Silesian
University
of Technology

SILESIAN UNIVERSITY OF TECHNOLOGY

FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

PROGRAMME: INFORMATICS

Final Project

Design and implementation of web application used for solving vehicle routing problems with time windows.

author: Łukasz Kwiecień

supervisor: Tomasz Jastrząb, PhD

Gliwice, December 2021

Contents

Abstract

Abstract - the abstract text should be copied into the respective field in the APD system. Abstract with keywords should not exceed one page.

Keywords: 2-5 keywords, separated by commas

Chapter 1

Introduction

Transportation is one of the most critical activities in the supply chain. Its importance comes from the fact that the transportation costs can reach up to almost 40% of the total logistics costs of a manufacturing company. [13] For that reason, the companies need to transport the goods or persons efficiently. That goal can be achieved by either minimizing the distance traveled by the vehicles or reducing the number of routes required to visit all destinations. The Vehicle Routing Problem (VRP) describes the problem of assigning loads to the vehicles and sequencing the customers assigned to each vehicle to obtain optimal routes.

VRP is one of the most studied combinatorial optimization problems. Its popularity is due to the fact that VRP can be used in many real-life scenarios in the fields of distribution, collection, and logistics. In a VRP the fleet of vehicles has to visit a set of customers starting from a given depot and deliver commodities to them, taking into consideration all given constraints. There are various constraints that could be introduced to the problem, for example,the time windows for the customers or the capacity of vehicles.

Depending on the problem variant, VRP is a combination of two or more NP-hard problems, which makes VRP also NP-hard problem.

The fact that the problem is NP-hard makes the process of finding an exact solution very time-consuming. Therefore it is necessary to use a heuristic approach to get good solutions in an acceptable time.

In this thesis a web application solving Capacitated Vehicle Routing Problem with Time Windows (CVRTPW) is considered. User defines the problem by picking waypoints on the map and determining the capacity and time window constraints. Application, if feasible, solves the problem and presents the results in a user-friendly way by visualizing all routes on the map. The problem is solved using Push Forward Insertion Heuristic and optimized using Local Search with λ interchange method. The fleet of vehicles is homogeneus, which means that every vehicle has the same amount of goods that can be transported in it. Every customer has its own time window within which the delivery must be made. The goal was to create a tool that will simplify the process of defining and solving the CVRTPW for the end user.

1.1 Content outline

The second chapter "Problem analysis" provides history and explores scientific background material for the Capacitated Vehicle Routing Problem with Time Windows. Moreover provides formal and detailed definition of the problem and the solution methods based on the researched literature. Chapter 3 focuses on the design and implementation process of the project. Shows the functional and nonfuctional requirements as well as diagrams describing the system. The tools and metodologies used for the design and implementation process are also described there. The fourth chapter covers the hardware and software requirements for the application. Provides the step by step installation procedure along with the user manual. The usage examples and screenshots of the working application can also be found there. Chapter 5 discusses the software part of the project in detail. Description of the architecture, code and structure of the project is explained in this

chapter. Furthermore specifies in detail the software-side of the entire process of solving the CVRPTW. Chapter 6 shows how the project was tested and verified if the requirements set during the design process were fulfilled. The last chapter 'Conclusions' summarizes the achieved results and describes the encountered difficulties.

Chapter 2

Problem analysis

2.1 Vehicle Routing Problem

The problem concerns a set of customers to whom products need to be delivered in such a way that the delivery cost is as low as possible. Transport is performed by a fleet of vehicles, each of which can carry a certain number of goods. The goal is to serve all customers with as few vehicles as possible and to keep the total distance travelled by all vehicles as short as possible within predefined constraints. The classic version of VRP has the following constraints:

1. Each vehicle's route starts and ends at a depot.
2. All goods must be delivered.
3. Customer must receive all goods at one time delivered by one vehicle.
4. The vehicle may not carry more goods than its capacity allows.

