b>5</b>
2.1	PROBLÉM BATOHU .....	5
2.1.1	Ukázka použití .....	6
2.2	PROBLÉM NAPLŇOVÁNÍ ZÁSOBNÍKU .....	8
2.2.1	Ukázka použití .....	10
2.3	LINEÁRNÍ PŘÍRAZOVACÍ PROBLÉM .....	12
2.3.1	Ukázka použití .....	14
2.4	KVADRATICKÝ PŘÍRAZOVACÍ PROBLÉM .....	15
2.4.1	Ukázka použití .....	17
2.5	PROBLÉM ROZVRHOVÁNÍ PROUDOVÉ VÝROBY .....	20
2.5.1	Ukázka použití .....	22
2.6	PROBLÉM OBCHODNÍHO CESTUJÍCÍHO .....	24
2.6.1	Ukázka použití .....	26
2.7	KAPACITNÍ ROZVOZNÍ PROBLÉM .....	27
2.7.1	Ukázka použití .....	29

# 1 KNIHOVNA OPERAČNÍHO VÝZKUMU

## 1.1 Úvod

Knihovna operačního výzkumu (dále jen knihovna), realizována v programovém prostředí C++, implementuje vybrané problémy operačního výzkumu (dále jen problémy). Návrh a následná realizace knihovny vychází z požadavků, které jsou na knihovnu kladeny. Jednotlivé problémy jsou zpracovány ve formě hodnotových (účelových) funkcí tak, aby mohly být v budoucnu podrobeny testům pomocí různých algoritmů.

## 1.2 Obsah knihovny

V této části je uveden přehled souborů knihovny. Celá knihovna se nachází v prostoru jmen *operational\_research*. Podrobnější informace jsou uvedeny v programové dokumentaci knihovny.

### 1.2.1 Hlavičkové soubory (C++ Header files)

Přehled hlavičkových souborů knihovny včetně odpovídajícího popisu je uveden v tabulce (Tab. 1).

Tab. 1. Hlavičkové soubory knihovny

Hlavičkový soubor	Popis
<i>KnapsackProblem.h</i>	hlavičkový soubor problému batohu
<i>BinPackingProblem.h</i>	hlavičkový soubor problému naplňování zásobníku
<i>LinearAssignmentProblem.h</i>	hlavičkový soubor lineárního přiřazovacího problému
<i>QuadraticAssignmentProblem.h</i>	hlavičkový soubor kvadratického přiřazovacího problému
<i>FlowShopSchedulingProblem.h</i>	hlavičkový soubor problému rozvrhování proudové výroby
<i>TravellingSalesmanProblem.h</i>	hlavičkový soubor problému obchodního cestujícího
<i>CapacitatedVehicleRoutingProblem.h</i>	hlavičkový soubor kapacitního rozvozního problému
<i>CoordinatesHelper.h</i>	hlavičkový soubor pomocné třídy pro

	souřadnice
<i>ScheduleHelper.h</i>	hlavičkový soubor pomocné třídy pro práci s plánem problému
<i>RandomHelper.h</i>	hlavičkový soubor pomocné třídy pro generování pseudonáhodných hodnot
<i>IData.h</i>	hlavičkový soubor rozhraní <i>IData</i>

### 1.2.2 Zdrojové soubory (C++ Source files)

Přehled zdrojových souborů knihovny včetně odpovídajícího popisu je uveden v tabulce (Tab. 2).

Tab. 2. Zdrojové soubory knihovny

<b>Zdrojový soubor</b>	<b>Popis</b>
<i>KnapsackProblem.cpp</i>	implementace problému batohu
<i>BinPackingProblem.cpp</i>	implementace problému naplňování zásobníku
<i>LinearAssignmentProblem.cpp</i>	implementace lineárního přiřazovacího problému
<i>QuadraticAssignmentProblem.cpp</i>	implementace kvadratického přiřazovacího problému
<i>FlowShopSchedulingProblem.cpp</i>	implementace problému rozvrhování proudové výroby
<i>TravellingSalesmanProblem.cpp</i>	implementace problému obchodního cestujícího
<i>CapacitatedVehicleRoutingProblem.cpp</i>	implementace kapacitního rozvozního problému
<i>CoordinatesHelper.cpp</i>	implementace pomocné třídy pro souřadnice
<i>ScheduleHelper.cpp</i>	implementace pomocné třídy pro práci s plánem problému
<i>RandomHelper.cpp</i>	implementace pomocné třídy pro generování pseudonáhodných hodnot

## 2 POUŽITÍ PROBLÉMU KNIHOVNY OPERAČNÍHO VÝZKUMU

V této části je uveden způsob použití jednotlivých problémů implementovaných v knihovně. Kromě stručné charakteristiky jednotlivých metod problémů je u každého problému uvedena také funkční ukázka.

### 2.1 Problém batohu

Pro využití problému batohu je třeba přidat do projektu hlavičkový soubor *KnapsackProblem.h* a zdrojový soubor *KnapsackProblem.cpp*. Podrobné informace o třídě *KnapsackProblem* jsou dostupné v programové dokumentaci knihovny.

V následujícím popisu budou uvažovány proměnné

<code>datasetFilename</code>	název souboru pro import/export dat problému
<code>scheduleFilename</code>	název souboru pro import/export plánu problému
<code>resultFilename</code>	název souboru pro export výstupu problému
<code>objects</code>	počet objektů
<code>capacity</code>	kapacita batohu
<code>minWeight</code>	minimální hmotnost předmětu
<code>maxWeight</code>	maximální hmotnost předmětu
<code>minProfit</code>	minimální cena předmětu
<code>maxProfit</code>	maximální cena předmětu
<code>integerValues</code>	typ generovaných dat
<code>weights</code>	hmotnosti předmětů
<code>profits</code>	ceny předmětů
<code>schedule</code>	plán problému

Vytvoření instance třídy *KnapsackProblem* je možné pomocí konstruktoru, který je dostupný ve 3 variantách.

```
// constructors
KnapsackProblem kp(objects, weights, profits, capacity);
KnapsackProblem kp(objects, capacity, minWeight, maxWeight, minProfit, maxProfit, integerValues);
KnapsackProblem kp(datasetFilename);
```

První varianta konstruktoru zajišťuje volání metody *KnapsackProblem::setData*, druhá varianta konstruktoru zajišťuje volání metody *KnapsackProblem::generateData*, třetí varianta konstruktoru zajišťuje volání metody *KnapsackProblem::readDataFromFile*.

