Computational Optimization for Association Rule Mining Using Weighted Transactions

A Thesis

submitted to the designated

by the Assembly

of the Department of Computer Science and Engineering

Examination Committee

by

Adam Kypriadis

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN DATA AND COMPUTER SYSTEMS ENGINEERING

WITH SPECIALIZATION

IN DATA SCIENCE AND ENGINEERING

University of Ioannina

School of Engineering

Ioannina 2023

Examining Committee:

- **Konstantinos Parsopoulos**, Professor, Department of Computer Science and Engineering, University of Ioannina (Advisor)
- **Aristidis Likas**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Konstantina Skouri**, Professor, Department of Mathematics, University of Ioannina

Dedication

To my family, partner, and friends for always standing by me.

Acknowledgements

First, I would like to express my gratitude for my supervisor, Professor Konstantinos Parsopoulos, for his consistent guidance, encouragement, and boundless patience throughout this dissertation.

Moreover, I owe special thanks to Professor Isaac Lagaris and PhD candidate Dimitra Triantali for our endless scientific discussions and their invaluable input during this study.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Adam Kypriadis, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2023.
Computational Optimization for Association Rule Mining Using Weighted Transactions.
Advisor: Konstantinos Parsopoulos, Professor

Association rules determine high-utility relations between items present in a transactional dataset. Despite the fact that several studies have attempted to develop effective association rule mining algorithms in order to handle the growing size of real-world data, these efforts have largely focused on mining frequent item combinations rather than information about item combination, quantity, and monetary value.

The present study builds on recent developments to propose an efficient algorithm for mining numerical association rules, in the context of rough values, by utilizing state-of-the-art metaheuristics such as the particle swarm optimization and the real-valued genetic algorithms. The novelty of the proposed approach lies in in the use of revenue-based weights of the dataset's transactions in order to improve the quality and usability of the obtained association rule solutions. Consequently, the proposed method is enabled to provide additional information regarding profitable item combinations alongside their quantities.

Typical statistical experimentation was used to extract the best-suited parameters for the proposed algorithms. Their assessment was based on a large real-world transactional dataset, providing insight regarding the algorithm's ability to extract association rules from real world data as well as the implications of using weighted transactions.

Εκτεταμενη Περιληψη

Αδάμ Κυπριάδης, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2023.
Υπολογιστική Βελτιστοποίηση για την Εξόρυξη Κανόνων Συσχέτισης με Χρήση Βαρών στις Συναλλαγές.
Επιβλέπων: Κωνσταντίνος Παρσόπουλος, Καθηγητής

Στόχος της παρούσας εργασίας είναι η ανάπτυξη ενός αποτελεσματικού αλγορίθμου για την εξόρυξη αριθμητικών κανόνων συσχέτισης από ένα εκτεταμένο σύνολο συναλλαγών που βασίζεται σε δεδομένα μιας πραγματικής εταιρείας. Επιπλέον, εισάγεται η έννοια της ανάθεσης βαρών στις συναλλαγές με στόχο την εξόρυξη ποιοτικότερων κανόνων συσχέτισης, μέσω της μέγιστης αξιοποίησης των διαθέσιμων δεδομένων.

Οι κανόνες συσχέτισης αποτελούν ενδιαφέρουσες σχέσεις μεταξύ αντικειμένων που εμφανίζονται σε ένα σύνολο συναλλαγών. Κάθε κανόνας αποτελείται από δύο μέρη, την «υπόθεση» (antecedent) και το «συμπέρασμα» (consequent). Για την ποιοτική αξιολόγηση των κανόνων αυτών αλλά και για την προσαρμογή των αλγορίθμων που αφορούν στην εξόρυξή τους, χρησιμοποιούνται δύο βασικές μετρικές. Η πρώτη από αυτές είναι η «υποστήριξη» (support), η οποία ορίζεται ως το ποσοστό των συναλλαγών που περιέχουν όλα τα αντικείμενα που συμπεριλαμβάνονται στην υπόθεση και στο συμπέρασμα του εκάστοτε κανόνα. Η δεύτερη ονομάζεται «εμπιστοσύνη» (confidence) και είναι ο αριθμός των συναλλαγών που περιέχουν τα αντικείμενα της υπόθεσης και του συμπεράσματος, προς τον αριθμό των συναλλαγών που περιέχουν τουλάχιστον τα αντικείμενα της υπόθεσης του κανόνα. Οι αριθμητικοί κανόνες συσχέτισης εμπεριέχουν πληροφορία που σχετίζεται με την ποσότητα των αντικειμένων της υπόθεσης και του συμπεράσματος, σε αντίθεση με

τους απλούς κανόνες συσχέτισης που περιέχουν πληροφορία μόνο για την παρουσία ή μη του κάθε αντικειμένου.

Στον σύγχρονο κόσμο των επιχειρήσεων είναι φανερό πως η αξία μιας συναλλαγής δεν μπορεί να εκτιμηθεί πλήρως μέσω της παρουσίας και της ποσότητας των αντικειμένων που συμπεριλαμβάνει. Λόγω αυτού, στην παρούσα εργασία αναθέτουμε επίσης βάρη στις συναλλαγές που, για την συγκεκριμένη εφαρμογή και σύνολο δεδομένων, εκφράζουν το ποσοστό του κέρδους που έχει αποφέρει στην επιχείρηση η κάθε συναλλαγή. Τα παραπάνω βάρη μπορούν να αντιστοιχηθούν με οποιαδήποτε εναλλακτική ποσότητα που εκφράζει σημαντικότητα σε διαφορετικές εφαρμογές, προσφέροντας έτσι έναν άμεσο τρόπο προσαρμογής τους αλγόριθμου σε διαφορετικά σύνολα δεδομένων. Τα βάρη χρησιμοποιούνται για να ορίσουμε εκ νέου την υποστήριξη και την εμπιστοσύνη, έτσι ώστε ο αλγόριθμος να παράγει πιο κερδοφόρους κανόνες συσχέτισης.

Το σύνολο δεδομένων που χρησιμοποιήθηκε για την εργασία αποτελείται από τις συναλλαγές μιας πραγματικής επιχείρησης σε διάστημα τριών ετών. Η έκταση του συνόλου αυτού, το οποίο περιέχει χιλιάδες αντικείμενα που συμμετέχουν σε εκατοντάδες χιλιάδες συναλλαγές, καθώς επίσης η αραιότητά του και οι ασυνέχειες της προς μεγιστοποίηση συνάρτησης καθιστούν την χρήση παραδοσιακών μεθόδων βελτιστοποίησης με παραγώγους αδύνατη. Έτσι επιλέχθηκαν οι στοχαστικές μέθοδοι της βελτιστοποίησης με σμήνος σωματιδίων και των γενετικών αλγόριθμων, οι οποίες αντιμετωπίζουν επιτυχώς την ασυνέχεια της αντικειμενικής συνάρτησης απαιτώντας παράλληλα μικρό υπολογιστικό κόστος. Τέλος, παρουσιάζονται τα αποτελέσματα στατιστικής ανάλυσης για την εξαγωγή των καλύτερων παραμέτρων του αλγορίθμου στο παραπάνω σύνολο δεδομένων, καθώς επίσης και η μέθοδος που χρησιμοποιήθηκε για την παράλληλη εκτέλεση των απαιτούμενων δοκιμών σε μία συστάδα υπολογιστών.

Chapter 1

Introduction

1.1 Objectives

As the economy has been steadily growing, corporations are becoming increasingly complex constructs. Their exponential growth has created the need for progressively bigger and more accurate data for their daily operations. Alongside the economic system, algorithms have evolved to be more efficient and precise. While information about profitable item combinations has been in need for as long as the trading of goods has existed, it is recently that the need for efficiency made traditional methods, ranging from the human experience to exhaustive search algorithms, obsolete. A significant amount of research has been conducted to construct optimization-based algorithms to uncover high utility item combinations, referred to as association rules, a term coined by [1].

The main objective of the present thesis is the development of an efficient numerical association rule mining method using weighted transactions. The algorithm's numerical aspect refers to its ability to provide information about the quantities of the involved items in the transactional dataset, whereas the novelty of assigning weights to transactions based on economic revenue allows the algorithm to produce high-quality rules by transforming the algorithm's output solutions from frequent item combinations to profitable item combinations.

The core method was developed and tuned using a large and sparse real-world dataset. It comprises information about the revenue of each transaction, the customer identification number, and the month it occurred for a Greek company over a three-year period. Naturally, mining for association rules on a dataset of this magnitude of more than 130000 transactions and 2300 items is computationally demanding, hence a heuristic optimization method is best suited against traditional approaches.

Particle swarm optimization and real-valued genetic algorithms are robust, stochastic methods for locating the minima (or maxima) of a given objective function. They are particularly suited for our application in association rule mining since, unlike standard optimization approaches, they do not require the objective function to be continuous or differentiable. Specifically for particle swarm optimization, in order to achieve a better solution each particle moves across the search space based on an adaptable velocity that combines the best position it has visited, as well as the best position visited by particles belonging in its neighborhood. Similarly, real-valued genetic algorithms generate new candidate solutions by combining some of the best positions discovered previously. The stochastic nature of these methods enables them to achieve a solution of comparable quality in a fraction of the computational budget required by an exhaustive search.

The suggested algorithm is applied to the aforementioned large dataset, along with a statistical analysis of the solutions' quality for different parameter combinations. As a result, a set of promising values for each one of the parameters is obtained.

1.2 Structure of the Thesis

The present thesis contains 4 chapters, and is organized as follows: Chapter 2 contains background information. This includes association rules, the particle swarm optimization method, and real-valued genetic algorithms. In Chapter 3, the proposed approach is presented, while Chapter 4 contains experimental analysis, the obtained results, and implementation details.

Chapter 2

Background Information

2.1 Association Rules

2.1.1 Relevant definitions and concepts

The exchange of goods between firms, individuals, and other economic entities is a crucial aspect of our society and economy. Due to the rising stability of this established economic system, the numerous transactions might reveal evolving but persistent patterns. Association rules are valuable relations between sets of items in a transactional dataset [1]. Their utility is measured using a range of quality metrics, the most typical of which are support and confidence. More specifically, given an itemset of n items,

$$I = \{i_1, i_2, ..., i_n\}, \tag{2.1}$$

a set of m transactions,

$$T = \{t_1, t_2, ..., t_m\}, \tag{2.2}$$

consists of subsets of the itemset. Each transaction is represented by an n dimensional vector

$$t_i = \{t_{i_1}, t_{i_2}, ..., t_{i_n}\} \tag{2.3}$$

where the components t_{i_j} may take two values:

$$t_{i_j} = \begin{cases} 1, & \text{if item } j \text{ is present in transaction } i, \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

Association rules are defined as relations between one subset of the itemset to another. Thus, an association rule between subsets A and C denoted as

$$A \rightarrow C, \tag{2.5}$$

can be interpreted as "The presence of the items of set A in a transaction means that the items of set C have a chance p of also taking part in the transaction". The subset A is called the "antecedent" of the rule and C is called the "consequent". In order to define the quality of an association rule, the following metrics are commonly used:

1. $Support$: The total number of transactions containing the items of both the antecedent and the consequent, divided by the total number of transactions:

$$Support = \frac{\Big|\{A\} \cup \{C\}\Big|}{Total\,\#\,of\,transactions} \tag{2.6}$$

2. $Confidence$: The total number of transactions that include the items of both the antecedent and the consequent of the rule, divided by the total number of transactions that include the items of the antecedent:

$$Confidence = \frac{\Big|\{A\} \cup \{C\}\Big|}{\Big|\{A\}\Big|} \tag{2.7}$$

Thus, $support$ can be considered to be the measure of rule frequency, and $confidence$ the measure of rule robustness.

2.1.2 Numerical association rule mining

The majority of previous studies have focused on binary representations both in the transactional dataset and the rules produced. As an example, consider the following rule:

$$\text{Beef} + \text{Ketchup} \rightarrow \text{Burger buns},$$

which can be interpreted as: "When beef and ketchup are purchased together there is strong likelihood that burger buns will also be included in the same transaction".

Our approach is based on [2] and focuses on numerical association rule mining, which implies that the amount of an object must also be considered. An example of such a transaction would be:

$$0.5kg \text{ of beef} + 300ml \text{ of ketchup} + 8 \text{ burger buns} \qquad \text{Revenue: } \$25$$

and the resulting association rules that contain two bounds for each product take the following form:

$$\text{beef} \in [0.25, 1.5] + \text{ketchup} \in [200, 600] \rightarrow \text{burger buns} \in [4, 10]$$

which could be interpreted as "When between 0.25 and 1.5 kilograms of beef are purchased with between 200 and 600 milliliters of ketchup, there is strong likelihood between 4 and 10 burger buns are going to be also purchased".

2.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO), introduced by R.C.Eberhart and J.Kennedy [3], is a heuristic designed for numerical optimization especially in cases where derivatives do not exist or they are difficult to compute. As described in [4], the process begins by randomly initializing a number of particles (search points) within the search space:

$$S = \{x_1, x_2, ..., x_N\}, \tag{2.8}$$

where N is the number of particles that constitute the swarm.

Each particle is represented as an n-dimensional vector,

$$x_i = (x_{i_1}, x_{i_2}, ..., x_{i_n}), \quad i = 1, ..., N. \tag{2.9}$$

It also has an adaptable velocity

$$v_i = (v_{i_1}, v_{i_2}, ..., v_{i_n}), \quad i = 1, ..., N, \tag{2.10}$$

which is updated for every iteration by combining the best position that the particle has detected itself, also called the "personal best", and the best position of the particle's neighborhood. Particle i's neighborhood can be represented as follows:

$$NG_i = \{i_1, i_2, ..., i_k\} \subseteq \{1, 2, ..., N\}, \quad 1 \leq k \leq N. \tag{2.11}$$

Two common PSO models with regard to the neighborhood type are:

(a) *gbest*: $NG_i = \{1, 2, ..., N\}$

(b) *lbest*: $NG_i = \{i - a, ..., i, ..., i + a\}$.

