# Prüfungsvorleistung Codierungs-Theorie

David Tarnow, Michael Narkus, Armel Siewe, Lukas Köhler

28. Juni 2018

# Inhaltsverzeichnis

# 1 Einführung

Einführung in die Thematik. Aufbau der Applikation, Imports (Gurobi, armadillo und co)

## 1.1 Verwendung von C++

Als Programmiersprache wurde in diesem Projekt C++ verwendet. Die entscheidenden Gründe für C++ waren zum einen die hohe Performance sowie die existierenden Bibliotheken (z.B. Armadillo) und Schnittstellen für die Nutzung von Gurobi.

## 1.2 Armadillo

Armadillo ist eine C++-Bibliothek, welche Funktionen und Klassen zur linearen Algebra bereitstellt. Angelehnt an MatLab werden hier Klassen für Vektoren, Matrizen und Kuben bereitgestellt. Im Rahmen dieses Projektes wurde Armadillo in der neusten stabilen Version (8.500.1) verwendet.

## 1.3 Gurobi

Mit dem Gurobi Solver wurde im Rahmen dieser Praktikumsaufgabe eine externe Software genutzt um die Optimierungsprobleme zu lösen. Eine Schnittstelle zur Ansteuerung des Gurobi-Solvers in der Version 8.5.0 wird hier vom Hersteller bereitgestellt.

## 1.4 LAPACK

Als Abhängigkeit von Armadillo wurde außerdem LAPACK genutzt. Um die komplexeren Berechnungen, welche mit Armadillo durchgeführt werden können, zu beschleunigen, wird unter Windows OpenBLAS (oder Intel MK) benötigt. Die Bibliothek LAPACK stellt hierfür ebenfalls Komponenten bereit.

# 2 Schritt 1

Ziel von Schritt 1 war es, anhand der Werte für k und q, eine Matrix der Form $A_{k,q}$ zu konstruieren.
Die Anwendung nimmt zunächst die Werte entgegen und berechnet eine Sammlung von linear unabhängigen Vektoren. Die Berechnung erfolgt hierbei indem die Vektoren der Reihe nach konstruiert werden und rekursiv auf ihre lineare Unabhängigkeit geprüft werden.
Sind ausreichend, linear voneinander unabhängigen, Vektoren gefunden, wird anhand der Werte von k und q die Größe der Matrix berechnet. Die Berechnung basiert auf der, in der Vorlesung vorgestellten, Formel: $\frac{q^k - 1}{q - 1}$.
Anhand der berechneten Größe der Matrix, wird zum Abschluss eine Matrix aus den zuvor erstellten Vektoren generiert und ausgegeben.

Getestet wurde dieser Teil der Anwendung beispielhaft für das folgenden Wertepaar:

q=3 und k=3:
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Neben den oben aufgeführten Werten wurden noch deutlich größere Eingaben, z.B. .. getestet. Die Matrizen werden hierzu korrekt berechnen, lassen sich allerdings nicht mehr korrekt innerhalb eines PDF's darstellen.