Pro nastavení dat problému je dostupná metoda *KnapsackProblem::setData*.

```
// set data
kp.setData(objects, weights, profits, capacity);
```

Pro generování dat problému je dostupná metoda *KnapsackProblem::generateData*.

```
// generate data
kp.generateData(objects, capacity, minWeight, maxWeight, minProfit, maxProfit, integerValues);
```

Pro import/export dat problému ze/do souboru jsou dostupné metody *KnapsackProblem::readDataFromFile* a *KnapsackProblem::writeDataToFile*.

```
// read data from file
kp.readDataFromFile(datasetFilename);

// write data to file
kp.writeDataToFile(datasetFilename);
```

Pro zobrazení dat problému do konzole je dostupná metoda *KnapsackProblem::showData*.

```
// show data
kp.showData();
```

Pro stanovení hodnoty účelové funkce problému batohu slouží metoda *KnapsackProblem::evaluateKP*, která je dostupná ve 2 variantách.

```
// evaluate knapsack problem (KP)
std::cout << "KP cost function value: " << kp.evaluateKP(schedule)
<< std::endl;

// evaluate knapsack problem (KP) to file
std::cout << "KP cost function value: " << kp.evaluateKP(schedule,
resultFilename) << std::endl;
```

První varianta metody slouží pro stanovení hodnoty účelové funkce pro daný plán, druhá varianta je obdoba první varianty, která navíc zajistí uložení výstupu (plán + odpovídající hodnota účelové funkce) do souboru.

### 2.1.1 Ukázka použití

```
1 // knapsack problem header
2 #include "KnapsackProblem.h"
```

```

3
4 // schedule helper header
5 #include "ScheduleHelper.h"
6
7 // whole operational research library is in namespace operatio-
8 nal_research
9 using namespace operational_research;
10
11 int main(int argc, char **argv)
12 {
13     // set precision
14     std::cout.precision(std::numeric_limits< double >::digits10 +
15 2);
16
17     /***** KNAPSACK PROBLEM *****/
18     // dataset filename
19     std::string datasetFilename = "dataset.txt";
20
21     // schedule filename
22     std::string scheduleFilename = "schedule.txt";
23
24     // result filename
25     std::string resultFilename = "result.txt";
26
27     // number of objects
28     unsigned int objects = 5;
29
30     // knapsack capacity
31     double capacity = 10;
32
33     // minimal and maximal weight of object
34     double minWeight = 1, maxWeight = 5;
35
36     // minimal and maximal profit of object
37     double minProfit = 1, maxProfit = 20;
38
39     // generate integer values?
40     bool integerValues = true;
41
42     // weights of objects
43     std::vector<double> weights = {1,4,2,3,5};
44
45     // profits of objects
46     std::vector<double> profits = {3,4,7,1,2};
47
48     // schedule
49     std::vector<unsigned int> schedule = {1,2};
50
51     try {
52
53         ScheduleHelper scheduleHelper;
54         schedule = scheduleHel-
55 per.generateSelectionSchedule(objects);
56 //         schedule = scheduleHel-
57 per.readScheduleFromFile(scheduleFilename);
58 //         scheduleHelper.writeScheduleToFile(schedule, schedule-

```

```

60 Filename);
61
62         // constructors
63 //         KnapsackProblem kp(objects, weights, profits, capaci-
64 ty);
65         KnapsackProblem kp(objects, capacity, minWeight, ma-
66 xWeight, minProfit, maxProfit, integerValues);
67 //         KnapsackProblem kp(datasetFilename);
68
69         // set data
70 //         kp.setData(objects, weights, profits, capacity);
71
72         // generate data
73 //         kp.generateData(objects, capacity, minWeight, ma-
74 xWeight, minProfit, maxProfit, integerValues);
75
76         // read data from file
77 //         kp.readDataFromFile(datasetFilename);
78
79         // write data to file
80 //         kp.writeDataToFile(datasetFilename);
81
82         // show data
83         kp.showData();
84
85         // evaluate knapsack problem (KP)
86         std::cout << "KP cost function value: " <<
kp.evaluateKP(schedule) << std::endl;
87
88         // evaluate knapsack problem (KP) to file
89 //         std::cout << "KP cost function value: " <<
90 kp.evaluateKP(schedule, resultFilename) << std::endl;
91
92         } catch (std::exception& e) {
93             std::cout << e.what();
94         }
95     }

```

## 2.2 Problém naplňování zásobníku

Pro využití problému naplňování zásobníku je třeba přidat do projektu hlavičkový soubor *BinPackingProblem.h* a zdrojový soubor *BinPackingProblem.cpp*. Podrobné informace o třídě *BinPackingProblem* jsou dostupné v programové dokumentaci knihovny.

V následujícím popisu budou uvažovány proměnné

<code>datasetFilename</code>	název souboru pro import/export dat problému
<code>scheduleFilename</code>	název souboru pro import/export plánu problému
<code>resultFilename</code>	název souboru pro export výstupu problému
<code>objects</code>	počet objektů



capacity	kapacita batohu
minWeight	minimální hmotnost předmětu
maxWeight	maximální hmotnost předmětu
integerValues	typ generovaných dat
weights	hmotnosti předmětů
schedule	plán problému

Vytvoření instance třídy *BinPackingProblem* je možné pomocí konstruktoru, který je dostupný ve 3 variantách.

```
// constructors
BinPackingProblem bpp(objects, weights, capacity);
BinPackingProblem bpp(objects, capacity, minWeight, maxWeight, integerValues);
BinPackingProblem bpp(datasetFilename);
```

První varianta konstruktoru zajišťuje volání metody *BinPackingProblem::setData*, druhá varianta konstruktoru zajišťuje volání metody *BinPackingProblem::generateData*, třetí varianta konstruktoru zajišťuje volání metody *BinPackingProblem::readDataFromFile*.

Pro nastavení dat problému je dostupná metoda *BinPackingProblem::setData*.

```
// set data
bpp.setData(objects, weights, capacity);
```

Pro generování dat problému je dostupná metoda *BinPackingProblem::generateData*.

```
// generate data
bpp.generateData(objects, capacity, minWeight, maxWeight, integerValues);
```

Pro import/export dat problému ze/do souboru jsou dostupné metody *BinPackingProblem::readDataFromFile* a *BinPackingProblem::writeDataToFile*.

```
// read data from file
bpp.readDataFromFile(datasetFilename);

// write data to file
bpp.writeDataToFile(datasetFilename);
```

Pro zobrazení dat problému do konzole je dostupná metoda *BinPackingProblem::showData*.

```
// show data
bpp.showData();
```

Pro stanovení hodnoty účelové funkce problému batohu slouží metoda *BinPackingProblem::evaluateBPP*, která je dostupná ve 2 variantách.

```
// evaluate bin packing problem (BPP)
std::cout << "BPP cost function value: " <<
bpp.evaluateBPP(schedule) << std::endl;

// evaluate bin packing problem (BPP) to file
std::cout << "BPP cost function value: " <<
bpp.evaluateBPP(schedule, resultFilename) << std::endl;
```

První varianta metody slouží pro stanovení hodnoty účelové funkce pro daný plán, druhá varianta je obdoba první varianty, která navíc zajistí uložení výstupu (plán + odpovídající hodnota účelové funkce) do souboru.