The VRP can be formally defined as directed graph $G = (V,A)$, where $V = \{0,\ldots,n\}$ is the set of vertices representing customers and the depot, and A is the set of arcs (i,j) connecting these vertices. The set of vehicles is denoted by W. The depot is marked as vertex 0. The

number of vehicles is denoted by *m*, and each vehicle has a capacity of *Q*. The cost of travel between vertices *i* and *j* is denoted by c_{ij}. The cost is the distance, duration or other costs that may occur during the travel between nodes. The demand of customer *i* is represented as q_i. The customer subset *S* $\subseteq$ *V*\ {*0*} can be served by a minimum of *r(S)* vehicles. This number can be obtained by solving the Bin Packing Problem (BPP) with bin set *S* with capacities *Q*, but because BPP is an NP-hard problem, it can be approximated by its lower bound: $\lceil \sum_{i \in S} q_i/Q \rceil$.[4]

The Integer Linear Programming formulation of the problem:[16]

$$Minimize \sum_{i,j \in V} c_{ij} x_{ij} \tag{2.1}$$

subject to:

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \setminus \{0\} \tag{2.2}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \setminus \{0\} \tag{2.3}$$

$$\sum_{j \in V} x_{0j} = m \tag{2.4}$$

$$\sum_{i \in V} x_{i0} = m \tag{2.5}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq \lceil \sum_{i \in S} q_i/Q \rceil \qquad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset \tag{2.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V \tag{2.7}$$

In this formula, x_{ij} is a binary variable whose value indicates whether the arc between vertices i and j is traversed in the solution. If the connection between these vertices belongs to the solution, then the variable

x_{ij} is equal to 1, otherwise it is equal to 0. ***Indegree*** (2.2) and ***outdegree*** (2.3) constraints guarantee that every customer is visited only once. Constraints 2.4 and 2.5 ensure that the solution has m routes, one for each vehicle (the number of routes does not exceed the number of vehicles). Constraints 2.6 are called ***capacity-cut constraints*** and refer to the minimum number of vehicles needed to serve a set of customers whose total demand is $\sum_{\mathrm{i}\in\mathrm{S}} q_i$. These constraints ensure the connectivity of the solution and that there are no paths whose load would exceed the capacity of the vehicle. The last constraint 2.7 ensures the binarity of the variable x_{ij}. If x_{0j} is equal to 1 it means that j belongs to this route. Any route from the solution can be reconstructed in this way: the next customer in the route is customer i, for whom x_{ij} is equal to 1. The last customer is customer i for whom x_{i0} is equal to 1. [16]

The problem of vehicle routing was first introduced by Dantzig and Ramser in 1959 as "The truck dispatching problem". [5] In the following years it became very popular, resulting in many studies, books and scientific publications exploring the topic. The problem is one of the most studied combinatorial optimisation problems, since it offers many benefits for logistics and transport companies. Different implementations can save as much as 5% to 30% of transport-related costs. [7]

2.2 Vehicle Routing Problem with Time Windows

The Vehicle Routing Problem with Time Windows (VRPTW) is a variant of VRP in which a constraint due to time windows is added. In this variant of the problem, each customer i has a time window $[a_i,b_i]$ in which it must be served. A customer cannot be served earlier than a_i and later than b_i. In case the vehicle arrives earlier than a_i it must wait until the window opens. Moreover, to each arc(i,j) the time t_{ij} is assigned. For each vertex i and vehicle k a decision variable s_{ik} is defined.

This variable represents the time at which the vehicle *k* starts to serve the customer *i*. [1] This variant of VRP is also an NP-hard problem. [11]

The VRPTW can be stated mathematically as multicommodity network flow formulation with time windows and capacity constraints:[1]

$$min \sum_{k\in W} \sum_{i,j\in V} c_{ij}x_{ijk}s.t. \tag{2.8}$$

subject to:

$$\sum_{k\in W} \sum_{i\in V} x_{ijk} = 1 \qquad \forall i \in V \setminus \{0\} \tag{2.9}$$

$$\sum_{i\in V\setminus\{0\}} q_i \sum_{j\in V} x_{ijk} \leq Q \qquad \forall k \in W \tag{2.10}$$

$$\sum_{j\in V} x_{0jk} = 1 \qquad \forall k \in W \tag{2.11}$$

$$\sum_{i\in V} x_{ihk} - \sum_{j\in V} x_{hjk} = 0 \qquad \forall h \in V \setminus \{0\}, \forall k \in W \tag{2.12}$$

$$\sum_{i\in V} x_{i0k} = 1 \qquad \forall k \in W \tag{2.13}$$

$$x_{ijk}(s_{ik} + t_{ij} - s_{jk}) \leq 0 \qquad \forall i, j \in V, \forall k \in W \tag{2.14}$$