# 3 Schritt 2

Im Rahmen des zweiten Schrittes kam zum ersten Mal Gurobi zum Einsatz. Ziel der Aufgabe war es hierbei, das Optimierungsproblem für (Formel hier einfügen), über eine Schnittstelle zum Gurobi-Solver, zu lösen.
Die Lösung des Optimierungsproblems erfolgt über eine Schnittstelle zum eingebundenen Gurobi-Solvers. Notwendig für diese Optimierung ist jedoch zunächst das Erstellen eines Models. Elementar ist hierbei, dass dem Model Grenzen übermittelt werden. Diese Grenzen ergeben sich in diesem Ansatz aus der eingangs, siehe Schritt 1, berechneten Matrix $A_{k,q}$. Sind diese Constraints ermittelt, wird im Anschluss der Optimierungsprozess angestoßen.
Ist die Optimierung erfolgreich, wird im Anschluss daran die optimierte Matrix ausgegeben.

```
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [3e+00, 3e+00]
Found heuristic solution: objective 3.0000000
Presolve time: 0.00s
Presolved: 13 rows, 13 columns, 52 nonzeros
Variable types: 0 continuous, 13 integer (0 binary)

Root relaxation: objective 9.750000e+00, 11 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    9.75000    0   13    3.00000    9.75000   225%     -    0s
H    0     0                       9.0000000    9.75000  8.33%     -    0s

Explored 1 nodes (11 simplex iterations) in 0.00 seconds
Thread count was 4 (of 4 available processors)

Solution count 2: 9 3

Optimal solution found (tolerance 1.00e-04)
Best objective 9.000000000000e+00, best bound 9.000000000000e+00, gap 0.0000%
x0 0
x1 0
x2 0
x3 0
x4 1
x5 1
x6 1
x7 1
x8 1
x9 1
x10 1
x11 1
x12 1
n=9
```

Abbildung 1: Optimierung für q,k,b=3

# 4 Schritt 3

Schritt 3 erweitert das bestehende Optimierungsproblem um ein wachsendes $b \geq 1$. Ist dieses Problem ausreichend optimiert für die Matrix A gelöst, wird für ein bestimmtes x die Generatormatrix des zugehörigen Codes C berechnet. Zum Abschluss wird nach der Generierung die Hamming-Distanz der errechneten, optimierten, Generatormatrix geprüft.

```
--------------------------------------------------

We have now following code
--------------------------------------------------
[n,k,d]_q
[9,3,6]_3
--------------------------------------------------

Generate Generatormatrix G
--------------------------------------------------
   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000
        0        0        0   1.0000   1.0000   1.0000   2.0000   2.0000   2.0000
        0   1.0000   2.0000        0   1.0000   2.0000        0   1.0000   2.0000
--------------------------------------------------

Proof Hamming Distance
--------------------------------------------------
6 = calculated distance
6 = should be the distance
--------------------------------------------------
```

Abbildung 2: Ausgabe für q, k, b = 3 aus Schritt 3

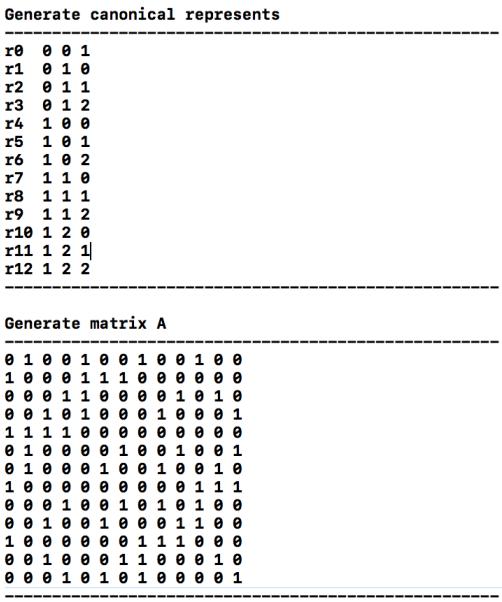


Abbildung 3: Berechnung der Matrix A über Repräsentanten

# 5 Schritt 4

Die nächste Erweiterung der Anwendung bildet die Konstruktion der $A_{k,q}^{S}$ für gegebene Primzahlen (k, q) und eine Gruppe S.