### 2.2.1 Ukázka použití

```
1 // bin packing problem header
2 #include "BinPackingProblem.h"
3
4 // schedule helper header
5 #include "ScheduleHelper.h"
6
7 // whole operational research library is in namespace operatio-
8 nal_research
9 using namespace operational_research;
10
11 int main(int argc, char **argv)
12 {
13     // set precision
14     std::cout.precision(std::numeric_limits< double >::digits10 +
15 2);
16
17     /***** BIN PACKING PROBLEM *****/
18     *****/
19     // dataset filename
20     std::string datasetFilename = "dataset.txt";
21
22     // schedule filename
23     std::string scheduleFilename = "schedule.txt";
24
25     // result filename
26     std::string resultFilename = "result.txt";
27
28     // number of objects
29     unsigned int objects = 5;
30
31     // bin capacity
32     double capacity = 10;
33
34     // minimal and maximal weight of object
35     double minWeight = 1, maxWeight = 10;
36
```

```

37         // generate integer values?
38         bool integerValues = true;
39
40         // weights of objects
41         std::vector<double> weights = {1,4,2,3,5};
42
43         // schedule
44         std::vector<unsigned int> schedule = {1,2,3,4,5};
45
46         try {
47
48             // schedule generate/read/write
49             ScheduleHelper scheduleHelper;
50             schedule = scheduleHel-
51 per.generatePermutationSchedule(objects);
52 //             schedule = scheduleHel-
53 per.readScheduleFromFile(scheduleFilename);
54 //             scheduleHelper.writeScheduleToFile(schedule, schedule-
55 Filename);
56
57             // constructors
58 //             BinPackingProblem bpp(objects, weights, capacity);
59             BinPackingProblem bpp(objects, capacity, minWeight,
60 maxWeight, integerValues);
61 //             BinPackingProblem bpp(datasetFilename);
62
63             // set data
64 //             bpp.setData(objects, weights, capacity);
65
66             // generate data
67 //             bpp.generateData(objects, capacity, minWeight, ma-
68 xWeight, integerValues);
69
70             // read data from file
71 //             bpp.readDataFromFile(datasetFilename);
72
73             // write data to file
74 //             bpp.writeDataToFile(datasetFilename);
75
76             // show data
77             bpp.showData();
78
79             // evaluate bin packing problem (BPP)
80             std::cout << "BPP cost function value: " <<
bpp.evaluateBPP(schedule) << std::endl;
//             // evaluate bin packing problem (BPP) to file
//             std::cout << "BPP cost function value: " <<
bpp.evaluateBPP(schedule, resultFilename) << std::endl;
        } catch (std::exception& e) {
            std::cout << e.what();
        }
    }

```

## 2.3 Lineární přiřazovací problém

Pro využití lineárního přiřazovacího problému je třeba přidat do projektu hlavičkový soubor *LinearAssignmentProblem.h* a zdrojový soubor *LinearAssignmentProblem.cpp*. Podrobné informace o třídě *LinearAssignmentProblem* jsou dostupné v programové dokumentaci knihovny.

V následujícím popisu budou uvažovány proměnné

<code>datasetFilename</code>	název souboru pro import/export dat problému
<code>scheduleFilename</code>	název souboru pro import/export plánu problému
<code>resultFilename</code>	název souboru pro export výstupu problému
<code>objects</code>	počet objektů
<code>minCost</code>	minimální cena přiřazení
<code>maxCost</code>	maximální cena přiřazení
<code>integerValues</code>	typ generovaných dat
<code>costs</code>	ceny přiřazení
<code>schedule</code>	plán problému

Vytvoření instance třídy *LinearAssignmentProblem* je možné pomocí konstruktoru, který je dostupný ve 3 variantách.

```
// constructors
LinearAssignmentProblem lap(objects, costs);
LinearAssignmentProblem lap(objects, minCost, maxCost, integerValues);
LinearAssignmentProblem lap(datasetFilename);
```

První varianta konstruktoru zajišťuje volání metody *LinearAssignmentProblem::setData*, druhá varianta konstruktoru zajišťuje volání metody *LinearAssignmentProblem::generateData*, třetí varianta konstruktoru zajišťuje volání metody *LinearAssignmentProblem::readDataFromFile*.

Pro nastavení dat problému je dostupná metoda *LinearAssignmentProblem::setData*.

```
// set data
lap.setData(objects, costs);
```

Pro generování dat problému je dostupná metoda *LinearAssignmentProblem::generateData*.

```
// generate data
lap.generateData(objects, minCost, maxCost, integerValues);
```

Pro import/export dat problému ze/do souboru jsou dostupné metody *LinearAssignmentProblem::readDataFromFile* a *LinearAssignmentProblem::writeDataToFile*.

```
// read data from file
lap.readDataFromFile(datasetFilename);

// write data to file
lap.writeDataToFile(datasetFilename);
```

Pro zobrazení dat problému do konzole je dostupná metoda *LinearAssignmentProblem::showData*.

```
// show data
lap.showData();
```

Pro stanovení hodnoty účelové funkce lineárního součtového přiřazovacího problému slouží metoda *LinearAssignmentProblem::evaluateLSAP*, která je dostupná ve 2 variantách.

```
// evaluate linear sum assignment problem (LSAP)
std::cout << "LSAP cost function value: " <<
lap.evaluateLSAP(schedule) << std::endl;

// evaluate linear sum assignment problem (LSAP) to file
std::cout << "LSAP cost function value: " <<
lap.evaluateLSAP(schedule, resultFilename) << std::endl;
```

Pro stanovení hodnoty účelové funkce lineárního úzkoprofilového přiřazovacího problému slouží metoda *LinearAssignmentProblem::evaluateLBAP*, která je dostupná ve 2 variantách.

```
// evaluate linear bottleneck assignment problem (LBAP)
std::cout << "LBAP cost function value: " <<
lap.evaluateLBAP(schedule) << std::endl;

// evaluate linear bottleneck assignment problem (LBAP) to file
std::cout << "LBAP cost function value: " <<
lap.evaluateLBAP(schedule, resultFilename) << std::endl;
```

První varianty metody slouží pro stanovení hodnoty účelové funkce pro daný plán, druhé varianty jsou obdobou první varianty, které navíc zajistí uložení výstupu (plán + odpovídající hodnota účelové funkce) do souboru.