In the first case, particle i's neighborhood is considered to include all the particles of the swarm. In the second, it consists of particle i, and particles with adjacent indexes, in a cyclic manner. Its extent is defined by radius a.
The velocities and positions for iteration $t + 1$ are calculated as follows:

$$v_{i_j}^{(t+1)} = \chi[v_{i_j}^{(t)} + rand()c_1(p_{i_j}^{(t)} - x_{i_j}^{(t)}) + rand()c_2(p_{g_{i_j}}^{(t)} - x_{i_j}^{(t)})] \quad (2.12)$$

$$x_{i_j}^{(t+1)} = x_{i_j}^{(t)} + v_{i_j}^{(t)} \quad (2.13)$$

$$i = 1, 2, ..., N, \quad j = 1, 2, ..., n,$$

where:

1) g_i is the index of the best particle of the NG_i neighborhood:

$$g_i = \arg\min_{j \in NG_i} f(p_j^{(t)}),$$

2) $c_1, c_2 > 0$ are the cognitive and social parameters, respectively,

3) χ is the constriction coefficient,

4) $rand()$ are random numbers in $[0, 1]$.

So every particle's new position is calculated based on:

(a) Its inertia $v_i^{(t)}$.

(b) Its attraction towards the best position $p_i^{(t)}$ it has visited.

(c) Its neighborhood's best position $p_{g_i}^{(t)}$.

It is important to impose a limit, v_{max}, on velocities in order to prevent particles from frequently departing the search space. These limits are usually set as percentage of the search space. The velocity for particle i's component j is limited as follows

$$-v_j^{max} \leq v_{i_j} \leq v_j^{max}, \quad (2.14)$$

$$v_j^{max} = a(u_j, l_j), \quad a \in (0, 1], \quad (2.15)$$

where

Algorithm 2.1 PSO pseudocode

Input: search space $X, \chi, c_1, c_2, v_{\max}$

$t \leftarrow 0$

$x^{(t)} \leftarrow$ Initialize(X)

$v^{(t)} \leftarrow 0$

Evaluate particles

$p^{(t)} \leftarrow$ updateBests$(v^{(t)}, x^{(t)})$

while termination condition not true **do**

$v^{(t+1)} \leftarrow$ calcVelocities$(v^{(t)}, x^{(t)}, p^{(t)}, \chi, c_1, c_2)$

check bounds$(v^{(t+1)}, v_{\max})$

$x^{(t+1)} \leftarrow$ updatePositions$(v^{(t+1)}, x^{(t)}, X)$

Evaluate particles

$p^{(t+1)} \leftarrow$ updateBests$(v^{(t+1)}, x^{(t)})$

$t++$

end while

Return best

1) a: denotes the fraction of the search space a particle is allowed to cross during a single iteration,

2) u_j, l_j: are the upper and lower bounds, respectively, for the j-th dimension of the search space.

Then, the objective values are calculated for the particles and, the personal bests are updated as follows:

$$p_i^{(t+1)} = \begin{cases} x_i^{(t+1)}, \text{ if } f(x_i^{(t+1)}) < f(p_i^{(t)}) \\ p_i^{(t)}, \text{ otherwise.} \end{cases}, \quad i = 1, 2, ..., N \tag{2.16}$$

The PSO algorithm is given in pseudocode in Algorithm 2.1.

2.3 Real-valued Genetic Algorithm

Genetic algorithms (GA), introduced by J.H. Holland [5], are the main type of evolutionary metaheuristics, and they are inspired by the process of natural evolution. As described in [6], a population of selectable size is initialized across the search space and then undergoes the processes of selection, mutation, and recombination in order to discover new candidate solutions.

2.3.1 Representation

Candidate solutions also called the members of the population, are represented as vectors. These vectors are inputs to the objective function, which is the function to be minimized and is used to assess their quality. The i-th member of the population is represented as,

$$p_i = (p_{i_1}, p_{i_2}, ..., p_{i_n}), \quad i = 1, ..., N, \tag{2.17}$$

where N is the population size and n the dimension of the search space.

2.3.2 Selection

Selection is the process of choosing some members of the population to be combined in order to form the next generation of candidate solutions. Tournament selection is a typical method in which population S of N members is produced by repeatedly selecting randomly N_t (tournament size) members and selecting the best among them. In order to select member s_i, N_t members are selected randomly,

$$PR = \{p'_1, p'_2, ..., p'_{N_\mathrm{t}}\}, \tag{2.18}$$

$$p'_i \in \{p_1, p_2, ..., p_N\}, \quad i = 1, ..., N_\mathrm{t}, \tag{2.19}$$

and the best of them is assigned to s_i

$$s_i = \mathrm{best}(PR). \tag{2.20}$$

The tournament selection algorithm is given in pseudocode in Algorithm 2.2.

2.3.3 Recombination

The recombination operator acts on the newly selected population to numerically recombine some of them, and forms population R. After ensuring that the number

Algorithm 2.2 Tournament selection pseudocode

1: **for** $n = 1, .., N$ **do**
2: Select N_t members randomly
3: $s_n \leftarrow$ best of selected
4: **end for**

of selected members is even, the members are split in pairs, and for every one an offspring is produced.

The recombination of two members of the population p_i, p_j produces an offspring

$$o = (o_1, o_2, ..., o_n), \tag{2.21}$$

where:

$$o_k = r_k p_{i_k} + (1 - r_k) p_{j_k}, \tag{2.22}$$

$$r_k \sim U([-\delta, 1 + \delta]), \quad \delta > 0. \tag{2.23}$$

Parameter δ aids in preventing the gradual shrinkage of the search space.

2.3.4 Mutation

Mutation is the final operation in which, noise is added to randomly selected components of the population vectors, in order to produce population M. Suppose component p_{i_j} has been selected for mutation. Then it is perturbed as follows:

$$p_{i_j} \leftarrow p_{i_j} + z_j, \tag{2.24}$$

where

$$z_j \sim N(0, \sigma^2) \quad \text{or} \tag{2.25}$$

$$z_j \sim U([-a, a]). \tag{2.26}$$

Special caution is required so that the resulting value remains inside the search space, which is achieved by selecting suitable values of σ or a, for the Gaussian or uniform distribution, respectively.

Then population M_t is evaluated, and combined with P_t in order to form P_{t+1}. This combination may take the form of complete replacement,

$$P_{t+1} = M_t, \tag{2.27}$$

Algorithm 2.3 Genetic algorithm pseudocode

Input: search space X, N_{t}
$k \leftarrow 0$
$P_k \leftarrow$ Initialize(X)
evaluate(P_k)
$x_b \leftarrow$ update best(P_k)
while termination condition not true **do**
  $S_k \leftarrow$ selection(P_k, N_{t})
  $R_k \leftarrow$ recombination(S_k)
  $M_k \leftarrow$ mutation(R_k)
  evaluate(M_k)
  $P_{k+1} \leftarrow$ new population(M_k, P_k)
  $x_b \leftarrow$ update best(P_{k+1})
  $k++$
end while
Print best x_b

or the best N particles of M_t and P_t may be chosen to form P_{t+1},

$$P_{t+1} = \text{best}_N(M_t \cup P_t). \tag{2.28}$$

Lastly, x_b, which holds the best discovered candidate solution, is updated accordingly. The GA algorithm is given in pseudocode in Algorithm 2.3.

Chapter 3

The proposed approach

3.1 Dataset and Preprocessing

The real-world dataset used for this application contains transactions of a Greek corporation that span 3 calendar years. The selection of a real world dataset with more than 130000 single-item transactions (referred to as pre-transactions) and 1830 items, in contrast to an artificial one, provides an indication that the algorithm can successfully cope with big and noisy data. Pre-transactions are coded as matrix rows that include the item's id, the client's id, the amount of the item purchased, the month the pre-transaction occurred, and the revenue it has brought to the company, as reported in Table 3.1.

In order to transform the original dataset that consists of single-item transactions to the transactional dataset required by the algorithm, we considered a transaction to contain the items purchased by a single client during a month. This is calculated by

Table 3.1: Dataset format

Pre-transaction No	Client id	Item id	Month	Amount	Revenue
1	2152	1267	3	50.6	145.5
2	1744	2189	6	10.0	77.0
...					

summing every item's amount for every client, and month of the 3 years. Transaction i is represented as follows:

$$t_i = (sumItem_{1jk}, sumItem_{2jk}, ..., sumItem_{N_{items}jk}), \tag{3.1}$$

where:

$i = 1, 2, ..., 3 * 12 * N_{clients}$ is the transaction's index,

$j = 1, ..., N_{clients}$ is the client's id,

$k = 1, ..., 36$ is the corresponding month,

$sumItem_{tjk}$ is the amount of item t for client j, purchased during the k-th month.

Empty or 1-item lines, which correspond to months that specific clients did not make any purchase or purchased just one single item, are eliminated from the transactional database. The total revenue of the pre-transactions that comprise a transaction is calculated and stored separately for every transaction. The minimum and maximum values for each item are also stored and define the search space limits to prevent the solver from drifting out of the search space.

3.2 Weighted Transactions

Most previous studies on association rule mining have evaluated the association rules produced by the algorithms using the standard quality metrics, namely support and confidence. However, in a real world business environment the high frequency of an itemset does not necessarily lead to high profitability for the company. Using weights for the dataset transactions can tackle this issue, by emphasizing on the profitability of

the company thereby producing association rules of relevant quality. For the specific dataset, the weight of a transaction is defined as the transaction's economic revenue divided by the total revenue of all transactions. While revenue-based weights are appropriate for this dataset, this novel method provides an easy way of adjusting the algorithm's orientation towards any metric of interest on different applications.

The redefined rule quality metrics are calculated as follows:

(a) Each transaction's weight is calculated as the revenue of the transaction divided by the sum of revenues of all transactions:

$$w_i = \frac{i\text{-th transaction's revenue}}{\text{sum of revenues}}. \tag{3.2}$$

(b) Support of rule R: $A \rightarrow C$ is redefined as the sum of weights of the transactions that conform to the rule divided by the sum of weights:

$$Support = \frac{\sum\limits_{t_i \in T(A \cup C)} w_i}{\sum\limits_{t_j \in T} w_j}. \tag{3.3}$$

(c) Confidence of rule R: $A \rightarrow C$ is redefined as the sum of weights of the transactions that conform to the rule, divided by the weights of the transactions that include the items of the antecedent of the rule:

$$Confidence = \frac{\sum\limits_{t_i \in T(A \cup C)} w_i}{\sum\limits_{t_j \in T(A)} w_j}. \tag{3.4}$$

The redefined quality metrics have been used in our application.

3.3 Solution Representation and Initialization

In the considered PSO and GA solvers, search points that correspond to candidate rule solutions are represented as vectors of length $3 * N_{\text{items}}$:

$$\text{rule}_i = (lb_{i1}, ub_{i1}, p_{i1}, lb_{i2}, ub_{i2}, p_{i2}, ..., lb_{iN_{\text{items}}}, ub_{iN_{\text{items}}}, p_{iN_{\text{items}}}), \tag{3.5}$$

where N_{items} is the number of total items present in the dataset. So participation of item j in rule i is denoted by three numbers:

(1) lb_{ij}: lower bound for the item's quantity

(2) ub_{ij}: upper bound for the item's quantity

(3) p_{ij}: participation indicator in the interval $[0, 1]$, defined as follows:

(a) If $p_{ij} \in [0, 1/3)$ then item j is not part of rule i.

(b) If $p_{ij} \in [1/3, 2/3)$ then item j is included in the antecedent of rule i.

(c) If $p_{ij} \in [2/3, 1]$ then item j is included in the consequent of rule i.

Transactions are represented as vectors of length (N_{items}):

$$t_k = (A_{k1}, A_{k2}, ..., A_{kN_{\text{items}}}) \tag{3.6}$$

where A_{kj} is the amount of item j in transaction k, and N_{items} is the number of items present in the dataset.

The initial positions of search points that define rules, either corresponding to particles for PSO, or population members for GAs, are of great importance, especially in the case of a high dimensional search space like the one used in our application. Exploiting the existing knowledge in the database, the points' initial values are selected around randomly chosen transactions. Thus, a rule

$$\text{rule}_i = (lb_{i1}, ub_{i1}, p_{i1}, lb_{i2}, ub_{i2}, p_{i2}, ..., lb_{iN_{\text{items}}}, ub_{iN_{\text{items}}}, p_{iN_{\text{items}}}), \tag{3.7}$$

is initialized around a randomly selected transaction

$$t_j = (A_{j1}, A_{j2}, ..., A_{jN_{items}}), \tag{3.8}$$

as follows:

(a) For items that are present in transaction j, i.e.,

$$A_{j*} > 0,$$

rule bounds are defined as:

$$lb_{i*} = A_{j*} - r_1, \quad r_1 \in [0, 20\% * (\max_i - \min_i)]$$

$$ub_{i*} = A_{j*} + r_2, \quad r_2 \in [0, 20\% * (\max_i - \min_i)]$$

$$p_{i*} = r_3, \quad r_3 \in [1/3, 1]$$

Algorithm 3.1 Initialization of search points

for every search point **do**
  Select transaction t randomly
  for $k = 1, ..., N_{\text{items}}$ **do**
    if item k is included in t **then**
      Set ub_k and lb_k so that $t(k) \in [lb_k, ub_k]$
      Set $p_k \in [1/3, 1]$
    else
      Set $ub_k = \max k$ and $lb_k = \min k$
      Set $p_k \in [0, 1/3)$
    end if
  end for
end for
Return initialized points

(b) For items that are not present in transaction j, i.e.,

$$A_{j*} = 0,$$

rule bounds are set as:

$$lb_{i*} = min_i, \quad ub_{i*} = max_i, \quad p_{i*} = r_3, \quad r_3 \in [0, 1/3),$$

where $\min_i$ is the minimum non-zero amount of item i found in the dataset, $\max_i$ is the maximum amount of item i found in the dataset, and r random numbers.