Die in Schritt 3 berechnete Erzeugermatrix wird nun weiterverwendet und auf die einzelnen Repräsentanten angewendet. Getestet wird hierbei, ob das Ergebnis in der gleichen Gruppe liegt. Dieser Fall trifft zu, wenn das Ergebnis den Repräsentanten selbst oder ein Vielfaches darstellt.
Trifft dies nicht zu, ist das Ergebnis ein Element einer anderen Gruppe. Sind alle Gruppen gefunden, startet die Suche nach Zyklen. Getestet wird hierbei, ob diese verschiedenen Gruppen in Kombination mit der Erzeugermatrix verkettet werden können. Ein Beispiel für einen solchen Zyklus wäre, wenn fortlaufend aus einem Element aus r0 ein Element der Gruppe r2 erzeugt wird, aus r2 eines der Gruppe r8 und aus r8 wiederum ein Element von r0.
Ist ein solcher Zyklus gefunden, werden alle entsprechenden Spalten addiert und ergeben einen gemeinsamen Vektor C. Dieser enthält somit alle Elemente innerhalb einer Gruppe.

Folgende Ausgaben ergeben sich beim Test für die Werte q,k,b = 3:

# 6 Schritt 5

Zum Abschluss sollen in Schritt 5 zufällige zyklische Gruppen (mit Erzeuger) Man versucht einen Erzeuger zu finden, der ein möglichst hohes n erzeugt wobei das ganze noch lösbar sein muss. Wenn man die Reduktion von A nicht machen würde, dann dauert das Rechnen sehr lange

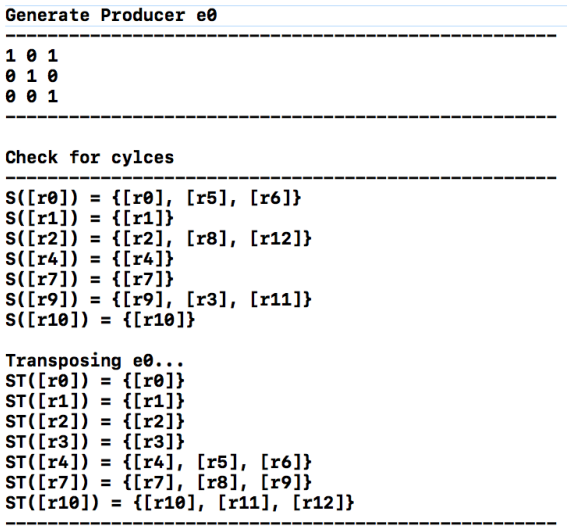


Abbildung 4: Erzeugermatrix e0 und die Suche nach Zyklen

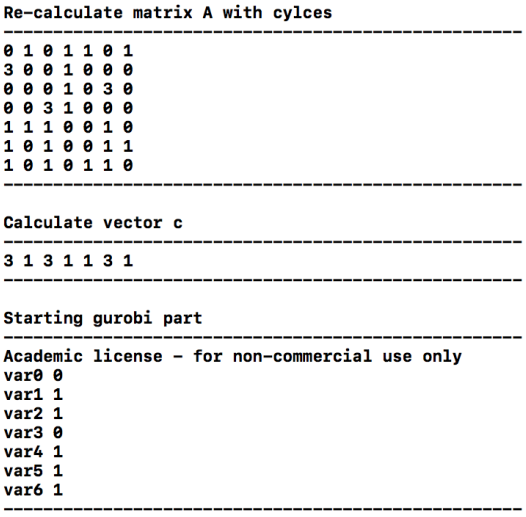


Abbildung 5: Erneute Berechnung von A und Gurobi

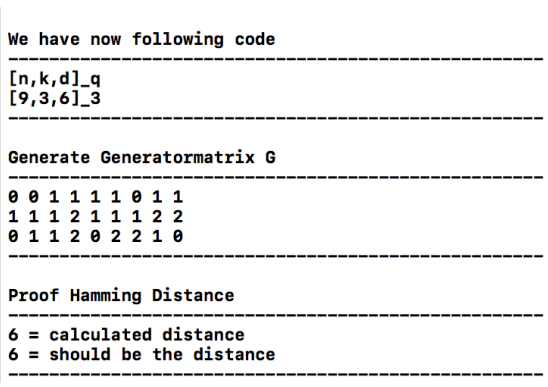


Abbildung 6: Code C, Generatormatrix G und Hamming-Test