$$a_i \leq s_{ik} \leq b_i \qquad \forall i \in V, \forall k \in W \tag{2.15}$$

$$x_{ijk} \in 0, 1 \qquad \forall i, j \in V, \forall k \in W \tag{2.16}$$

Objective function (2.8) minimizes travel cost. Constraints (2.9) and (2.10) ensures that each customer is visited once and vehicle cannot surpass its capacity. Equations (2.11), (2.12) and (2.13) indicate that

each vehicle must leave the depot, once it has reached the customer it must travel to the next location and finally return to the depot. Inequality (2.14) shows the relationship between vehicle departure time and its successor. Inequalities (2.15) provide that the time windows are observed. Constraints (2.16) are the integrality constraints. [1]

2.3 Solution approaches

There are two approaches to solving VRPs. The first is to solve using exact methods, which are guaranteed to find the most optimal solution. However, these methods are very expensive in terms of memory and computation time, which makes them applicable only to problems of low complexity. The second approach is to use approximate methods (heuristics), which allow solving larger and more complex problems in a reasonable time. However, heuristics do not guarantee the most optimal solution.

2.3.1 Exact methods

The exact methods were divided by Laporte and Norbert's into three families: Direct Tree Search methods, Dynamic Programming and Integer Linear Programming. [9]

The Direct Tree Search family includes branch and bound algorithms. The solution using this algorithm was first proposed by Christofides and Eilon in 1969 [2]. However, the algorithm was only able to solve problems up to 13 clients, due to time complexity. Over the years, the methods in this family have been improved, allowing solutions for problems with up to 25 [3] clients and even, in later years, 250 [8].

The Dynamic Programming solutions were, at first, able to solve problems up to 25 customers. Later, significant improvements were made that allowed to solve problems up to 50 customers.

The Integer Linear Programming methods produced some of the best exact solutions. This has been achieved by extending the set partitioning algorithm using the column generation method [6].

2.3.2 Heuristics

Heuristics do not guarantee the best solution, their aim is to obtain a solution close to the best, in a relatively short time. They make it possible to solve larger and more complex problems that would not be possible to solve with exact methods. For VRP, we can distinguish between two types of herustics: construction and improvement. The process of solving a VRP usually can be divided into two steps: an initial solution is created using the construction heuristic, and then the improvement heuristics improves the solution.

Construction heuristics

One of the best known construction heuristics is Clarke and Wright's savings algorithm. In the first step, separate routes are created for each customer, even if the number of vehicles is insufficient. Then, the savings for all connections between edges (*i,j*) are calculated using the formula $s_{ij} = c_{i0} + C_{0j}$ - c_{ij}. After that, the savings are sorted in the descending order and algorithm tries to merge two routes that will reduce the total costs of travel. Routes are merged by connecting the customer from one route to the customer from the second route. A merge between *i* and *j* is only possible if they are the first or last vertices of their routes and the vehicle capacity constraint is not violated. The algorithm stops when there are no more possible connections.

Another type of Construction herustics is insertion heuristics. The Push forward insertion heuristics used in this thesis will be discussed in detail later in the chapter.

The last type of design heuristics are two-phase heuristics. They divide the problem into two sub-problems and then solve each of them. One such method is cluster first, route second. In this approach, customers are clustered into a base set of routes. Each cluster is treated as a separate TSP instance. There are various methods of clustering customers, one of which is the sweep algorithm. The sweep algorithm builds routes by sweeping a ray, centered at the depot, clockwise adding each customer lying on the line of the circle to the cluster.

Improvement heuristics

Once an initial solution is found using the construction heuristics, it is improved using the improvement heuristics. One type of such heuristics is the local search heuristic, in which the best solution is selected from the neighbourhood until a stopping criterion is reached. Neighbourhood solutions are created from the current one by applying operators that modify the original solution.

Local search with λ interchange heuristic will be discussed later in the chapter. This improvement heuristic method was used in the project.

Simulated annealing is another example. This method replicates the process of heating up a material and then slowly lowering the temperature to reduce defects. The main assumption of the algorithm is to infrequently accept worse solutions in order to avoid a local optimum. The solution is accepted with a probability calculated by means of a function depending on the difference in quality between the new and the current solution, and a parameter called the current temperature.