Another reason for the initialization of search points around existing transactions is to ensure that at the beginning of the algorithm, every member of the swarm has a positive fitness value. Alternatively, if the search points were initialized randomly across the search space, there would exist a significant likelihood that all of them assume near-zero fitness values due to the sparsity of the dataset. In this scenario, there would be no meaningful comparison between them, therefore there would be no best search point to lead the swarm or population, for PSO and GA respectively, towards optimal solutions.

The initialization procedure is given in pseudocode form in Algorithm 3.1.

3.4 Objective Function

An efficient implementation requires a rule quality measure that adheres to the basic assumptions of this specific application. The ultimate goal is to find rules of the highest possible quality, also called the rule's fitness, which translates to maximizing the objective function. Selecting the objective function is not a trivial task since a slight imbalance between confidence and support on either side may result in rules of low value. Example scenarios of imbalanced objective function are presented below:

(Example 1) Support imbalance: If the objective function is prone to high support, the algorithm may discover rules with frequent items or high total revenue (in the case of revenue based weighted transactions), but the rules would lack robustness. For example, a supermarket application would include rules of the type:

$$\text{plastic bag} \rightarrow \text{milk},$$

which are characterized by low confidence since plastic bags are not purchased solely or primarily with milk. We can make the assumption that this rule does not provide the business with valuable information as the extracted information about milk could be obtained by analyzing milk sales alone.

(Example 2) Confidence imbalance: If the objective function rewards high confidence, the algorithm may concentrate around a single or few specific transactions, so that the antecedent is found nowhere else in the transactional database. Thus, the produced rule would be highly infrequent or unprofitable on the case of weighted transactions. In the supermarket example above, we could have rules of type:

$$\text{beef} \in [0.2499, 0.2501] + \text{ketchup} \in [199, 201] \rightarrow \text{burger buns} \in [5, 5]\,,$$

which is concentrated on specific transaction instances.

(Example 3) A third aspect specific to our application on numerical association rule mining is the width of the rule bounds. If no constraint is imposed to the fitness function, rule bounds tend to expand in order to match more transactions. An example for the supermarket application would include rules of type:

$$\text{beef} \in [0, 10000] + \text{ketchup} \in [0, 20000] \rightarrow \text{burger buns} \in [0, 50000]\,,$$

while a rule of narrow margins and possibly lower support or confidence would be far more useful, e.g.:

$$\text{beef} \in [0.25, 1.5] + \text{ketchup} \in [200, 600] \rightarrow \text{burger buns} \in [4, 10]\,.$$

Algorithm 3.2 Find Matching Transactions

Input: rule , $transactions$
Logical:: $matches(N_{\text{tran}})$
$matches(1 : N_{\text{tran}}) \leftarrow True$
for $j = 1, 4, ..., 3 * N_{\text{items}}$ **do**
  if rule$(j + 2) < 0.33$ **then**
    continue
  end if
  $item \leftarrow (j + 2)/3$
  for $i = 1, ..., N_{\text{tran}}$ **do**
    if $matches(i) == False$ **then**
      continue
    else if $transactions(item, i) <= 0.001$ **then**
      $matches(i) = False$
    else if rule$(j) > transactions(item, i)$ **then**
      $matches(i) = False$
    else if rule$(j + 1) < transactions(item, i)$ **then**
      $matches(i) = False$
    end if
  end for
end for
Return $matches$

In order to prevent rule bounds from arbitrarily expanding, a penalty term is introduced in the objective function. More specifically, the sum of the normalized width of the bounds for items included in the rule is subtracted from the rule's objective function value. This handling has the additional implication of limiting the products included in the produced rules, which is important when dealing with itemsets containing thousands of items.

Based on the above analysis, the final form of the objective function is:

$$f = a * \text{Confidence} + b * \text{Support} - c * \text{Penalty} \tag{3.9}$$

where a, b, and c are constants defined through experimentation for rules that contain items in both the antecedent and the consequent. Our analysis for the specific dataset

suggested the following suitable values:

$$a = 10^2, \quad b = 10^5, \quad c = 10^{-1},$$

which were experimentally proven to achieve balanced solutions, considering support and confidence, for the specific dataset.
For rules that lack items in the antecedent or the consequent, the corresponding values are:

$$a = 0, \quad b = 0,$$

as these rules have no meaning.

Since Eq. (3.9) is obviously discontinuous, optimization algorithms that rely on derivative information become inapplicable. This is one of the main reasons that metaheuristics such as PSO and GA have been selected for the present thesis. In order to calculate support and confidence, the following procedure is used:

Transaction j takes part in the i-th rule's calculation of support and confidence, if all of the items included in the rule

$$\text{rule}_i = (lb_{i1}, ub_{i1}, p_{i1}, lb_{i2}, ub_{i2}, p_{i2}, ..., lb_{iN_{\text{items}}}, ub_{iN_{\text{items}}}, p_{iN_{\text{items}}}),$$

such that $p_{i*} \in [1/3, 1]$, are also present in transaction

$$t_j = (A_{j1}, A_{j2}, ..., A_{jN_{\text{items}}}),$$

and all the amounts of these items fall between the rule's lower and upper bounds:

$$A_{j*} \in [lB_{i*}, uB_{i*}], \text{ for every item that } p_{i*} \in [1/3, 1]. \tag{3.10}$$

The procedure is given in pseudocode form in Algorithm 3.2.

3.5 Restarting the Algorithm

Restarting the particles is another key element of the method. The necessity derives from the fact that convergence is typically achieved in a fraction of the available computational budget, regardless of being defined as the number of function evaluations (most commonly) or as the algorithm's iterations limit.

A simple approach would be to periodically reinitialize the particles when a specific fixed number function evaluations have elapsed without detecting a new best position. However, this is inefficient due to the following peculiarities:

(a) When restarting occurs, the swarm may have not fully converged. Therefore, the particles may be prevented from approaching a better position.

(b) The particles may have fully converged, keep spending computational resources until the predefined limit is reached, without significant solution improvement.

A more advanced strategy would be to use as restarting criterion the standard deviation among the particles' positions. This would be close to zero if the particles had converged to a single point in the search space. A disadvantage of this strategy appears in the case where some particles get trapped due to their personal best and the neighborhood best canceling each other out, resulting in particle velocity close to zero at each subsequent iteration of the algorithm. If this situation emerges during the algorithm's later stage, when most other particles have converged close to the swarm's best position, there is limited chance of finding a new best position. Thus, the trapped particle would remain almost stationary.

In order to overcome this obstacle, it is common practice to restart the algorithm when the standard deviation is close to zero, indicating that the majority of the particles have converged close to the swarm's best position. The exact limit of the standard deviation that triggers the algorithm to restart is application-specific and introduces an additional parameter that needs to be defined.

The approaches above do not work in their standard form in our application on numerical association rule mining. This is attributed primarily to the dataset's high dimension that renders the calculation of the standard deviation of particle positions computationally expensive. A less expensive alternative would be the use of the standard deviation of the particles' function values. An obvious disadvantage of this choice is the misleading information of the objective values near discontinuities. Since each particle is initialized around an existing transaction, the corresponding rule has positive support and confidence at the beginning of the algorithm. Due to the size of the considered dataset that consists of thousands of items, the particles that move towards the best frequently pass across points in the search space that correspond to rules with near zero objective values, indicating the lack of transactions containing the rules' product combinations. Consequently, many particles possess near zero values for many iterations, resulting also in near zero standard deviations of the objective values, although the swarm has not converged yet.

These obstacles can be addressed by combining the two strategies for restarting

the algorithm. Hence, we may restart under 2 conditions:

(a) Restarting condition 1: The average deviation of the particles' function values from the swarm's best approaches zero. This condition effectively treats the early restart problem that would arise under the mainstream approach, as the best can only be a valid rule with positive associated objective value. This is due to the initialization process, which ensures that particles include at least one transaction.

(b) Restarting condition 2: A new best has not been discovered after a specific fixed number of iterations. This limit is set through statistical experimentation to a relatively high value, as a secondary measure. It is only activated on the rare case that too many particles get trapped far away from the swarm's best, preventing the average deviation from the best being further reduced.

It is important to note that this process involves a complete restart of the algorithm. The swarm's overall best position is saved to a separate variable and the swarm's best is redefined among them, as if the algorithm has no prior knowledge of what it has discovered so far. This prevents the algorithm from converging to the same solution.

3.6 Search Space Boundaries

Search space boundaries are usually related to the specific application. Three separate restrictions are defined in our case:

(a) $Upper\ bound\ limits$: The upper bound for an item in a rule is limited to the maximum amount present in the transactional dataset for the item.

(b) $Lower\ bound\ limits$: The lower bound for an item in a rule is prevented from falling below the minimum amount present in the transactional dataset for the item.

(c) $Velocity\ limit$: In order to avoid particles that overfly significant parts of the search space that may contain optimal solutions, a limit as the percentage of the search space a particle is permitted to move during a single iteration of the algorithm must be specified.

If a particle's new position falls outside these limits, the new position is set exactly on the limit. A complete walkthrough of the algorithm can be found in Section 4.1.

3.7 Tunable Parameters

Obtaining suitable parameter values for the proposed algorithm may have a major impact on solution quality and necessary computation budget. The high dimension of the dataset is prohibitive for mining good association rules through exhaustive search. Thus, the detection of good rules that could be used as compass for fine-tuning the proposed algorithm is not feasible. Another obstacle on the path of extracting optimal parameters is their dependency on the dataset. Hence, a trivial dataset with known solutions cannot be used either.

A common practice for tuning stochastic algorithms is to extract the best set of parameters via preliminary statistical analysis. In a full factorial design, a number of candidate values is set for each parameter, and all their possible combinations are assessed under the average solution quality over a number of experiments.
The parameters considered for PSO in our application are the following:

(1) Swarm size: If the computational budget for the execution of the algorithm is considered to be constant, a balance between the number of particles and the algorithm's iterations, which are inversely related, must be achieved.

(2) Velocity limit: It defines the maximum distance a particle can move in a single iteration of the algorithm. These limits are usually set as percentage of the search space, and prevent the swarm from frequently departing the search space.

(3) Neighborhood type: It specifies if each particle moves towards the best particle of the swarm (gbest model), or towards the best of neighborhood (lbest model).

(4) Restarting limit: It defines the number of iterations of not discovering a new overall best position before the algorithm restarts.

The parameters considered for the real-valued GA are the following:

1. Population size: Similar to PSO's swarm size above.

2. N_t: The number of selected population members during tournament selection. The best among them is selected to participate in recombination.

3. Crossover rate: This is a real number in $[0, 1]$. During recombination, a random number in $[0, 1]$ is drawn for every member of the population. If this number is greater than the crossover rate, the member is chosen for recombination.

4. Mutation rate: This is also a real number in $[0, 1]$. During mutation a random number in $[0, 1]$ is drawn for every component of every member of the population. If this number is greater than mutation rate, the corresponding component is mutated (perturbed).

5. Restarting limit: Same as PSO above.

Experimental analysis on the aforementioned parameters is presented in Section 4.2.

Chapter 4

Experimental Analysis

4.1 Implementation Details

A very efficient implementation of the proposed algorithm was required in order to cope with the total number of more than 2950000000 objective function calls needed for extracting the best suited parameters for the two methods, namely PSO and GAs. Moreover, the high dimension of the vectors that represent candidate rule-solutions, which is equal to 5490 ($=3*1830$), where 1830 is the number of non-zero items in the selected dataset, imposed difficulties on memory management and data distribution among physical machines. For this reason, the implementation was based on Fortran using OpenMP for the parallelization of specific procedures described below.

Algorithm 4.1 Main Algorithm

1: **Input:** $totalRuns, computationalBudget, neighborhoodType,$
2: $\quad swarmSize, restartingLimit, v_{\max}, \chi, c_1, c_2$
3: $sentJobs[1] \leftarrow 0$
4: $completedJobs[1] \leftarrow 0$
5: initialize transactions
6: $X \leftarrow$ calculate limits
7: **while** $sentJobs[1] < totalRuns$ **do**
8: $\quad sentJobs[1] \leftarrow sentJobs[1] + 1$
9: $\quad t \leftarrow 0$
10: $\quad x^{(t)} \leftarrow Initialize(X)$
11: $\quad v^{(t)} \leftarrow 0$
12: $\quad$ Evaluate particles
13: $\quad p^{(t)} \leftarrow$ updateBests$(v^{(t)}, x^{(t)})$
14: $\quad$ **while** $computationalBudget$ not exhausted **do**
15: $\quad\quad$ checkRestart$(restartingLimit)$
16: $\quad\quad v^{(t+1)} \leftarrow$ calcVelocities$(v^{(t)}, x^{(t)}, p^{(t)}, \chi, c_1, c_2)$
17: $\quad\quad$ check bounds$(v^{(t+1)}, v_{\max})$
18: $\quad\quad x^{(t+1)} \leftarrow$ updatePositions$(v^{(t+1)}, x^{(t)}, X)$
19: $\quad\quad$ Evaluate particles
20: $\quad\quad p^{(t+1)} \leftarrow$ updateBests$(v^{(t+1)}, x^{(t)})$
21: $\quad\quad t++$
22: $\quad$ **end while**
23: $\quad completedJobs[1] \leftarrow completedJobs[1] + 1$
24: $\quad$ Write solution to file
25: **end while**
26: **while** $.True.$ **do**
27: $\quad$ **if** $completedJobs[1] == totalRuns$ **then**
28: $\quad\quad$ exit
29: $\quad$ **end if**
30: **end while**

The algorithm's input includes the following parameters:

1) $totalRuns$: The number of independent experiments, in our case 25 experiments

for each set of parameters.