### 2.3.1 Ukázka použití

```
1 // linear assignment problem header
2 #include "LinearAssignmentProblem.h"
3
4 // schedule helper header
5 #include "ScheduleHelper.h"
6
7 // whole operational research library is in namespace operatio-
8 nal_research
9 using namespace operational_research;
10
11 int main(int argc, char **argv)
12 {
13     // set precision
14     std::cout.precision(std::numeric_limits< double >::digits10 +
15 2);
16
17     /***** LINEAR ASSIGNMENT PROBLEM *****/
18     // dataset filename
19     std::string datasetFilename = "dataset.txt";
20
21     // schedule filename
22     std::string scheduleFilename = "schedule.txt";
23
24     // result filename
25     std::string resultFilename = "result.txt";
26
27     // number of objects
28     unsigned int objects = 3;
29
30     // minimal and maximal cost
31     double minCost = 1, maxCost = 10;
32
33     // generate integer values?
34     bool integerValues = true;
35
36     // costs
37     std::vector<std::vector<double> > costs =
38     {{1,3,4},{3,5,2},{4,1,1}};
39
40     // schedule
41     std::vector<unsigned int> schedule = {1,2,3};
42
43     try {
44
45         ScheduleHelper scheduleHelper;
46         schedule = scheduleHel-
47 per.generatePermutationSchedule(objects);
48 //         schedule = scheduleHel-
49 per.readScheduleFromFile(scheduleFilename);
50 //         scheduleHelper.writeScheduleToFile(schedule, schedule-
51 Filename);
52
53
54         // constructors
```

```

55 //          LinearAssignmentProblem lap(objects, costs);
56          LinearAssignmentProblem lap(objects, minCost, maxCost,
57 integerValues);
58 //          LinearAssignmentProblem lap(datasetFilename);
59
60          // set data
61 //          lap.setData(objects, costs);
62
63          // generate data
64 //          lap.generateData(objects, minCost, maxCost, integerVa-
65 lues);
66
67          // read data from file
68 //          lap.readDataFromFile(datasetFilename);
69
70          // write data to file
71 //          lap.writeDataToFile(datasetFilename);
72
73          // show data
74          lap.showData();
75
76          // evaluate linear sum assignment problem (LSAP)
77          std::cout << "LSAP cost function value: " <<
78 lap.evaluateLSAP(schedule) << std::endl;
79
80          // evaluate linear sum assignment problem (LSAP) to
81 file
82 //          std::cout << "LSAP cost function value: " <<
83          lap.evaluateLSAP(schedule, resultFilename) << std::endl;
84
85          // evaluate linear bottleneck assignment problem
86 (LBAP)
87          std::cout << "LBAP cost function value: " <<
88 lap.evaluateLBAP(schedule) << std::endl;
89
90          // evaluate linear bottleneck assignment problem
91 (LBAP) to file
92 //          std::cout << "LBAP cost function value: " <<
93          lap.evaluateLBAP(schedule, resultFilename) << std::endl;
94
95          } catch (std::exception& e) {
96              std::cout << e.what();
97          }
98      }

```

## 2.4 Kvadratický přiřazovací problém

Pro využití kvadratického přiřazovacího problému je třeba přidat do projektu hlavičkový soubor *QuadraticAssignmentProblem.h* a zdrojový soubor *QuadraticAssignmentProblem.cpp*. Podrobné informace o třídě *QuadraticAssignmentProblem* jsou dostupné v programové dokumentaci knihovny.

V následujícím popisu budou uvažovány proměnné

<code>datasetFilename</code>	název souboru pro import/export dat problému
<code>scheduleFilename</code>	název souboru pro import/export plánu problému
<code>resultFilename</code>	název souboru pro export výstupu problému
<code>objects</code>	počet objektů
<code>minFlow</code>	minimální tok
<code>maxFlow</code>	maximální tok
<code>minDistance</code>	minimální vzdálenost
<code>maxDistance</code>	maximální vzdálenost
<code>minCost</code>	minimální cena přiřazení
<code>maxCost</code>	maximální cena přiřazení
<code>integerValues</code>	typ generovaných dat
<code>costs</code>	ceny přiřazení
<code>schedule</code>	plán problému

Vytvoření instance třídy *QuadraticAssignmentProblem* je možné pomocí konstruktoru, který je dostupný ve 3 variantách.

```
// constructors
QuadraticAssignmentProblem qap(objects, flows, distances, costs);
QuadraticAssignmentProblem qap(objects, minFlow, maxFlow, min-
Distance, maxDistance, minCost, maxCost, integerValues);
QuadraticAssignmentProblem qap(datasetFilename);
```

První varianta konstrukturu zajišťuje volání metody *QuadraticAssignmentProblem::setData*, druhá varianta konstrukturu zajišťuje volání metody *QuadraticAssignmentProblem::generateData*, třetí varianta konstrukturu zajišťuje volání metody *QuadraticAssignmentProblem::readDataFromFile*.

Pro nastavení dat problému je dostupná metoda *QuadraticAssignmentProblem::setData*.

```
// set data
qap.setData(objects, flows, distances, costs);
```

Pro generování dat problému je dostupná metoda *QuadraticAssignmentProblem::generateData*.

```
// generate data
```



```
qap.generateData(objects, minFlow, maxFlow, minDistance, max-
Distance, minCost, maxCost, integerValues);
```

Pro import/export dat problému ze/do souboru jsou dostupné metody *QuadraticAssignmentProblem::readDataFromFile* a *QuadraticAssignmentProblem::writeDataToFile*.

```
// read data from file
qap.readDataFromFile(datasetFilename);

// write data to file
qap.writeDataToFile(datasetFilename);
```

Pro zobrazení dat problému do konzole je dostupná metoda *QuadraticAssignmentProblem::showData*.

```
// show data
qap.showData();
```

Pro stanovení hodnoty účelové funkce kvadratického přiřazovacího problému slouží metoda *QuadraticAssignmentProblem::evaluateQAP*, která je dostupná ve 2 variantách.

```
// evaluate quadratic assignment problem (QAP)
std::cout << "QAP cost function value: " <<
qap.evaluateQAP(schedule) << std::endl;

// evaluate quadratic assignment problem (QAP) to file
std::cout << "QAP cost function value: " <<
qap.evaluateQAP(schedule, resultFilename) << std::endl;
```

Pro stanovení hodnoty účelové funkce kvadratického úzkoprofilového přiřazovacího problému slouží metoda *QuadraticAssignmentProblem::evaluateQBAP*, která je dostupná ve 2 variantách.

```
// evaluate quadratic bottleneck assignment problem (QBAP)
std::cout << "QBAP cost function value: " <<
qap.evaluateQBAP(schedule) << std::endl;

// evaluate quadratic bottleneck assignment problem (QBAP) to file
std::cout << "QBAP cost function value: " <<
qap.evaluateQBAP(schedule, resultFilename) << std::endl;
```

První varianty metody slouží pro stanovení hodnoty účelové funkce pro daný plán, druhé varianty jsou obdobou první varianty, které navíc zajistí uložení výstupu (plán + odpovídající hodnota účelové funkce) do souboru.