Another method is Tabu search. Based on the initial solution, neighbouring solutions are determined and the best of these solutions is selected as the new best solution. The previous solutions are stored in a tabu list, it has a predefined length, and when it is full, adding another solution removes the oldest one from the list. Such a mechanism avoids a local optimum and a return to previous solutions.

VRP can be solved using many other heuristics, including: Large Neighbourhood Searches, Genetic Algorithms or Ant Colonies.

2.3.3 Push forward insertion heuristic

In this thesis, the initial solution is created using Push Forward Insertion Heuristic. It was introduced by Solomon [12] in 1987 and it is an efficient method to insert customers into routes. The algorithm creates a new route by selecting the starting customer furthest from the depot and then inserts each unassigned customer into the route until the capacity or time constraints are met. Then new route is created and the process repeats until all customers are routed. The feasibility check ensures that all constraints are not violated. This method generates a good starting solution for later improvements, in a short time.

For the pseudocode for PFIH algorithm for VPRTW see figure 2.1 [14].

2.3.4 Local search with λ interchange

The λ interchange neighbourhood generation method was introduced by Osman and Christofides [10]. Local Search is performed by interchanging clients between routes. The interchanges are performed through operators whose number depends on the λ parameter. In this work, this parameter is set to 2, which means that the maximum number of interchanged clients between routes will be 2. For this parameter value, we can distinguish 8 interchange operators: (0,1), (1,0), (1,1), (0,2), (2,0), (2,1), (1,2), (2,2). Operator (1,1) means that one customer will be transferred from route R_i to R_j and also one customer will be transferred from route R_j to R_i. Operator 2,1 means that two customers will be moved from route R_i to R_j and only one customer will be moved

```
1  Begin an empty route $r_0$ starting from the depot; $i := 0$;
2  Among all unassigned customers, select the customer
     farthest from the depot and insert into the current
     route $r_i$;
3  if all customers are routed then goto line 13.
4  else
5      if the capacity $Q$ of the vehicle $k$ involved in the
           current route $r_i$ is exceeded then goto line 12.
6     else
7        foreach unassigned customer
8           find the best position for insertion in $r_i$
               without violating any specified time window
               to compare the cost of starting a new route
               against that for the best position found.
9  if there exists no feasible position for insertion into
     the current route $r_i$ then goto line 12.
10 else
11    Pick the customer with the greatest cost difference
        and insert it into $r_i$ and update the capacity $Q$ of
        the vehicle $k$ involved. goto line 3.
12 Start a new route $r_{i+1}$ starting from the depot; $i := i + 1$;
     goto line 2.
13 Return the current solution
```

Figure 2.1: Push Forward Insertion Heuristic for Vehicle Routing Problem with Time Windows

from route R_j to R_i. The other operators work on the same principle. If the solution has improved after the interchange of customers between routes, it is then accepted, otherwise such a solution is rejected. [15]

To select between the candidate solutions a global-best strategy was chosen. The global-best strategy searches all solutions in the neighbourhood of the current solution for all operators and selects the one that gives the greatest cost reduction. The local search is stopped after a fixed number of iterations or other specified stopping criterion. [15]

The algorithm can be summarised as follows: After generating a starting solution its entire neighbourhood (generated using the operators) is searched for a candidate solution. The candidate becomes the new current solution and its neighbourhood is searched to find a better solution. In case there are no more solutions in the neighbourhood that improve our current solution the algorithm is stopped.

Chapter 3

Requirements and tools

- **functional and nonfunctional requirements**
- **use cases (UML diagrams)**
- **description of tools**
- **methodology of design and implementation**

Chapter 4

External specification

- **hardware and software requirements**
- **installation procedure**
- **activation procedure**
- **types of users**
- **user manual**
- **system administration**
- **security issues**
- **example of usage**
- **working scenarios (with screenshots or output files)**

Figure 4.1: Figure caption (below the figure).