2) $computationalBudget$: The number of function evaluations per run.

3) The PSO parameters: neighborhood type, swarm size, restarting limit, χ, $v_{\max}$, c_1, c_2.

Then, copies of the program (images) equal to the number of physical machines available in the computer cluster are initialized, using the Fortran's Coarray programming. When an image completes a total run of the algorithm it is assigned another one, and so on until all the requested runs (parameter: $totalRuns$) are completed. Using OpenMP libraries, each image is able to execute code in parallel.

A computer cluster comprised of 7 physical machines outfitted with Intel i7 processors was used for the completion of this thesis. It took about 48 hours to complete the 3000 runs of the algorithm, each with a computational budget of 10^6 function evaluations.

Algorithm 4.1 reports the complete pseudocode of the parallel implementation for the PSO algorithm. The GAs case was implemented following exactly the same approach. It is trivial for the user to make the corresponding modification for that case.

4.2 Experimental and Parameter Setting

In order to extract the most efficient parameter values for the considered algorithms, different levels were set for each parameter, and their possible combinations were assessed in terms of the average solution quality over 25 independent experiments. Each experiment assumed a computational budget of 10^6 objective function calls.
The resulting 54 parameter combinations for PSO were obtained by combining the following values:

1. Swarm size: $20, 200, 400$
2. Velocity limit: $10\%, 50\%, 100\%$ of the search space
3. Neighborhood type: $gbest, lbest$
4. Restarting limit: $100, 500, 1000$ iterations

The 64 parameter combinations for the GAs were obtained for the following values:

1. Population size: $20, 200$
2. Tournament size: $10\%, 50\%$ of the population size
3. Recombination rate: $0.9, 0.6$
4. Mutation rate: $0.01, 0.02, 0.03, 0.04$
5. Restarting limit: $100, 300$

Each experiment yielded the following information, which was used to evaluate and pick the optimal parameter set for each solver:

1. The maximum achieved objective function value.
2. The corresponding solution.
3. The iterations of the algorithm needed to reach the solution, also called the $last\ hit$.

For each parameter combination, the minimum, maximum, average, median, and standard deviation of the obtained solution values were calculated over the 25 conducted experiments, and for presentation compactness they are all reported in Appendix A.
Finally, the parameter sets were compared using the following criteria, in descending order of importance:

1. Maximum mean value over the 25 experiments
2. Minimum standard deviation over the 25 experiments
3. Minimum average last-hit over the 25 experiments

The necessity for more than one selection criterion derives from the fact that multiple sets of parameters can successfully locate what is deemed to be the function's global maximum within the available budget, although at different cost. Boxplots of the objective values over the 25 experiments for each parameter combination for PSO are presented in Appendix B. Each one is labeled in the following form:

$$v_{max}\_N\_RL\_NT,$$

where v_{max} denotes the limit imposed on velocities, N the swarm size, RL the restarting limit, and NT the neighborhood type. For example label

$$10\_20\_100\_\text{gbest},$$

corresponds to velocity limit (v_{max}) 10% of the search space, swarm size 20, restarting limit 100, and $gbest$ neighborhood type.
Similarly, boxplots for the GA parameter combinations are presented in Appendix C, labeled in the following form:

$$N\_RL\_N_t\_RR\_MR,$$

where N is the population size, RL the restarting limit, N_t is the tournament size, RR the recombination rate and MR the mutation rate.
Based on the results, the most efficient parameter setting for PSO was the following one:

(a) Velocity Limit: 100% of the search space

(b) Swarm Size: 400

(c) Restarting Limit: 100

(d) Neighborhood Type: $gbest$

For the GAs, the most efficient parameter setting was the following:

(1) Population size: 20

(2) N_t: 2

(3) Recombination rate: 0.6

(4) Mutation rate: 0.04

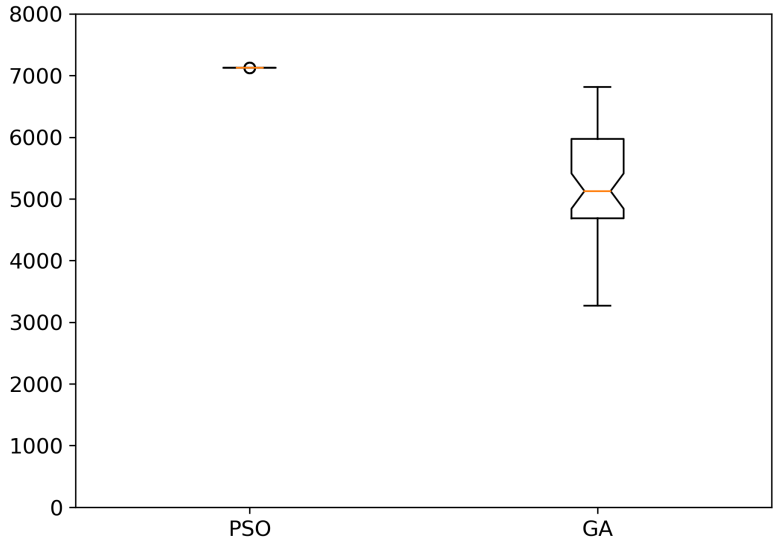
(5) Restarting limit: 100

Between the two solvers, PSO was shown to provide solutions of higher quality, considering the criteria described above. More specifically, Table 4.1 shows that the average of the objective values over 25 experiments for PSO, outfitted with the most efficient parameter setting, was found to be 7126.488, while for the GA the respective

Table 4.1: Statistics for PSO and GA solution quality, setting optimal parameters

Solver	Minimum	Maximum	Average	Median	Std Dev.
PSO	7126.485	7126.489	7126.488	7126.488	0.001
GA	3266.887	6817.273	5259.064	5126.499	845.696

Figure 4.1: Boxplots for PSO and GA solution quality, setting optimal parameters



most efficient parameter setting yielded the average of 5259.064. Figure 4.1 illustrates the objective values in boxplots.

In order to determine if all the parameters can affect the obtained results, the Friedman and Wilcoxon tests were used for comparison among PSO models. For each test, a single parameter of the aforementioned most efficient PSO parameter setting was perturbed. The obtained values over 25 experiments for each parameter setting were compared, in order to detect statistically significant differences among them. The results are presented in Tables 4.2 and 4.3, for Friedman and Wilcoxon tests, respectively. For example, consider the first line of Table 4.2, where the perturbed parameter is velocity limit ($v_{\max}$). Candidate values for the velocity limit are $10\%, 50\%,$ and 100% of the search space. Hence, the Friedman test was performed for the following parameter settings:

(1) $10\_400\_100\_gbest$,

(2) $50\_400\_100\_gbest$,

(3) $100\_400\_100\_gbest$,

Table 4.2: Friedman Tests for PSO

Perturbed Parameter	Candidate Values	p-value
Velocity Limit (%)	10, 50, 100	0.0000
Swarm Size	20, 200, 400	0.0000
Restarting Limit (iterations)	100, 500, 1000	0.0038

Table 4.3: Wilcoxon Test for PSO

Perturbed Parameter	Candidate Values	p-value
Neighborhood Type	lbest, gbest	0.0000

and the resulting p value was found to be 0.0000, which indicates that at least one set of the compared objective values shows statistical difference that is not attributed to random chance.

Since p values were found to approach zero for all the tests presented above, it can be safely assumed that all the examined PSO parameters, namely velocity limit, swarm size, restarting limit, and neighborhood type, could have an impact on the quality of the obtained solutions.

4.3 Solution Profitability Assessment

Through the process described in Chapter 3, it was shown that PSO and GA solvers can be suitably modified for numerical association rule mining, on real-world big data.

In our experimental evaluation, the proposed method was compared to numerical association rule mining without weighted transactions in order to assess its ability to extract more profitable item combinations. Each one of the two cases was evaluated over 25 independent experiments, by utilizing PSO with the most efficient parameter setting. For each one of the 50 experiments, the computational budget was 10^6 function evaluations. The performance metric was the total revenue of the transactions included in the rule solution.

In the case of not utilizing weighted transactions, all 25 experiments converged to

solution (rule):

$$\text{item } 144 \in [0.751, 117.219] \rightarrow \text{item } 133 \in [0, 507.047],$$

which incorporates transactions of 495539.750 total revenue.
However, in the case of utilizing weighted transactions, all 25 experiments converged to more profitable solution (rule):

$$\text{item } 154 \in [0, 191.218] \rightarrow \text{item } 133 \in [0, 205.362],$$

which incorporates transactions of 574736.000 total revenue, 15.98% higher than in the case of no weights in the transactions.
This is a strong indication that using weighted transactions can lead to results that are more interesting from the managerial point of view, even in huge real-world datasets.

The dataset's items were further processed through ABC analysis, which is a method to group the stock into three categories of importance, A, B, and C. There are no prespecified rules per class, and different bounds between the classes can be applied based on user-defined objectives and criteria [7]. An ABC analysis was performed on the dataset, based on the total revenue, according to the following categorization:

(1) A items offer 80% of revenue,

(2) B items offer 15% of revenue,

(3) C items offer 5% of revenue.

According to this categorization, both items of the obtained rule-solution, namely 133, and 154 belong to class "A", which additionally confirms the solution's high quality in respect to profitability.

4.4 Conclusions

The aforementioned experimental findings and the corresponding statistical analysis offer compelling evidence that the proposed optimization-based approach for association rule mining, outfitted with the novel profit-oriented method of weighted transactions, can effectively extract profitable item combinations from a real-world dataset, which can aid the company become more economically competitive. Moreover, the

solutions were compared and shown to be robust, and aligned with results obtained using the well-established operations research method of ABC analysis. Future work will expand our approach to more general frameworks and datasets.

Bibliography

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, no. 2, p. 207–216, jun 1993.

[2] A. E. Alatas Bilal, "Rough particle swarm optimization and its applications in data mining," *Soft Computing*, vol. 12, pp. 1205–1218, 2008.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.

[4] K. E. Parsopoulos and M. N. Vrahatis, *Particle Swarm Optimization and Intelligence: Advances and Applications*. Hershey, PA: Information Science Reference - Imprint of: IGI Publishing, 2010.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[6] A. H. Wright, "Genetic algorithms for real parameter optimization," ser. Foundations of Genetic Algorithms, G. J. RAWLINS, Ed. Elsevier, 1991, vol. 1, pp. 205–220.

[7] K. Lysons, B. Farrington, C. I. of Purchasing, and Supply, *Purchasing and Supply Chain Management*. Financial Times/Prentice Hall, 2005.

Appendix A

Statistics for PSO and GA

The following pages contain the minimum, maximum, average, median, and standard deviation calculated for the solution values obtained over 25 experiments, for each parameter setting of PSO (Tables A.1-A.3) and GA (Tables A.4-A.7).
For PSO, each parameter setting is labeled in the following form:

$$v_{max}\_N\_RL\_NT,$$

where v_{max} denotes the limit imposed on velocities, N the swarm size, RL the restarting limit, and NT the neighborhood type.
For GA, each parameter setting is labeled in the following form:

$$N\_RL\_N_t\_RR\_MR,$$

where N is the population size, RL the restarting limit, N_t is the tournament size, RR the recombination rate and MR the mutation rate.
Dominant values fore each column, are presented in bold.