## 2.4.1 Ukázka použití

```
1 // quadratic assignment problem header
2 #include "QuadraticAssignmentProblem.h"
```

```

3
4 // schedule helper header
5 #include "ScheduleHelper.h"
6
7 // whole operational research library is in namespace operatio-
8 nal_research
9 using namespace operational_research;
10
11 int main(int argc, char **argv)
12 {
13     // set precision
14     std::cout.precision(std::numeric_limits< double >::digits10 +
15 2);
16
17     /***** QUADRATIC ASSIGNMENT PROBLEM *****/
18     /*****
19     // dataset filename
20     std::string datasetFilename = "dataset.txt";
21
22     // schedule filename
23     std::string scheduleFilename = "schedule.txt";
24
25     // result filename
26     std::string resultFilename = "result.txt";
27
28     // number of objects
29     unsigned int objects = 3;
30
31     // minimal and maximal flow
32     double minFlow = 1, maxFlow = 10;
33
34     // minimal and maximal distance
35     double minDistance = 1, maxDistance = 10;
36
37     // minimal and maximal cost
38     double minCost = 1, maxCost = 10;
39
40     // generate integer values?
41     bool integerValues = true;
42
43     // flows
44     std::vector<std::vector<double> > flows =
45 {{1,3,4},{3,5,2},{4,1,1}};
46
47     // distances
48     std::vector<std::vector<double> > distances =
49 {{1,3,4},{3,5,2},{4,1,1}};
50
51     // costs
52     std::vector<std::vector<double> > costs =
53 {{1,3,4},{3,5,2},{4,1,1}};
54
55     // schedule
56     std::vector<unsigned int> schedule = {1,2,3};
57
58     try {
59

```

```

60         ScheduleHelper scheduleHelper;
61         schedule = scheduleHel-
62 per.generatePermutationSchedule(objects);
63 //         schedule = scheduleHel-
64 per.readScheduleFromFile(scheduleFilename);
65 //         scheduleHelper.writeScheduleToFile(schedule, schedule-
66 Filename);
67
68         // constructors
69 //         QuadraticAssignmentProblem qap(objects, flows, distan-
70 ces, costs);
71         QuadraticAssignmentProblem qap(objects, minFlow,
72 maxFlow, minDistance, maxDistance, minCost, maxCost, integerValues);
73 //         QuadraticAssignmentProblem qap(datasetFilename);
74
75         // set data
76 //         qap.setData(objects, flows, distances, costs);
77
78         // generate data
79 //         qap.generateData(objects, minFlow, maxFlow, minDistan-
80 ce, maxDistance, minCost, maxCost, integerValues);
81
82         // read data from file
83 //         qap.readDataFromFile(datasetFilename);
84
85         // write data to file
86 //         qap.writeDataToFile(datasetFilename);
87
88         // show data
89         qap.showData();
90
91         // evaluate quadratic assignment problem (QAP)
92         std::cout << "QAP cost function value: " <<
93 qap.evaluateQAP(schedule) << std::endl;
94
95         // evaluate quadratic assignment problem (QAP) to file
96 //         std::cout << "QAP cost function value: " <<
97 qap.evaluateQAP(schedule, resultFilename) << std::endl;
98
99         // evaluate quadratic bottleneck assignment problem
100 (QBAP)
101         std::cout << "QBAP cost function value: " <<
102 qap.evaluateQBAP(schedule) << std::endl;
103
104         // evaluate quadratic bottleneck assignment problem
105 (QBAP) to file
106 //         std::cout << "QBAP cost function value: " <<
107 qap.evaluateQBAP(schedule, resultFilename) << std::endl;
108
109     } catch (std::exception& e) {
110         std::cout << e.what();
111     }
112 }

```

## 2.5 Problém rozvrhování proudové výroby

Pro využití problému rozvrhování proudové výroby je třeba přidat do projektu hlavičkový soubor *FlowShopSchedulingProblem.h* a zdrojový soubor *FlowShopSchedulingProblem.cpp*. Podrobné informace o třídě *FlowShopSchedulingProblem* jsou dostupné v programové dokumentaci knihovny.

V následujícím popisu budou uvažovány proměnné

<code>datasetFilename</code>	název souboru pro import/export dat problému
<code>scheduleFilename</code>	název souboru pro import/export plánu problému
<code>resultFilename</code>	název souboru pro export výstupu problému
<code>jobs</code>	počet úloh
<code>machines</code>	počet strojů
<code>minProcessingTime</code>	minimální doba zpracování úlohy na stroji
<code>maxProcessingTime</code>	maximální doba zpracování úlohy na stroji
<code>integerValues</code>	typ generovaných dat
<code>processingTimes</code>	doby zpracování úloh na strojích
<code>schedule</code>	plán problému

Vytvoření instance třídy *FlowShopSchedulingProblem* je možné pomocí konstruktoru, který je dostupný ve 3 variantách.

```
// constructors
FlowShopSchedulingProblem fssp(jobs, machines, processingTimes);
FlowShopSchedulingProblem fssp(jobs, machines, minProcessingTime,
maxProcessingTime, integerValues);
FlowShopSchedulingProblem fssp(datasetFilename);
```

První varianta konstrukturu zajišťuje volání metody *FlowShopSchedulingProblem::setData*, druhá varianta konstrukturu zajišťuje volání metody *FlowShopSchedulingProblem::generateData*, třetí varianta konstrukturu zajišťuje volání metody *FlowShopSchedulingProblem::readDataFromFile*.