Chapter 5

Internal specification

- concept of the system
- system architecture
- description of data structures (and data bases)
- components, modules, libraries, resume of important classes (if used)
- resume of important algorithms (if used)
- details of implementation of selected parts
- applied design patterns
- UML diagrams

Use special environment for inline code, eg `descriptor` or `descriptor_gaussian` . Longer parts of code put in the figure environment, eg. code in Fig. 5.1. Very long listings–move to an appendix.

```
class descriptor_gaussian : virtual public descriptor
{
   protected:
      /** core of the gaussian fuzzy set */
      double _mean;
      /** fuzzyfication of the gaussian fuzzy set */
      double _stddev;

   public:
      /** @param mean core of the set
          @param stddev standard deviation */
      descriptor_gaussian (double mean, double stddev);
      descriptor_gaussian (const descriptor_gaussian & w
         );
      virtual ~descriptor_gaussian();
      virtual descriptor * clone () const;

      /** The method elaborates membership to the
         gaussian fuzzy set. */
      virtual double getMembership (double x) const;

};
```

Figure 5.1: The **descriptor_gaussian** class.

Chapter 6

Verification and validation

- **testing paradigm (eg V model)**
- **test cases, testing scope (full / partial)**
- **detected and fixed bugs**
- **results of experiments (optional)**

Chapter 7

Conclusions

- **achieved results with regard to objectives of the thesis and requirements**
- **path of further development (eg functional extension . . .)**
- **encountered difficulties and problems**

Table 7.1: A caption of a table is **above** it.

	method						
			alg. 3			alg. 4, $\gamma = 2$	
ζ	alg. 1	alg. 2	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 3$	$\beta = 0.1$	$\beta = -0.1$
0	8.3250	1.45305	7.5791	14.8517	20.0028	1.16396	1.1365
5	0.6111	2.27126	6.9952	13.8560	18.6064	1.18659	1.1630
10	11.6126	2.69218	6.2520	12.5202	16.8278	1.23180	1.2045
15	0.5665	2.95046	5.7753	11.4588	15.4837	1.25131	1.2614
20	15.8728	3.07225	5.3071	10.3935	13.8738	1.25307	1.2217
25	0.9791	3.19034	5.4575	9.9533	13.0721	1.27104	1.2640
30	2.0228	3.27474	5.7461	9.7164	12.2637	1.33404	1.3209
35	13.4210	3.36086	6.6735	10.0442	12.0270	1.35385	1.3059
40	13.2226	3.36420	7.7248	10.4495	12.0379	1.34919	1.2768
45	12.8445	3.47436	8.5539	10.8552	12.2773	1.42303	1.4362
50	12.9245	3.58228	9.2702	11.2183	12.3990	1.40922	1.3724

Bibliography

[1] Kallehauge B. et al. “Vehicle Routing Problem with Time Windows.” In: *Column Generation.* Boston, MA.: Springer, 2005, pp. 67–98.

[2] N. Christofides and S. Eilon. “An Algorithm for the Vehicle-dispatching Problem”. In: *Journal of the Operational Research Society* 20.3 (1969), pp. 309–318.

[3] N. Christofides, A. Mingozzi and P. Toth. “Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations”. In: *Mathematical Programming* 20.1 (1981), pp. 255–282.

[4] J. F. Cordeau et al. “Vehicle Routing”. In: *Management Science* 14 (2007), pp. 367–428.

[5] G. B. Dantzig and J. H. Ramser. “The Truck Dispatching Problem”. In: *Management Science* 6.1 (1959), pp. 80–91.

[6] M. Desrochers, J. Desrosiers and M. Solomon. “A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows”. In: *Operations Research* 40.2 (1992), pp. 342–354.

[7] Geir Hasle, Knut-Andreas Lie and Ewald Quak. *Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF.* Berlin: Springer, 2007. ISBN: 978-3-540-68783-2.

[8] G. Laporte, H. Mercure and Y. Nobert. “An exact algorithm for the asymmetrical capacitated vehicle routing problem”. In: *Networks* 16.1 (1986), pp. 33–46.

[9] G. Laporte and Y. Nobert. “Exact Algorithms for the Vehicle Routing Problem.” In: *Surveys in Combinatorial Optimization*. Ed. by S. Martello et al. Vol. 132. North-Holland Mathematics Studies, 1987, pp. 147–184.

[10] I. H. Osman and N. Christofides. “Simulated annealing and descent algorithms for capacitated clustering problem”. In: *Imperial College, University of London, Research Report* (1989).

[11] M. W. P. Savelsbergh. “Local search in routing problems with time windows”. In: *Annals of Operations Research* 4 (1985), pp. 285–305.