Table A.1: Statistics for PSO, swarm size 20

Algorithm	Minimum	Maximum	Average	Median	Std Dev.
10_20_100_gbest	5037.307	7125.323	6556.897	6248.088	577.852
10_20_100_lbest	5576.803	7125.014	6487.041	6247.532	450.593
10_20_500_gbest	3136.610	7115.280	5592.232	5725.098	1087.575
10_20_500_lbest	3282.600	7125.318	5821.053	6144.742	1015.486
10_20_1000_gbest	1712.648	6248.090	4360.911	4313.850	1532.262
10_20_1000_lbest	2268.606	7125.168	5710.111	6096.620	1276.518
50_20_100_gbest	**7116.312**	7126.359	**7125.902**	7126.355	**1.999**
50_20_100_lbest	6248.664	7126.358	7089.960	7126.232	175.301
50_20_500_gbest	6245.692	7126.359	6906.696	7116.305	378.402
50_20_500_lbest	6248.183	7126.356	7020.497	7126.236	291.024
50_20_1000_gbest	6099.235	7126.357	6641.175	6248.667	475.056
50_20_1000_lbest	5578.073	7126.351	6744.324	7115.918	486.985
100_20_100_gbest	6248.679	7126.488	7055.813	7126.422	242.918
100_20_100_lbest	7116.111	7126.488	7125.607	**7126.463**	2.859
100_20_500_gbest	6146.388	**7126.489**	6908.817	7116.440	388.884
100_20_500_lbest	6248.388	7126.488	6914.495	7126.303	381.960
100_20_1000_gbest	6146.371	7126.488	6805.255	7126.303	435.735
100_20_1000_lbest	6248.383	7126.488	6879.786	7126.366	401.772

Table A.2: Statistics for PSO, swarm size 200

Algorithm	Minimum	Maximum	Average	Median	Std Dev.
10_200_100_gbest	-0.000	7125.322	5237.918	6248.088	2710.201
10_200_100_lbest	2680.812	7114.491	5325.534	5407.556	952.930
10_200_500_gbest	340.480	7125.322	5050.406	6132.305	1994.875
10_200_500_lbest	1015.698	6704.773	4426.010	5575.238	1783.186
10_200_1000_gbest	268.338	7125.322	4320.247	5582.311	2463.093
10_200_1000_lbest	2.050	7125.159	3418.454	3092.456	2272.498
50_200_100_gbest	7126.236	7126.359	7126.349	7126.359	0.034
50_200_100_lbest	6245.586	7126.352	6791.045	7126.103	425.307
50_200_500_gbest	6248.667	7126.360	7056.104	7126.359	243.009
50_200_500_lbest	4359.403	7126.350	6304.744	6248.305	757.287
50_200_1000_gbest	4445.113	7126.359	6878.625	7126.237	603.351
50_200_1000_lbest	1321.406	7126.346	5943.154	6248.645	1637.676
100_200_100_gbest	**7126.370**	**7126.489**	**7126.478**	**7126.488**	**0.029**
100_200_100_lbest	6245.743	7126.485	7013.406	7126.368	289.432
100_200_500_gbest	**7126.370**	**7126.489**	7126.442	**7126.488**	0.055
100_200_500_lbest	4558.580	7126.488	6749.387	7126.367	776.412
100_200_1000_gbest	6248.740	**7126.489**	6985.215	7126.422	328.062
100_200_1000_lbest	3786.789	7126.485	6072.893	6248.383	1030.715

Table A.3: Statistics for PSO, swarm size 400

Algorithm	Minimum	Maximum	Average	Median	Std Dev.
10_400_100_gbest	6245.246	7125.326	6843.522	7125.168	417.394
10_400_100_lbest	3280.654	7021.866	5472.868	6096.261	1190.926
10_400_500_gbest	285.605	7125.324	4633.207	5582.319	2383.345
10_400_500_lbest	340.427	7024.310	3597.080	3388.579	1744.009
10_400_1000_gbest	231.093	7125.322	4324.223	5051.998	2529.852
10_400_1000_lbest	1.806	7125.249	2805.342	1960.440	2510.109
50_400_100_gbest	7126.236	7126.360	7126.349	7126.359	0.034
50_400_100_lbest	5047.093	7126.350	6858.029	7126.103	513.681
50_400_500_gbest	6248.675	7126.360	7021.027	7126.359	291.091
50_400_500_lbest	1182.218	7126.353	5182.133	5269.893	1599.255
50_400_1000_gbest	309.766	7126.360	6643.040	7126.359	1373.075
50_400_1000_lbest	13.213	7126.351	5216.913	5770.583	2049.099
100_400_100_gbest	**7126.485**	**7126.489**	**7126.488**	**7126.488**	**0.001**
100_400_100_lbest	5773.284	7126.484	6823.721	7126.303	454.660
100_400_500_gbest	7126.370	**7126.489**	7126.474	**7126.488**	0.039
100_400_500_lbest	3581.093	7126.422	6310.954	6248.642	948.531
100_400_1000_gbest	3208.865	7126.488	6711.540	**7126.488**	892.732
100_400_1000_lbest	3352.730	7126.484	5835.818	6248.440	1088.186

Table A.4: Statistics for GA, population size 200 (a)

Algorithm	Minimum	Maximum	Average	Median	Std Dev.
200_100_100.0_0.6_0.01	3621.955	6248.519	4917.894	4917.793	797.974
200_100_100.0_0.6_0.02	2928.788	6086.778	4482.497	4322.263	802.083
200_100_100.0_0.6_0.03	2630.944	7023.297	4806.800	4761.761	993.009
200_100_100.0_0.6_0.04	3393.298	6593.076	4864.619	4535.096	955.301
200_100_100.0_0.9_0.01	3002.298	6245.751	4156.009	4152.343	761.527
200_100_100.0_0.9_0.02	2886.444	6883.329	4221.776	4236.877	838.024
200_100_100.0_0.9_0.03	3122.244	6248.629	4131.776	4173.803	734.955
200_100_100.0_0.9_0.04	2545.162	6547.853	4153.626	3915.671	929.794
200_100_20.0_0.6_0.01	3229.563	6248.557	4755.047	4682.729	927.679
200_100_20.0_0.6_0.02	3532.235	6248.494	4537.548	4429.746	834.462
200_100_20.0_0.6_0.03	**3705.724**	6443.214	**5082.711**	**5100.427**	804.190
200_100_20.0_0.6_0.04	3074.717	6883.316	4940.388	5092.024	1207.815
200_100_20.0_0.9_0.01	2986.699	6380.158	4207.450	4198.822	796.787
200_100_20.0_0.9_0.02	2614.431	5577.828	4001.349	3932.378	**611.082**
200_100_20.0_0.9_0.03	2621.634	6248.592	4158.012	4338.138	738.542
200_100_20.0_0.9_0.04	3103.165	**7036.004**	3965.492	3781.059	880.109

Table A.5: Statistics for GA, population size 200 (b)

Algorithm	Minimum	Maximum	Average	Median	Std Dev.
200_300_100.0_0.6_0.01	2657.800	**6585.668**	4235.354	3612.178	1219.847
200_300_100.0_0.6_0.02	2535.903	6245.731	3989.541	3809.940	1010.842
200_300_100.0_0.6_0.03	2595.321	6097.424	4213.748	3948.591	1068.842
200_300_100.0_0.6_0.04	2169.505	6248.697	3699.852	3688.358	1112.558
200_300_100.0_0.9_0.01	2450.562	6248.444	3638.618	3506.953	849.513
200_300_100.0_0.9_0.02	1986.509	6275.677	3253.627	2812.466	1174.250
200_300_100.0_0.9_0.03	1876.788	4888.670	3289.228	3429.913	720.094
200_300_100.0_0.9_0.04	1978.429	5834.727	3743.302	3637.616	918.200
200_300_20.0_0.6_0.01	2536.539	6394.704	4008.347	3779.562	998.596
200_300_20.0_0.6_0.02	**2934.198**	6246.933	4183.226	3938.560	892.092
200_300_20.0_0.6_0.03	1915.508	6453.896	3556.588	3497.353	1085.761
200_300_20.0_0.6_0.04	2435.234	6468.349	**4403.993**	**4113.407**	1264.631
200_300_20.0_0.9_0.01	1450.889	5770.680	3581.286	3510.707	1031.405
200_300_20.0_0.9_0.02	1868.792	4429.561	3567.229	3696.145	**701.714**
200_300_20.0_0.9_0.03	1816.598	6245.757	3501.050	3364.362	1069.577
200_300_20.0_0.9_0.04	2131.235	4798.282	3346.302	3359.447	721.598

Table A.6: Statistics for GA, population size 20 (a)

Algorithm	Minimum	Maximum	Average	Median	Std Dev.
20_100_10.0_0.6_0.01	2870.147	4714.019	3999.715	3943.643	527.481
20_100_10.0_0.6_0.02	3094.579	4953.131	3929.509	3896.028	504.627
20_100_10.0_0.6_0.03	2749.081	4728.539	4009.868	4215.555	575.901
20_100_10.0_0.6_0.04	3035.769	4958.150	4111.375	4181.183	464.352
20_100_10.0_0.9_0.01	2753.447	4777.631	4081.093	4285.192	602.557
20_100_10.0_0.9_0.02	2740.872	5080.116	3878.510	3822.302	558.955
20_100_10.0_0.9_0.03	2693.944	5131.071	3865.457	3930.757	641.328
20_100_10.0_0.9_0.04	3166.069	4720.538	3984.379	3973.337	**442.033**
20_100_2.0_0.6_0.01	3260.795	6883.378	5144.616	**5212.992**	987.997
20_100_2.0_0.6_0.02	2999.685	7025.758	5115.896	5132.411	1003.298
20_100_2.0_0.6_0.03	2976.725	7036.031	5084.202	5097.699	1078.006
20_100_2.0_0.6_0.04	**3266.887**	6817.273	**5259.064**	5126.499	845.696
20_100_2.0_0.9_0.01	3004.779	6086.782	4252.191	4270.096	822.824
20_100_2.0_0.9_0.02	2969.691	6672.118	4349.800	4381.869	818.939
20_100_2.0_0.9_0.03	2612.516	6661.419	4108.147	3913.064	773.055
20_100_2.0_0.9_0.04	3245.571	**7126.445**	4500.016	4248.384	1006.687

Table A.7: Statistics for GA, population size 20 (b)

Algorithm	Minimum	Maximum	Average	Median	Std Dev.
20_300_10.0_0.6_0.01	2243.670	4508.809	3478.907	3552.589	672.223
20_300_10.0_0.6_0.02	1834.217	4280.087	3298.084	3375.234	674.076
20_300_10.0_0.6_0.03	1724.753	4478.125	3135.266	3084.563	809.608
20_300_10.0_0.6_0.04	2336.007	4323.456	3377.682	3357.480	538.891
20_300_10.0_0.9_0.01	2274.379	4646.893	3337.583	3438.632	671.561
20_300_10.0_0.9_0.02	2514.589	4438.024	3289.852	3221.731	**528.060**
20_300_10.0_0.9_0.03	1996.193	5101.667	3692.895	3746.250	763.979
20_300_10.0_0.9_0.04	1592.483	4648.966	3089.188	3061.362	782.243
20_300_2.0_0.6_0.01	1898.938	6547.860	4131.962	4144.852	1130.473
20_300_2.0_0.6_0.02	1992.700	6883.311	**4228.676**	**4251.548**	1325.921
20_300_2.0_0.6_0.03	**2719.144**	7023.260	4168.837	4186.604	1005.016
20_300_2.0_0.6_0.04	2154.028	**7025.816**	3848.858	3455.608	1092.819
20_300_2.0_0.9_0.01	1336.940	6828.028	3535.281	3362.869	1035.542
20_300_2.0_0.9_0.02	2022.382	6248.699	3495.209	3373.948	1028.138
20_300_2.0_0.9_0.03	1793.497	6853.859	3894.181	3695.269	1130.465
20_300_2.0_0.9_0.04	2068.851	5101.668	3650.026	3610.784	823.998

Appendix B

Boxplots for PSO Solution Quality

The following pages contain boxplots for PSO solution quality, obtained over 25 experiments for each parameter setting. Each boxplot is labeled in the following form:

$$v_{max}\_N\_RL\_NT,$$

where v_{max} denotes the limit imposed on velocities, N the swarm size, RL the restarting limit, and NT the neighborhood type.