Pro nastavení dat problému je dostupná metoda *FlowShopSchedulingProblem::setData*.

```
// set data
fssp.setData(jobs, machines, processingTimes);
```

Pro generování dat problému je dostupná metoda *FlowShopSchedulingProblem::generateData*.

```
// generate data
fssp.generateData(jobs, machines, minProcessingTime, maxProcessingTime, integerValues);
```

Pro import/export dat problému ze/do souboru jsou dostupné metody *FlowShopSchedulingProblem::readDataFromFile* a *FlowShopSchedulingProblem::writeDataToFile*.

```
// read data from file
fssp.readDataFromFile(datasetFilename);

// write data to file
fssp.writeDataToFile(datasetFilename);
```

Pro zobrazení dat problému do konzole je dostupná metoda *FlowShopSchedulingProblem::showData*.

```
// show data
fssp.showData();
```

Pro stanovení hodnoty účelové funkce permutačního problému rozvrhování proudové výroby s neomezeným skladem slouží metoda *FlowShopSchedulingProblem::evaluatePFSSP*, která je dostupná ve 2 variantách.

```
// evaluate permutative flow shop scheduling problem (PFSSP)
std::cout << "PFSSP cost function value: " <<
fssp.evaluatePFSSP(schedule) << std::endl;

// evaluate permutative flow shop scheduling problem (PFSSP) to
file
std::cout << "PFSSP cost function value: " <<
fssp.evaluatePFSSP(schedule, resultFilename) << std::endl;
```

Pro stanovení hodnoty účelové funkce problému rozvrhování proudové výroby s omezeným skladem slouží metoda *FlowShopSchedulingProblem::evaluateFSSPB*, která je dostupná ve 2 variantách.

```
// evaluate flow shop scheduling problem with blocking (FSSPB)
std::cout << "FSSPB cost function value: " <<
fssp.evaluateFSSPB(schedule) << std::endl;

// evaluate flow shop scheduling problem with blocking (FSSPB) to
file
std::cout << "FSSPB cost function value: " <<
fssp.evaluateFSSPB(schedule, resultFilename) << std::endl;
```

Pro stanovení hodnoty účelové funkce problému rozvrhování proudové výroby s nulovým zpožděním slouží metoda *FlowShopSchedulingProblem::evaluateFSSPNW*, která je dostupná ve 2 variantách.

```
// evaluate flow shop scheduling problem with no-wait (FSSPNW)
std::cout << "FSSPNW cost function value: " <<
fssp.evaluateFSSPNW(schedule) << std::endl;

// evaluate flow shop scheduling problem with no-wait (FSSPNW) to
file
std::cout << "FSSPNW cost function value: " <<
fssp.evaluateFSSPNW(schedule, resultFilename) << std::endl;
```

První varianty metody slouží pro stanovení hodnoty účelové funkce pro daný plán, druhé varianty jsou obdobou první varianty, které navíc zajistí uložení výstupu (plán + odpovídající hodnota účelové funkce) do souboru.

### 2.5.1 Ukázka použití

```
1 // flow shop scheduling problem header
2 #include "FlowShopSchedulingProblem.h"
3
4 // schedule helper header
5 #include "ScheduleHelper.h"
6
7 // whole operational research library is in namespace operatio-
8 nal_research
9 using namespace operational_research;
10
11 int main(int argc, char **argv)
12 {
13     // set precision
14     std::cout.precision(std::numeric_limits< double >::digits10 +
15 2);
16
17     /***** FLOW SHOP SCHEDULING PROBLEM *****/
18     // dataset filename
19     std::string datasetFilename = "dataset.txt";
20
21     // schedule filename
22     std::string scheduleFilename = "schedule.txt";
23
24     // result filename
25     std::string resultFilename = "result.txt";
26
27     // number of jobs
28     unsigned int jobs = 3;
29
30     // number of machines
31     unsigned int machines = 4;
32
33     // minimal and maximal processing time
```

```

35         double minProcessingTime = 1, maxProcessingTime = 10;
36
37         // generate integer values?
38         bool integerValues = true;
39
40         // processing times
41         std::vector<std::vector<double> > processingTimes =
42 {{1,3,4},{3,5,2},{4,2,1},{2,2,3}};
43
44         // schedule
45         std::vector<unsigned int> schedule = {1,2,3};
46
47         try {
48
49             ScheduleHelper scheduleHelper;
50             schedule = scheduleHel-
51 per.generatePermutationSchedule(jobs);
52 //             schedule = scheduleHel-
53 per.readScheduleFromFile(scheduleFilename);
54 //             scheduleHelper.writeScheduleToFile(schedule, schedule-
55 Filename);
56
57             // constructors
58 //             FlowShopSchedulingProblem fssp(jobs, machines, proces-
59 singTimes);
60             FlowShopSchedulingProblem fssp(jobs, machines, minPro-
61 cessingTime, maxProcessingTime, integerValues);
62 //             FlowShopSchedulingProblem fssp(datasetFilename);
63
64             // set data
65 //             fssp.setData(jobs, machines, processingTimes);
66
67             // generate data
68 //             fssp.generateData(jobs, machines, minProcessingTime,
69 maxProcessingTime, integerValues);
70
71             // read data from file
72 //             fssp.readDataFromFile(datasetFilename);
73
74             // write data to file
75 //             fssp.writeDataToFile(datasetFilename);
76
77             // show data
78             fssp.showData();
79
80             // evaluate permutative flow shop scheduling problem
81 (PFSSP)
82             std::cout << "PFSSP cost function value: " <<
83 fssp.evaluatePFSSP(schedule) << std::endl;
84
85             // evaluate permutative flow shop scheduling problem
86 (PFSSP) to file
87 //             std::cout << "PFSSP cost function value: " <<
88 fssp.evaluatePFSSP(schedule, resultFilename) << std::endl;
89
90             // evaluate flow shop scheduling problem with blocking
91 (FSSPB)

```

```

        std::cout << "FSSPB cost function value: " <<
fssp.evaluateFSSPB(schedule) << std::endl;

        // evaluate flow shop scheduling problem with blocking
(FSSPB) to file
        //      std::cout << "FSSPB cost function value: " <<
fssp.evaluateFSSPB(schedule, resultFilename) << std::endl;

        // evaluate flow shop scheduling problem with no-wait
(FSSPNW)
        std::cout << "FSSPNW cost function value: " <<
fssp.evaluateFSSPNW(schedule) << std::endl;

        // evaluate flow shop scheduling problem with no-wait
(FSSPNW) to file
        //      std::cout << "FSSPNW cost function value: " <<
fssp.evaluateFSSPNW(schedule, resultFilename) << std::endl;

        } catch (std::exception& e) {
            std::cout << e.what();
        }
    }
}

```

## 2.6 Problém obchodního cestujícího

Pro využití problému obchodního cestujícího je třeba přidat do projektu hlavičkový soubor *TravellingSalesmanProblem.h* a zdrojový soubor *TravellingSalesmanProblem.cpp*. Podrobné informace o třídě *TravellingSalesmanProblem* jsou dostupné v programové dokumentaci knihovny.