[12] Marius M. Solomon. “Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints”. In: *Operations Research* 35.2 (1987), pp. 254–265.

[13] Katarzyna Sukiennik. “Logistics Costs in Manufacturing Companies”. In: *Zeszyty Naukowe Politechniki Częstochowskiej. Zarządzanie* 4 (2011), pp. 131–139.

[14] V. Tam and K.T. Ma. “An Effective Search Framework Combining Meta-Heuristics to Solve the Vehicle Routing Problems with Time Windows”. In: *Vehicle Routing Problem*. Ed. by Tonci Caric and Hrvoje Gold. Rijeka: IntechOpen, 2008. Chap. 3.

[15] K. Tan et al. “Heuristic methods for vehicle routing problem with time windows.” In: *AI in Engineering* 15 (Jan. 2001), pp. 281–295.

[16] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Society for Industrial & Applied Mathematics, 2002. ISBN: 978-0-89871-498-2.

Appendices

Index of abbreviations and symbols

VRP	**Vehicle Routing Problem**
VRPTW	**Vehicle Routing Problem with Time Windows**
CVRPTW	**Capacitated Vehicle Routing Problem with Time Windows**
V	**Set of vertices representing customers and the depot**
A	**Arcs connecting vertices from** V
W	**Set of vehicles**
Q	**Capacity of vehicle**

Listings

(Put long listings in the appendix.)

```
partition fcm_possibilistic::doPartition
                              (const dataset & ds)
{
   try
   {
      if (_nClusters < 1)
         throw std::string ("unknown␣number␣of␣clusters");
      if (_nIterations < 1 and _epsilon < 0)
         throw std::string ("You␣should␣set␣a␣maximal␣number␣of␣iteration␣or␣minimal␣difference␣--␣epsilon.");
      if (_nIterations > 0 and _epsilon > 0)
         throw std::string ("Both␣number␣of␣iterations␣and␣minimal␣epsilon␣set␣--␣you␣should␣set␣either␣number␣of␣iterations␣or␣minimal␣epsilon.");

      auto mX = ds.getMatrix();
      std::size_t nAttr = ds.getNumberOfAttributes();
      std::size_t nX     = ds.getNumberOfData();
      std::vector<std::vector<double>> mV;
     mU = std::vector<std::vector<double>> (_nClusters
        );
    for (auto & u : mU)
        u = std::vector<double> (nX);
    randomise(mU);
    normaliseByColumns(mU);
    calculateEtas(_nClusters, nX, ds);
    if (_nIterations > 0)
    {
        for (int iter = 0; iter < _nIterations; iter
            ++)
        {
            mV = calculateClusterCentres(mU, mX);
            mU = modifyPartitionMatrix (mV, mX);
        }
    }
    else if (_epsilon > 0)
    {
        double frob;
        do
        {
            mV = calculateClusterCentres(mU, mX);
             auto mUnew = modifyPartitionMatrix (mV, mX)
                ;

             frob = Frobenius_norm_of_difference (mU,
                mUnew);
            mU = mUnew;
        } while (frob > _epsilon);
    }
   mV = calculateClusterCentres(mU, mX);
    std::vector<std::vector<double>> mS =
        calculateClusterFuzzification(mU, mV, mX);

```

```
      partition part;
      for (int c = 0; c < _nClusters; c++)
      {
         cluster cl;
         for (std::size_t a = 0; a < nAttr; a++)
         {
            descriptor_gaussian d (mV[c][a], mS[c][a]);
            cl.addDescriptor(d);
         }
         part.addCluster(cl);
      }
      return part;
   }
   catch (my_exception & ex)
   {
      throw my_exception (__FILE__, __FUNCTION__,
         __LINE__, ex.what());
   }
   catch (std::exception & ex)
   {
      throw my_exceptionn (__FILE__, __FUNCTION__,
         __LINE__, ex.what());
   }
   catch (std::string & ex)
   {
      throw my_exception (__FILE__, __FUNCTION__,
         __LINE__, ex);
   }
   catch (...)
   {
      throw my_exception (__FILE__, __FUNCTION__,
         __LINE__, "unknown␣expection");
   }
```

```
}
```

List of additional files in electronic submission

Additional files uploaded to the system include:

- **source code of the application,**
- **test data,**
- **a video showing how software or hardware developed for thesis is used**
- **etc.**

List of Figures

List of Tables