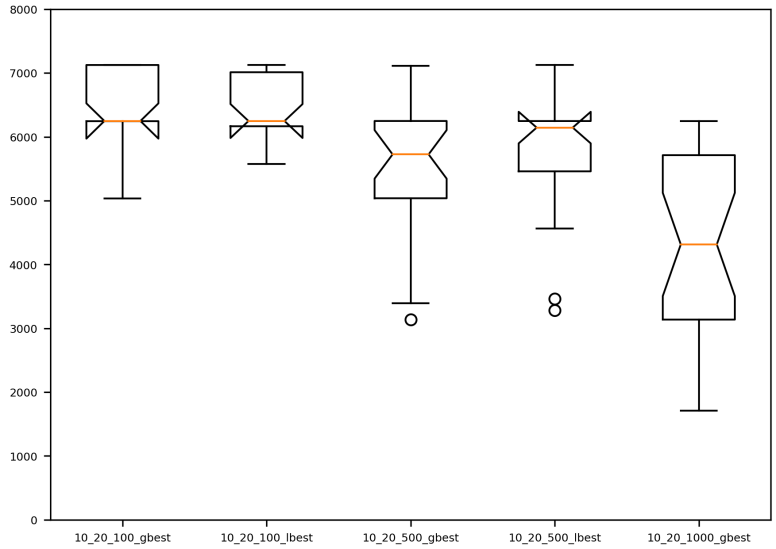


Figure B.1: Boxplots for PSO solution quality, settings 1-5

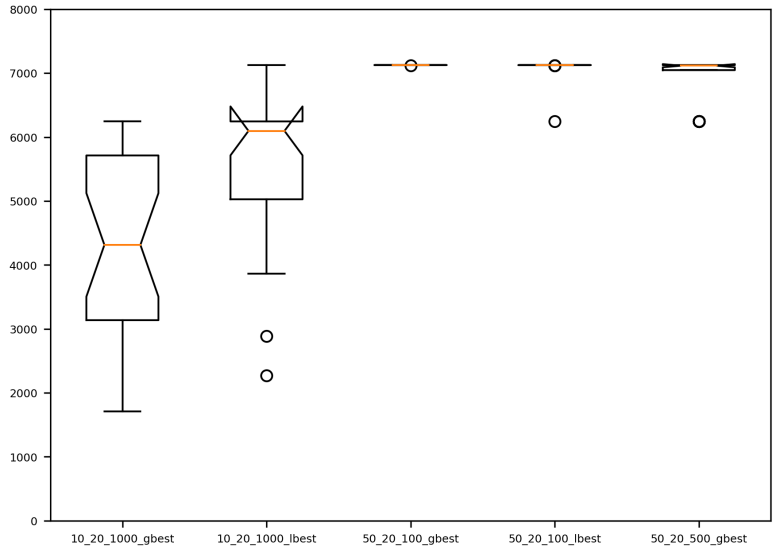


Figure B.2: Boxplots for PSO solution quality, settings 5-9

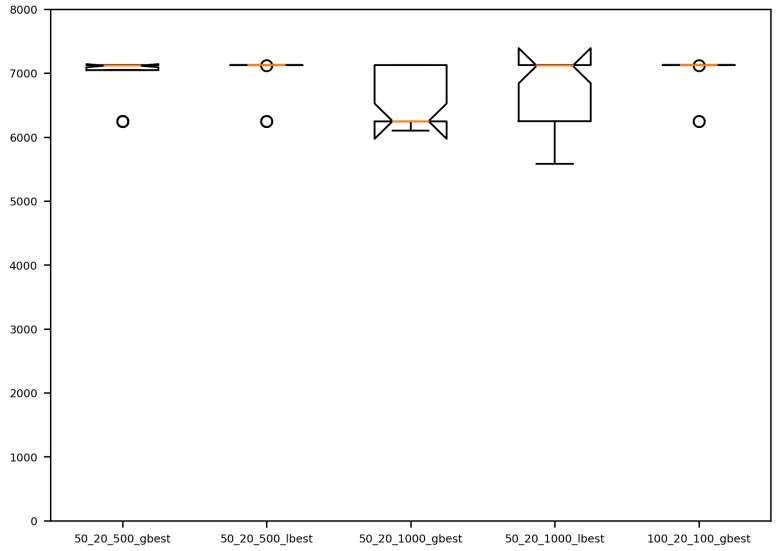


Figure B.3: Boxplots for PSO solution quality, settings 9-13

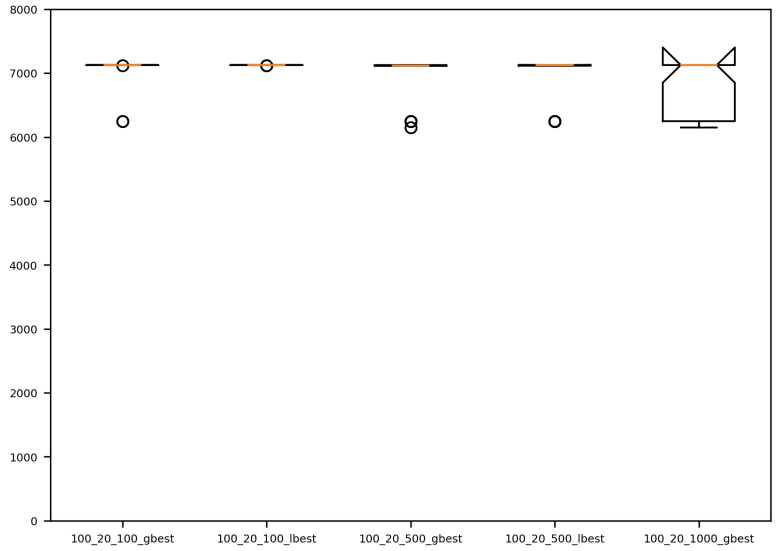


Figure B.4: Boxplots for PSO solution quality, settings 13-17

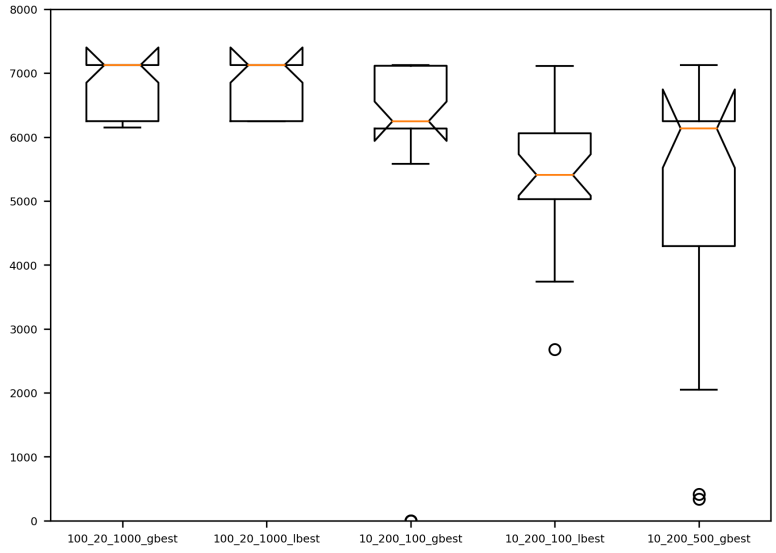


Figure B.5: Boxplots for PSO solution quality, settings 17-21

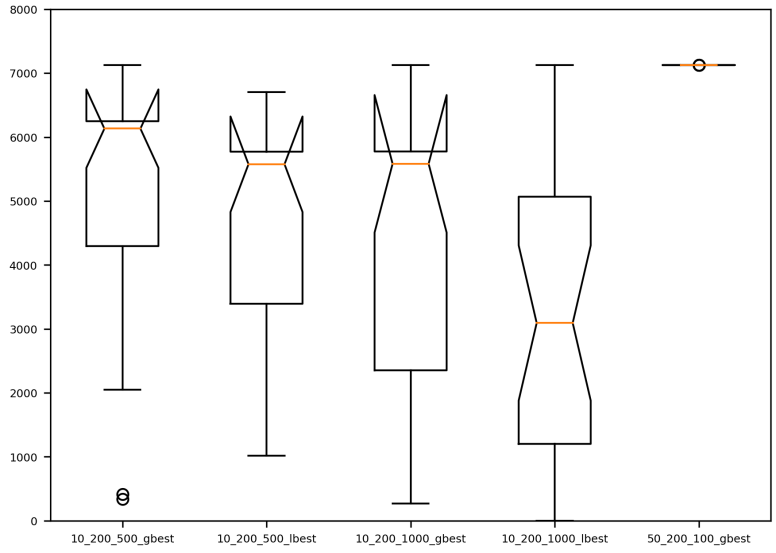


Figure B.6: Boxplots for PSO solution quality, settings 21-25

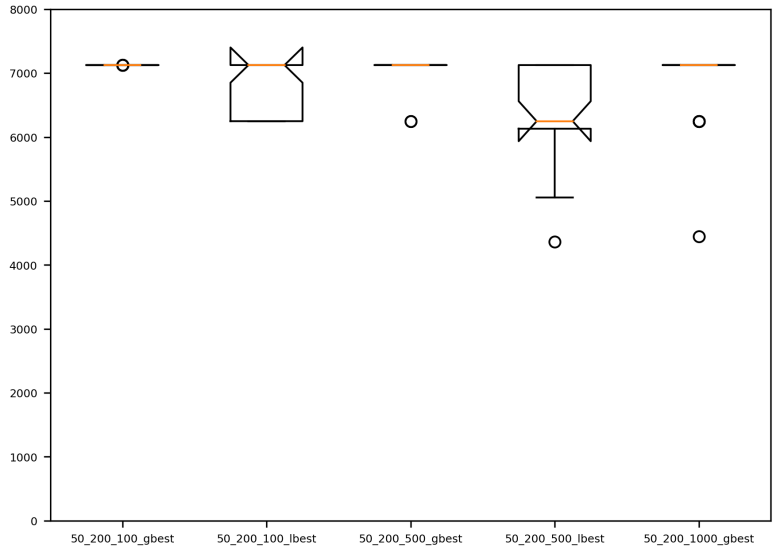


Figure B.7: Boxplots for PSO solution quality, settings 25-29

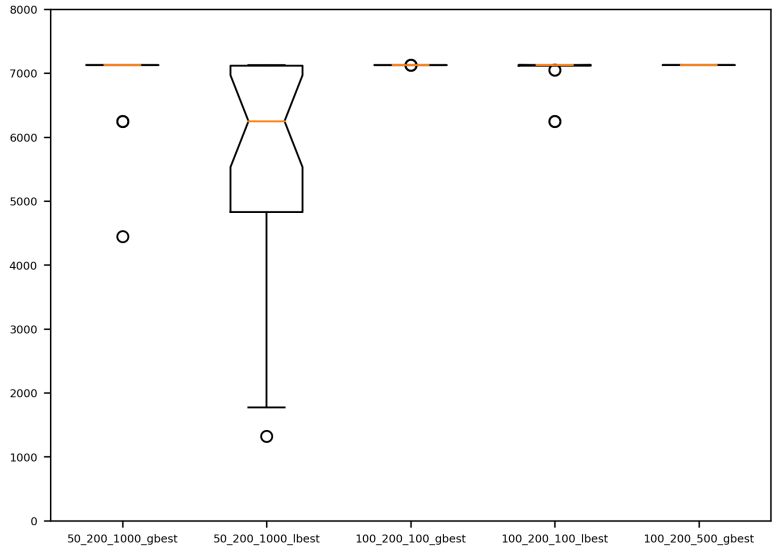


Figure B.8: Boxplots for PSO solution quality, settings 29-33

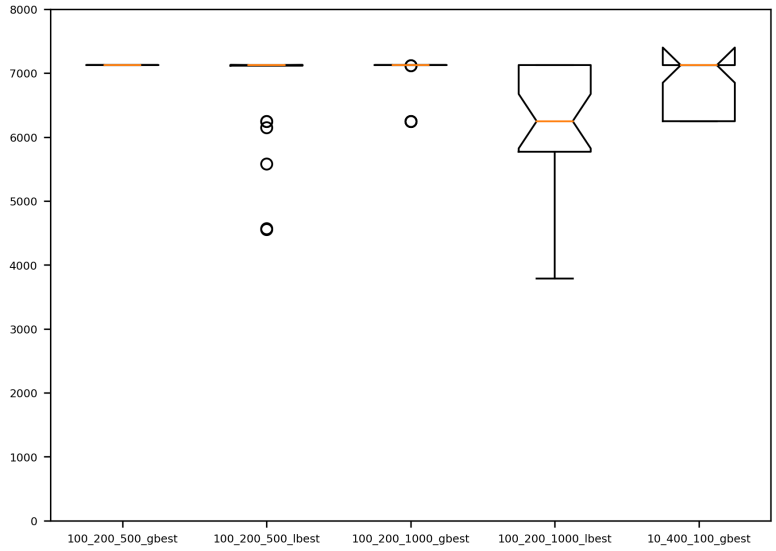


Figure B.9: Boxplots for PSO solution quality, settings 33-37

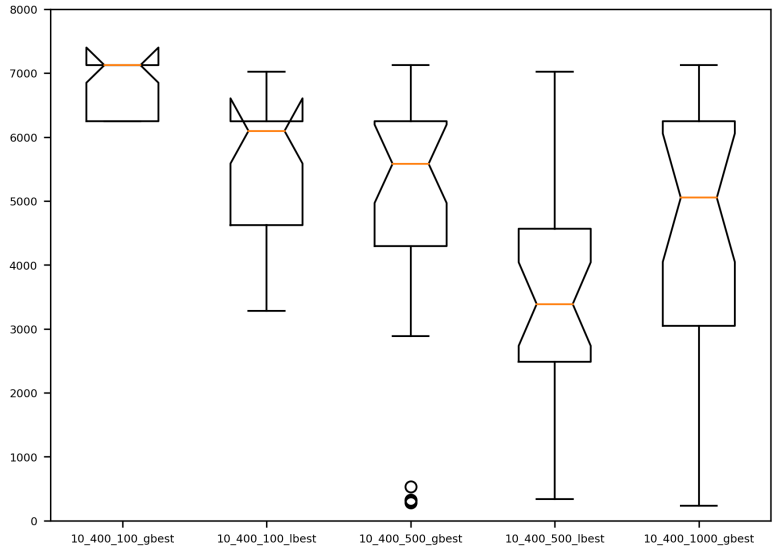


Figure B.10: Boxplots for PSO solution quality, settings 37-41

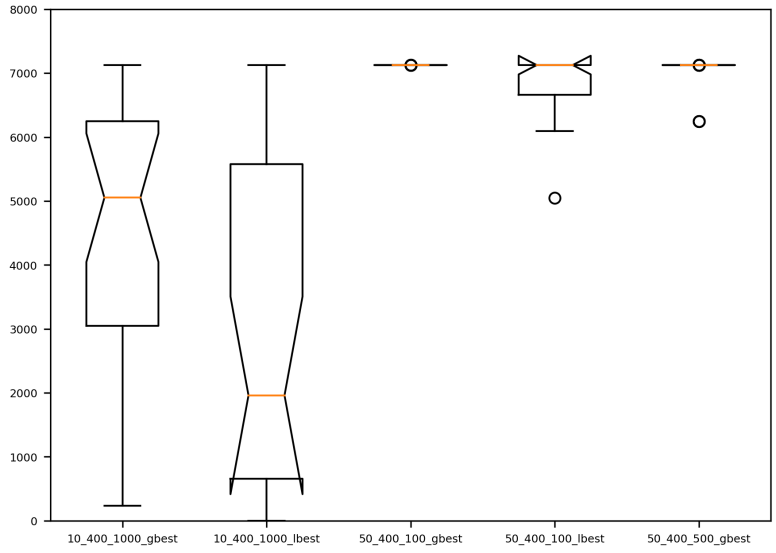


Figure B.11: Boxplots for PSO solution quality, settings 41-45

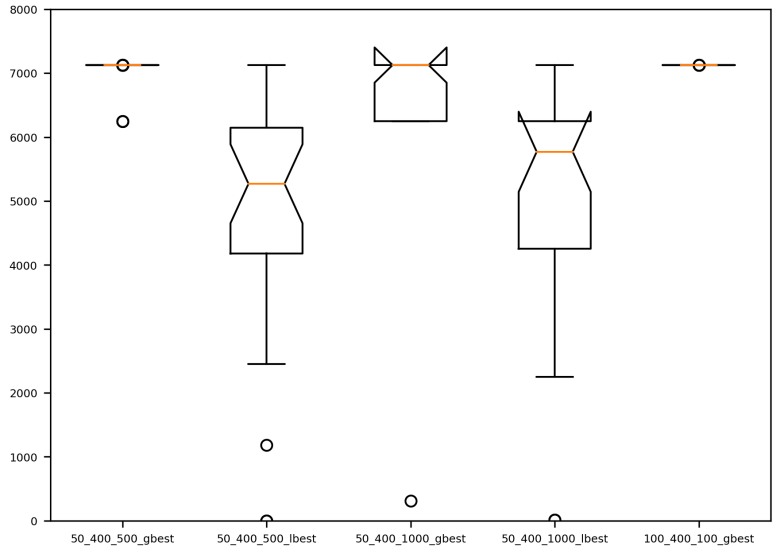


Figure B.12: Boxplots for PSO solution quality, settings 45-49

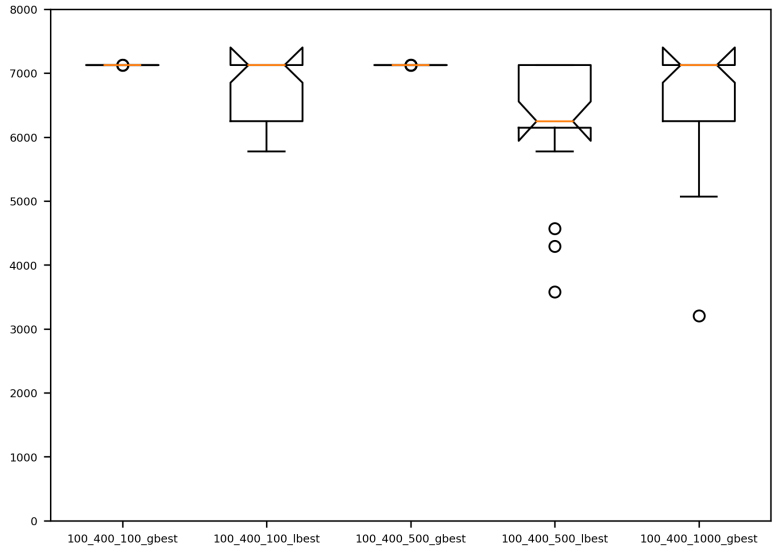


Figure B.13: Boxplots for PSO solution quality, settings 49-53

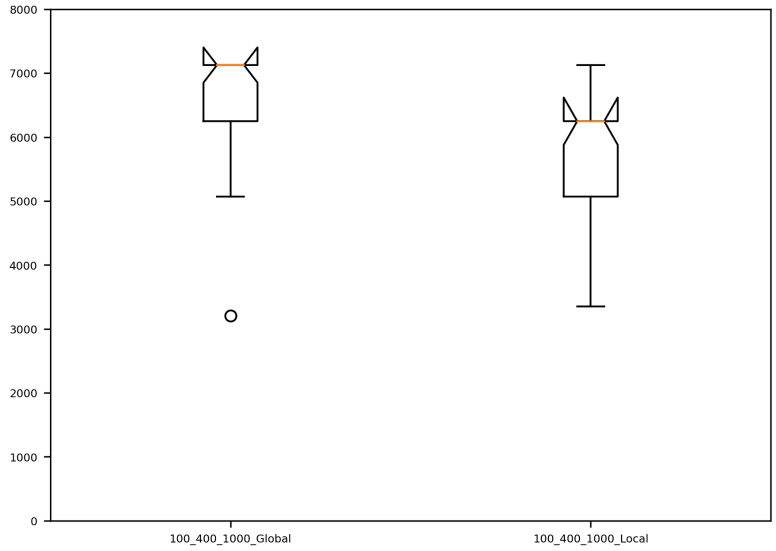


Figure B.14: Boxplots for PSO solution quality, settings 53-54

Appendix C

Boxplots for GA Solution Quality

The following pages contain boxplots for GA solution quality, obtained over 25 experiments for each parameter setting. Each boxplot is labeled in the following form:

$$N\_RL\_N_t\_RR\_MR,$$

where N is the population size, RL the restarting limit, N_t is the tournament size, RR the recombination rate and MR the mutation rate.