V následujícím popisu budou uvažovány proměnné

<code>datasetFilename</code>	název souboru pro import/export dat problému
<code>scheduleFilename</code>	název souboru pro import/export plánu problému
<code>resultFilename</code>	název souboru pro export výstupu problému
<code>cities</code>	počet měst
<code>minCoordinate</code>	minimální souřadnice města
<code>maxCoordinate</code>	maximální souřadnice města
<code>integerValues</code>	typ generovaných dat
<code>coordinates</code>	souřadnice měst
<code>schedule</code>	plán problému



Vytvoření instance třídy *TravellingSalesmanProblem* je možné pomocí konstruktoru, který je dostupný ve 3 variantách.

```
// constructors
TravellingSalesmanProblem tsp(cities, coordinates);
TravellingSalesmanProblem tsp(cities, minCoordinate, maxCoordinate, integerValues);
TravellingSalesmanProblem tsp(datasetFilename);
```

První varianta konstruktoru zajišťuje volání metody *TravellingSalesmanProblem::setData*, druhá varianta konstruktoru zajišťuje volání metody *TravellingSalesmanProblem::generateData*, třetí varianta konstruktoru zajišťuje volání metody *TravellingSalesmanProblem::readDataFromFile*.

Pro nastavení dat problému je dostupná metoda *TravellingSalesmanProblem::setData*.

```
// set data
tsp.setData(cities, coordinates);
```

Pro generování dat problému je dostupná metoda *TravellingSalesmanProblem::generateData*.

```
// generate data
tsp.generateData(cities, minCoordinate, maxCoordinate, integerValues);
```

Pro import/export dat problému ze/do souboru jsou dostupné metody *TravellingSalesmanProblem::readDataFromFile* a *TravellingSalesmanProblem::writeDataToFile*.

```
// read data from file
tsp.readDataFromFile(datasetFilename);

// write data to file
tsp.writeDataToFile(datasetFilename);
```

Pro zobrazení dat problému do konzole je dostupná metoda *TravellingSalesmanProblem::showData*.

```
// show data
tsp.showData();
```

Pro stanovení hodnoty účelové funkce symetrického problému obchodního cestujícího slouží metoda *TravellingSalesmanProblem::evaluateSTSP*, která je dostupná ve 2 variantách.

```
// evaluate symmetric travelling salesman problem (STSP)
std::cout << "STSP cost function value: " <<
tsp.evaluateSTSP(schedule) << std::endl;
```

```

// evaluate symmetric travelling salesman problem (STSP) to file
std::cout << "STSP cost function value: " <<
tsp.evaluateSTSP(schedule, resultFilename) << std::endl;

```

První varianta metody slouží pro stanovení hodnoty účelové funkce pro daný plán, druhá varianta je obdobou první varianty, která navíc zajistí uložení výstupu (plán + odpovídající hodnota účelové funkce) do souboru.

### 2.6.1 Ukázka použití

```

1 // travelling salesman problem header
2 #include "TravellingSalesmanProblem.h"
3
4 // schedule helper header
5 #include "ScheduleHelper.h"
6
7 // whole operational research library is in namespace operatio-
8 nal_research
9 using namespace operational_research;
10
11 int main(int argc, char **argv)
12 {
13     // set precision
14     std::cout.precision(std::numeric_limits< double >::digits10 +
15 2);
16
17     /***** TRAVELLING SALESMAN PROBLEM *****/
18     *****/
19     // dataset filename
20     std::string datasetFilename = "dataset.txt";
21
22     // schedule filename
23     std::string scheduleFilename = "schedule.txt";
24
25     // result filename
26     std::string resultFilename = "result.txt";
27
28     // number of cities
29     unsigned int cities = 3;
30
31     // minimal and maximal coordinates of cities
32     double minCoordinate = -10, maxCoordinate = 10;
33
34     // generate integer values?
35     bool integerValues = true;
36
37     // coordinates
38     std::vector<CoordinatesHelper> coordinates = {Coordinate-
39 sHelper(-5,2), CoordinatesHelper(1,3), CoordinatesHelper(6,9)};
40
41     // schedule
42     std::vector<unsigned int> schedule = {1,2,3};
43
44     try {

```

```

45
46         // schedule generate/read/write
47         ScheduleHelper scheduleHelper;
48         schedule = scheduleHel-
49 per.generatePermutationSchedule(cities);
50 //         schedule = scheduleHel-
51 per.readScheduleFromFile(scheduleFilename);
52 //         scheduleHelper.writeScheduleToFile(schedule, schedule-
53 Filename);
54
55         // constructors
56 //         TravellingSalesmanProblem tsp(cities, coordinates);
57         TravellingSalesmanProblem tsp(cities, minCoordinate,
58 maxCoordinate, integerValues);
59 //         TravellingSalesmanProblem tsp(datasetFilename);
60
61         // set data
62 //         tsp.setData(cities, coordinates);
63
64         // generate data
65 //         tsp.generateData(cities, minCoordinate, maxCoordinate,
66 integerValues);
67
68         // read data from file
69 //         tsp.readDataFromFile(datasetFilename);
70
71         // write data to file
72 //         tsp.writeDataToFile(datasetFilename);
73
74         // show data
75         tsp.showData();
76
77         // evaluate symmetric travelling salesman problem
78         (STSP)
79
80         std::cout << "STSP cost function value: " <<
81 tsp.evaluateSTSP(schedule) << std::endl;
82
83         // evaluate symmetric travelling salesman problem
84         (STSP) to file
85 //         std::cout << "STSP cost function value: " <<
86 tsp.evaluateSTSP(schedule, resultFilename) << std::endl;
87
88         } catch (std::exception& e) {
89             std::cout << e.what();
90         }
91     }

```

## 2.7 Kapacitní rozvozní problém

Pro využití kapacitního rozvozního problému je třeba přidat do projektu hlavičkový soubor *CapacitatedVehicleRoutingProblem.h* a zdrojový soubor *CapacitatedVehicleRoutingProblem.cpp*. Podrobné informace o třídě *CapacitatedVehicleRoutingProblem* jsou dostupné v programové dokumentaci knihovny.

V následujícím popisu budou uvažovány proměnné

<code>datasetFilename</code>	název souboru pro import/export dat problému
<code>scheduleFilename</code>	název souboru pro import/export plánu problému
<code>resultFilename</code>	název souboru pro export výstupu problému
<code>customers</code>	počet zákazníků
<code>vehicleCapacity</code>	kapacita vozidla
<code>minCoordinate</code>	minimální souřadnice zákazníka
<code>maxCoordinate</code>	maximální souřadnice zákazníka
<code>minDemand</code>	minimální požadavek zákazníka
<code>maxDemand</code>	maximální požadavek zákazníka
<code>integerValues</code>	typ generovaných dat
<code>coordinates</code>	souřadnice zákazníků
<code>demands</code>	požadavky zákazníků
<code>schedule</code>	plán problému

Vytvoření instance třídy *CapacitatedVehicleRoutingProblem* je možné pomocí konstruktoru, který je dostupný ve 3 variantách.

```
// constructors
CapacitatedVehicleRoutingProblem cvrp(customers, demands, coordinates, capacity);
CapacitatedVehicleRoutingProblem cvrp(customers, capacity, minDemand, maxDemand, minCoordinate, maxCoordinate, integerValues);
CapacitatedVehicleRoutingProblem cvrp(datasetFilename);
```

První varianta konstruktoru zajišťuje volání metody *CapacitatedVehicleRoutingProblem::setData*, druhá varianta konstruktoru zajišťuje volání metody *CapacitatedVehicleRoutingProblem::generateData*, třetí varianta konstruktoru zajišťuje volání metody *CapacitatedVehicleRoutingProblem::readDataFromFile*.

Pro nastavení dat problému je dostupná metoda *CapacitatedVehicleRoutingProblem::setData*.

```
// set data
cvrp.setData(customers, demands, coordinates, capacity);
```

Pro generování dat problému je dostupná metoda *CapacitatedVehicleRoutingProblem::generateData*.

```
// generate data
cvrp.generateData(customers, capacity, minDemand, maxDemand, min-
Coordinate, maxCoordinate, integerValues);
```

Pro import/export dat problému ze/do souboru jsou dostupné metody *CapacitatedVehicleRoutingProblem::readDataFromFile* a *CapacitatedVehicleRoutingProblem::writeDataToFile*.

```
// read data from file
cvrp.readDataFromFile(datasetFilename);

// write data to file
cvrp.writeDataToFile(datasetFilename);
```

Pro zobrazení dat problému do konzole je dostupná metoda *CapacitatedVehicleRoutingProblem::showData*.

```
// show data
cvrp.showData();
```

Pro stanovení hodnoty účelové funkce kapacitního rozvozního problému slouží metoda *CapacitatedVehicleRoutingProblem::evaluateCVRP*, která je dostupná ve 2 variantách.

```
// evaluate capacitated vehicle routing problem problem (CVRP)
std::cout << "CVRP cost function value: " <<
cvrp.evaluateCVRP(schedule) << std::endl;

// evaluate capacitated vehicle routing problem problem (CVRP) to
file
std::cout << "CVRP cost function value: " <<
cvrp.evaluateCVRP(schedule, resultFilename) << std::endl;
```

První varianta metody slouží pro stanovení hodnoty účelové funkce pro daný plán, druhá varianta je obdobou první varianty, která navíc zajistí uložení výstupu (plán + odpovídající hodnota účelové funkce) do souboru.

### 2.7.1 Ukázka použití

```
1 // capacitated vehicle routing problem header
2 #include "CapacitatedVehicleRoutingProblem.h"
3
4 // schedule helper header
5 #include "ScheduleHelper.h"
6
7 // whole operational research library is in namespace operatio-
8 nal_research
```

```

9 using namespace operational_research;
10
11 int main(int argc, char **argv)
12 {
13     // set precision
14     std::cout.precision(std::numeric_limits< double >::digits10 +
15 2);
16
17     /***** CAPACITATED VEHICLE ROUTING PROBLEM *****/
18     // dataset filename
19     std::string datasetFilename =
20 "C:\\Users\\Milan\\Desktop\\test\\dataset.txt";
21
22     // schedule filename
23     std::string scheduleFilename =
24 "C:\\Users\\Milan\\Desktop\\test\\schedule.txt";
25
26     // result filename
27     std::string resultFilename =
28 "C:\\Users\\Milan\\Desktop\\test\\result.txt";
29
30     // number of cities
31     unsigned int customers = 3;
32
33     // vehicle capacity
34     double capacity = 10;
35
36     // minimal and maximal coordinates of customers
37     double minCoordinate = -10, maxCoordinate = 10;
38
39     // minimal and maximal demands of customers
40     double minDemand = 1, maxDemand = 10;
41
42     // generate integer values?
43     bool integerValues = true;
44
45     // coordinates
46     std::vector<CoordinatesHelper> coordinates = {Coordinate-
47 sHelper(-5,2), CoordinatesHelper(1,3), CoordinatesHelper(6,9)};
48
49     // demands
50     std::vector<double> demands = {3,5,4};
51
52     // schedule
53     std::vector<unsigned int> schedule = {1,2,3};
54
55     try {
56
57         // schedule generate/read/write
58         ScheduleHelper scheduleHelper;
59         schedule = scheduleHel-
60 per.generatePermutationSchedule(customers);
61 // schedule = scheduleHel-
62 per.readScheduleFromFile(scheduleFilename);
63 // scheduleHelper.writeScheduleToFile(schedule, schedule-
64 Filename);
65

```

```

66
67         // constructors
68 //         CapacitatedVehicleRoutingProblem cvrp(customers, de-
69 mands, coordinates, capacity);
70         CapacitatedVehicleRoutingProblem cvrp(customers, capa-
71 city, minDemand, maxDemand, minCoordinate, maxCoordinate, integerVa-
72 lues);
73 //         CapacitatedVehicleRoutingProblem
74 cvrp(datasetFilename);
75
76         // set data
77 //         cvrp.setData(customers, demands, coordinates, capaci-
78 ty);
79
80         // generate data
81 //         cvrp.generateData(customers, capacity, minDemand, max-
82 Demand, minCoordinate, maxCoordinate, integerValues);
83
84         // read data from file
85 //         cvrp.readDataFromFile(datasetFilename);
86
87         // write data to file
88 //         cvrp.writeDataToFile(datasetFilename);
89
90         // show data
91         cvrp.showData();
92
93         // evaluate capacitated vehicle routing problem pro-
94 blem (CVRP)
95         std::cout << "CVRP cost function value: " <<
96 cvrp.evaluateCVRP(schedule) << std::endl;
97
98         // evaluate capacitated vehicle routing problem pro-
99 blem (CVRP) to file
100 //         std::cout << "CVRP cost function value: " <<
101 cvrp.evaluateCVRP(schedule, resultFilename) << std::endl;
102
103     } catch (std::exception& e) {
104         std::cout << e.what();
105     }
106 }

```