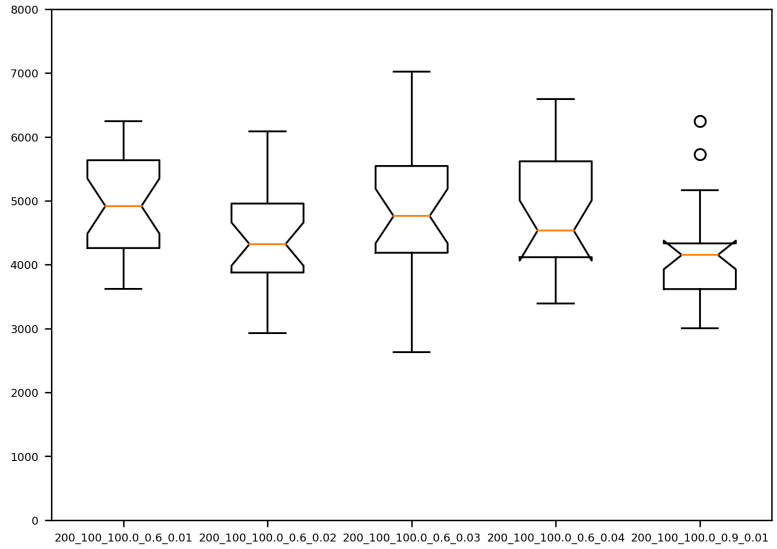


Figure C.1: Boxplots for GA solution quality, settings 1-5

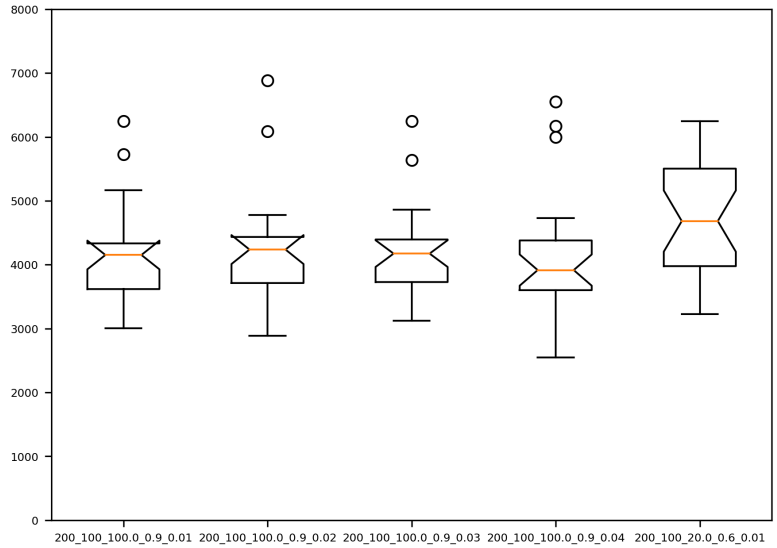


Figure C.2: Boxplots for GA solution quality, settings 5-9

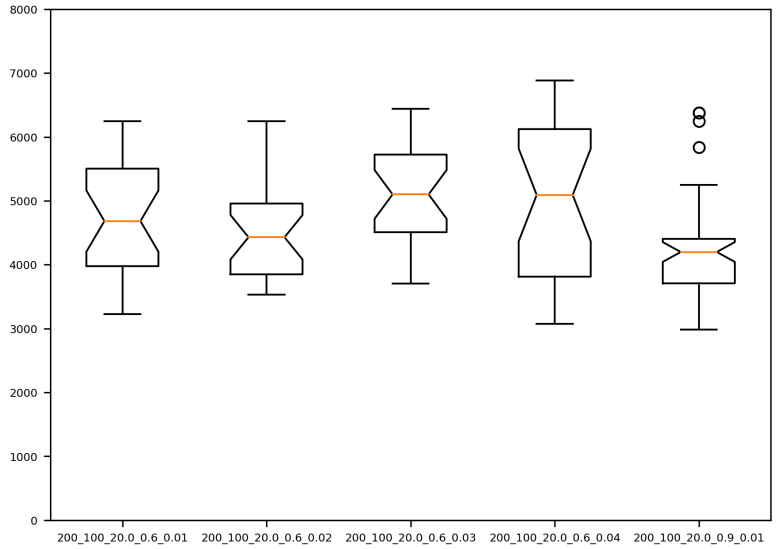


Figure C.3: Boxplots for GA solution quality, settings 9-13

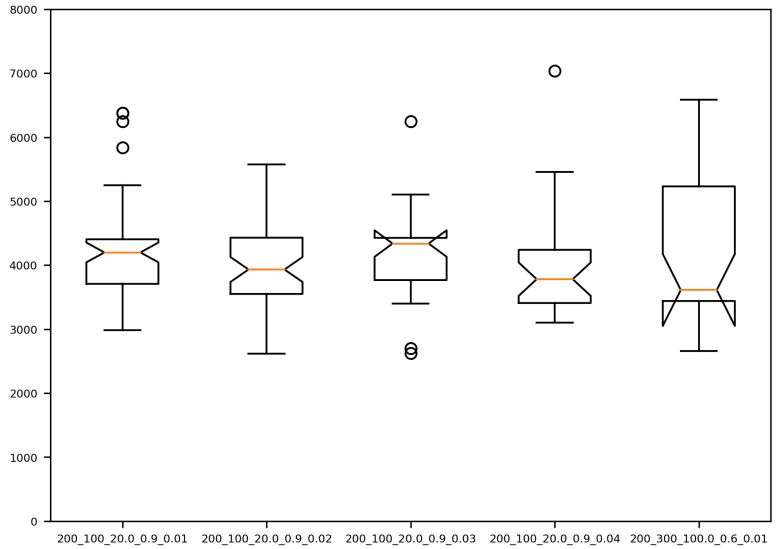


Figure C.4: Boxplots for GA solution quality, settings 13-17

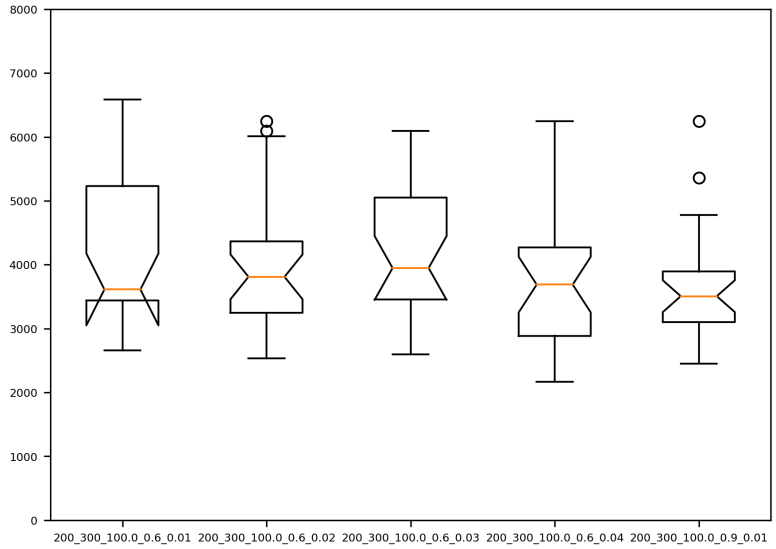


Figure C.5: Boxplots for GA solution quality, settings 17-21

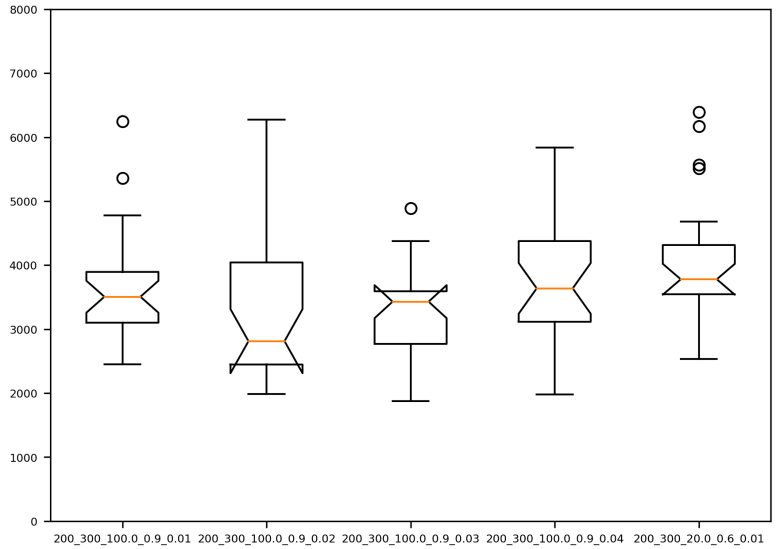


Figure C.6: Boxplots for GA solution quality, settings 21-25

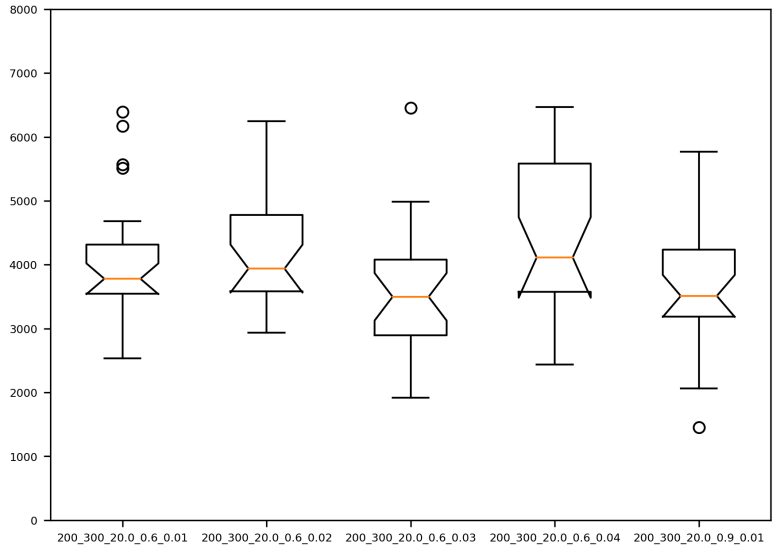


Figure C.7: Boxplots for GA solution quality, settings 25-29

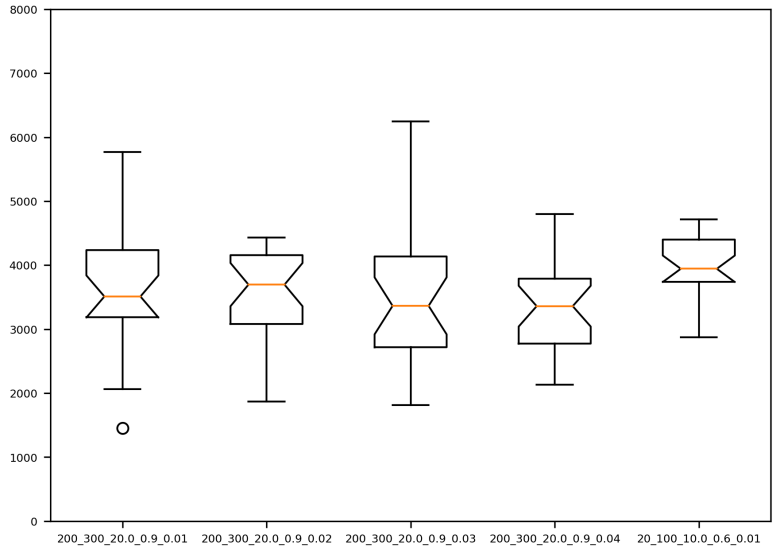


Figure C.8: Boxplots for GA solution quality, settings 29-33

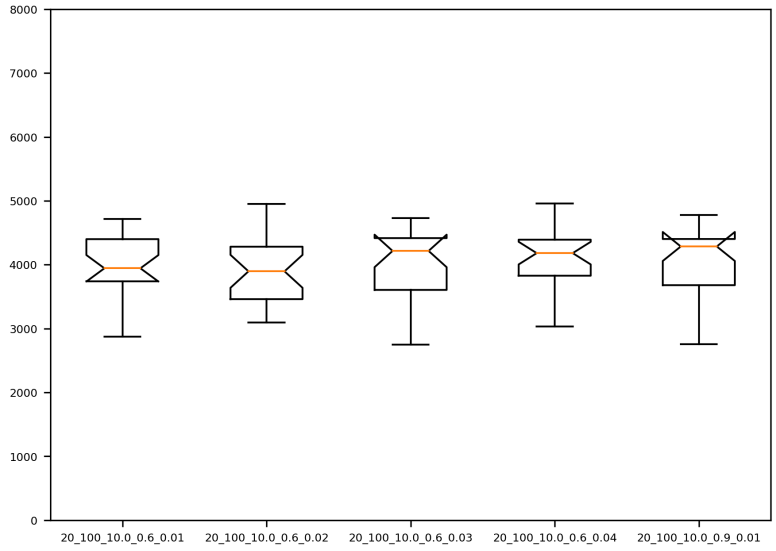


Figure C.9: Boxplots for GA solution quality, settings 33-37

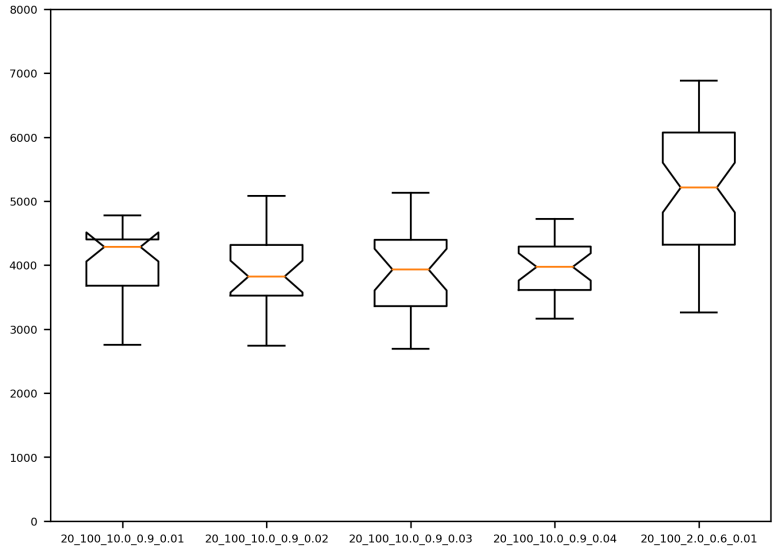


Figure C.10: Boxplots for GA solution quality, settings 37-41

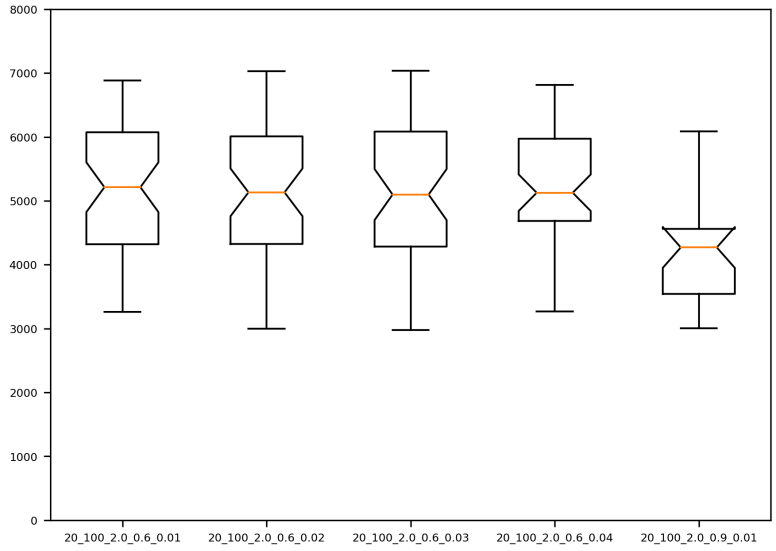


Figure C.11: Boxplots for GA solution quality, settings 41-45

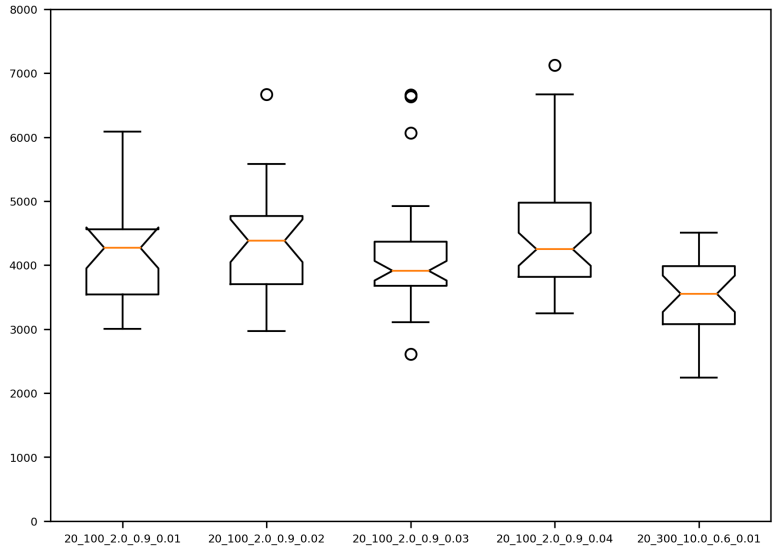


Figure C.12: Boxplots for GA solution quality, settings 45-49

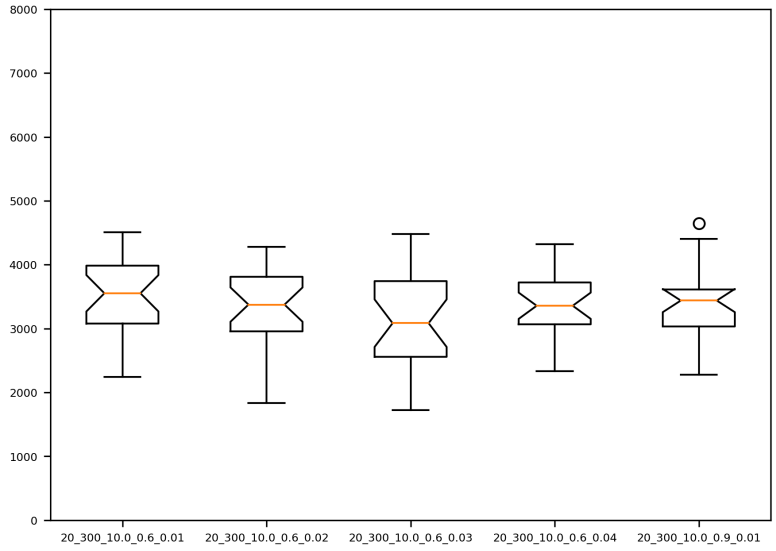


Figure C.13: Boxplots for GA solution quality, settings 49-53

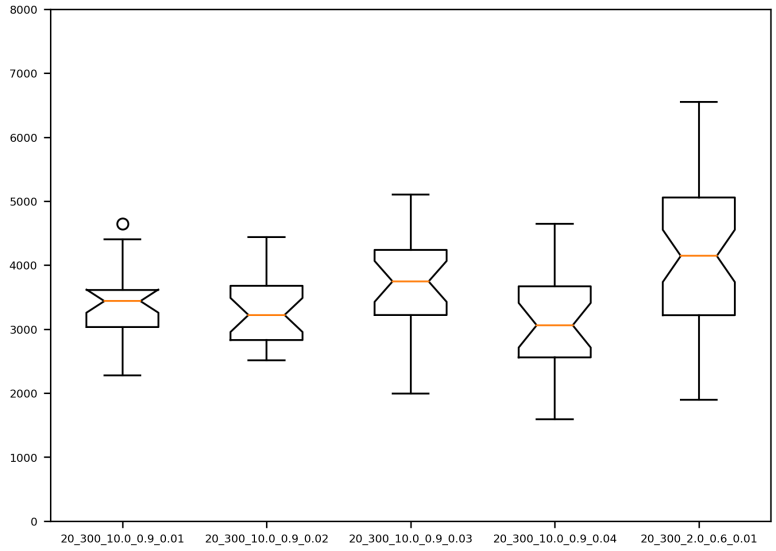


Figure C.14: Boxplots for GA solution quality, settings 53-57

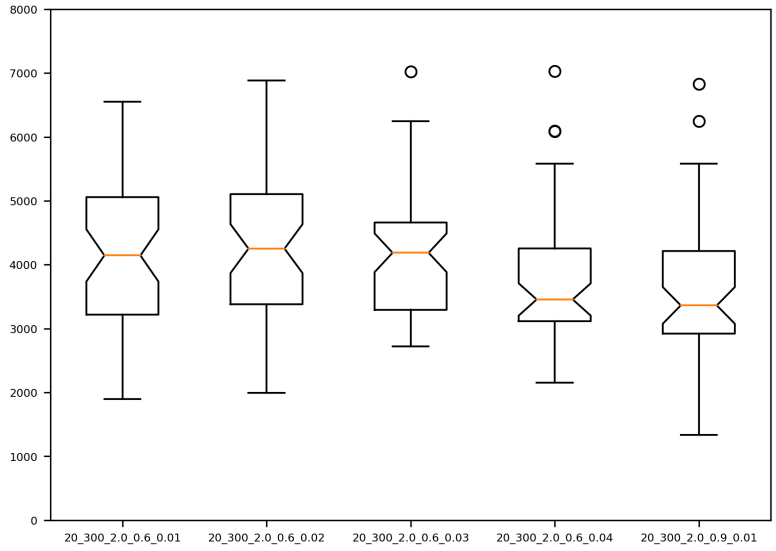


Figure C.15: Boxplots for GA solution quality, settings 57-61

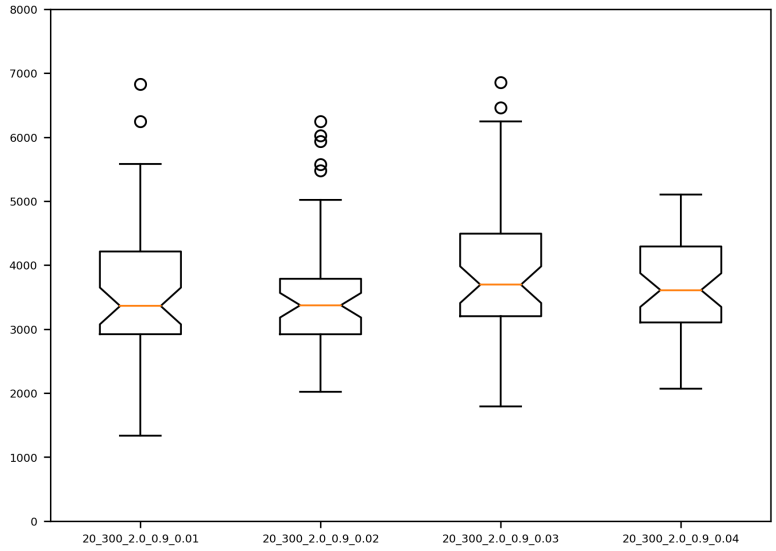


Figure C.16: Boxplots for GA solution quality, settings 61-64

Short Biography

Adam Kypriadis was born in Thessaloniki, Greece, in 1994. In 2012 he enrolled in the undergraduate program of Mathematics of the University of Ioannina and earned his Degree in 2020. In 2021 he enrolled in the Graduate Program of the Department of Computer Science and Engineering of University of Ioannina, and is persuing a Master's Degree in "Data and Computer Systems Engineering".