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 Konstruktion der Matrix A

Zunächst werden die linear unabhängigen Vektoren für gegebene Werte q und k (nur Primzahlen) berechnet. Daraus ergeben sich insgesamt q^k -1 Vektoren, die in $\frac{q^k - 1}{q - 1}$ Klassen eingeteilt werden können. Dabei besitzt jede Klasse genau q - 1 Vielfache. Die Berechnung der linear unabhängigen Vektoren erfolgt im gegebenen Programm rekursiv.

Es wird im folgenden der Fall für q = 3 und k = 3 betrachtet. Dabei ergeben sich folgende linear unabhängigen Vektoren:

r0 = $(0\ 0\ 1)^T$

r1 = $(0\ 1\ 0)^T$

r2 = $(0\ 1\ 1)^T$

r3 = $(0\ 1\ 2)^T$

r4 = $(1\ 0\ 0)^T$

r5 = $(1\ 0\ 1)^T$

r6 = $(1\ 0\ 2)^T$

r7 = $(1\ 1\ 0)^T$

r8 = $(1\ 1\ 1)^T$

r9 = $(1\ 1\ 2)^T$

r10 = $(1\ 2\ 0)^T$

r11 = $(1\ 2\ 1)^T$

r12 = $(1\ 2\ 2)^T$

Im Anschluss wird die Matrix A der Form $A_{k,q}$ konstruiert. Dies geschieht innerhalb von geschachtelten for- Schleifen, die jeweils das Skalarprodukt von zwei Repräsentanten berechnet. Das Ergebnis wird modulo q genommen. Sofern das Ergebnis gleich Null ist, wird in der Matrix die entsprechende Stelle auf 1 gesetzt, andernfalls auf 0. Bei der beispielhaften Eingabe erhalten wir folgende Werte:

$$A_{3,3} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Es wurden ebenfalls deutlich höhere Werte für k und q getestet, die jedoch den Rahmen dieser Ausarbeitung sprengen würden.

3 Optimierungsproblem

Um das wachsende Optimierungsproblem $max\{<1,x> \text{ — } A_{k,q}x \leq b\}$ zu lösen, wird die Software GUROBI genutzt. Diese optimiert die Werte für den gesuchten Vektor x bei einem eingegeben Vektor c von

$$\text{c} = (1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1)^T$$
$$\text{x} = (0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1)^T$$

4 Codetabellen

Für den nächsten Schritt betrachten wir den Fall für q = 7 und k = 3. Dabei ergibt sich schon eine deutlich größere Matrix A, sowie der Vektor x wie folgt:

$$\text{x} = (1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1$$
$$1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0)^T$$

Die dazugehörige Generatormatrix G berechnet sich dabei anhand des Vektors x, der angibt, welche Repräsentanten an welche Stelle eingefügt werden sollen. Im gegeben Beispiel ergibt sich:

$$\text{G} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 2 & 2 & 3 & 3 & 4 & 4 & 5 & 5 & 6 & 6 \\ 1 & 5 & 3 & 3 & 6 & 0 & 5 & 0 & 5 & 5 & 6 & 2 & 6 & 2 & 3 \end{pmatrix}$$

Der daraus resultierende Code ist ein $[15, 3, 12]_q$ Code. Um die gegebene Minimaldistanz d zu verifizieren wird die Hamming-Distanz manuell berechnet. Dazu werden zunächst alle möglichen Nachrichtenwörter für die gegebenen q und k generiert (mit Ausnahme des Nullwortes) und mit der Generatormatrix G kodiert. Anschließend wird von jedem Codewort zu jedem anderen Codewort (mit Ausnahme zu sich selbst) die Hamming-Distanz berechnet und anschließend der Minimalwert zurückgeliefert. In unserem Beispiel ergibt sich dabei:
12 = calculated distance, 12 = should be the distance

5 Zyklenerkennung

Da mit steigendem q und k ebenfalls die Matrix $A_{k,q}$ anwächst wird es für GUROBI zunehmend aufwändiger eine Lösung für den Vektor x zu finden. Um die eingegebene Matrix zu reduzieren wird daher eine Erzeugermatrix e der Größe k x k gesucht, die anschließend auf die einzelnen Repräsentanten angewandt wird. Sofern eine Konvertierung in eine andere Gruppe möglich ist wird eine entsprechende Queue angelegt. In dem Fall, dass sowohl Anfangs- als auch Endpunkt das gleiche Element sind kann ein Zyklus festgehalten werden. Als Beispiel für q = 7, k = 3 und b = 3 kann die Matrix $A_{k,q}$ der Größe 57 x 57 auf eine 13 x 13 Matrix reduziert werden.

$$\text{e} = \begin{pmatrix} 6 & 0 & 0 \\ 6 & 5 & 4 \\ 6 & 0 & 4 \end{pmatrix}$$

Daraus ergeben sich folgende Gruppen für die Spalten:

S([r0]) = [r0], [r2], [r3], [r4], [r5], [r7]

S([r1]) = [r1]

S([r6]) = [r6]

S([r8]) = [r8], [r10], [r16], [r42], [r54], [r55]

S([r9]) = [r9], [r14], [r29], [r40], [r45], [r48]

S([r11]) = [r11], [r25], [r32]

S([r12]) = [r12], [r17], [r22], [r34], [r44], [r56]

S([r13]) = [r13], [r19], [r21], [r23], [r24], [r36]

S([r15]) = [r15], [r20], [r30], [r33], [r38], [r49]

S([r18]) = [r18], [r46], [r53]

S([r26]) = [r26], [r31], [r35], [r41], [r50], [r51]

S([r27]) = [r27], [r28], [r37], [r43], [r47], [r52]

S([r39]) = [r39]

Sowie für die Reihen (mit transponiertem e^T)

S^T([r0]) = [r0], [r9], [r11], [r12], [r13], [r14]

S^T([r1]) = [r1], [r7], [r24], [r25], [r35], [r44]

S^T([r2]) = [r2], [r18], [r26], [r42], [r43], [r51]

S^T([r3]) = [r3], [r4], [r15], [r20], [r32], [r39]

S^T([r5]) = [r5], [r38], [r53]

S^T([r6]) = [r6], [r33], [r46], [r48], [r50], [r54]

S^T([r8]) = [r8]

S^T([r10]) = [r10]

S^T([r16]) = [r16], [r21], [r22], [r31], [r52], [r55]

S^T([r17]) = [r17], [r29], [r40], [r41], [r45], [r47]

S^T([r19]) = [r19], [r23], [r49]

S^T([r27]) = [r27], [r28], [r30], [r36], [r37], [r56]

S^T([r34]) = [r34]

Im Anschluss daran werden in den Spalten die jeweiligen Gruppen zusammenaddiert bzw. in den Reihen die überflüssigen Reihen entfernt, sodass sich die verkleinerte Matrix

$$A_{3,7} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 2 & 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 2 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 2 & 2 & 1 & 0 & 0 & 0 \\ 6 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 3 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 2 & 0 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 0 & 2 & 0 & 0 & 2 & 0 & 1 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 2 & 0 & 1 & 0 & 1 & 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

mit den daraus resultierenden Vektoren

$$\begin{aligned} \mathrm{c} &= (6\ 1\ 1\ 6\ 6\ 3\ 6\ 6\ 6\ 3\ 6\ 6\ 1)^T \\ \mathrm{x} &= (0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1)^T \end{aligned}$$

Insgesamt ergibt sich dadurch ein $[15,\ 3,\ 12]_7$ Code.

6 Generierung des Erzeugers

Schließlich ist die Herausforderung einen Erzeuger e zu finden, der die Matrix A soweit vereinfacht, sodass GUROBI das Optimierungsproblem in einer angemessenen Zeit lösen kann. Bei den Tests ist aufgefallen, dass vergleichsweise kleine Matrizen A sich ebenfalls auf die Länge des Nachrichtenwortes n. Diese wird mit zunehmender Größe der Matrix A ebenfalls größer, sodass ein Codewort mit der Länge k ebenfalls in mehr Bits kodiert werden kann.

Um möglichst große n zu erzeugen muss nun der Erzeuger e so gewählt werden, dass er zwar die Matrix A in der Größe reduziert jedoch nur so weit wie nötig, sodass das Optimierungsproblem in akzeptabler Zeit lösbar ist bzw. überhaupt eine Lösung gefunden werden kann.

```
Testing for e0...
-----------------------------------------------------
Current producer e0 reduces A from [57] to [57] columns.
Current producer e0 reduces A from [57] to [13] columns.
Found Producer e0 - reducing to [13] groups
6 0 0
6 5 4
6 0 4
-----------------------------------------------------

Found cycles for columns
-----------------------------------------------------
S([r0]) = {[r0], [r2], [r3], [r4], [r5], [r7]}
S([r1]) = {[r1]}
S([r6]) = {[r6]}
S([r8]) = {[r8], [r10], [r16], [r42], [r54], [r55]}
S([r9]) = {[r9], [r14], [r29], [r40], [r45], [r48]}
S([r11]) = {[r11], [r25], [r32]}
S([r12]) = {[r12], [r17], [r22], [r34], [r44], [r56]}
S([r13]) = {[r13], [r19], [r21], [r23], [r24], [r36]}
S([r15]) = {[r15], [r20], [r30], [r33], [r38], [r49]}
S([r18]) = {[r18], [r46], [r53]}
S([r26]) = {[r26], [r31], [r35], [r41], [r50], [r51]}
S([r27]) = {[r27], [r28], [r37], [r43], [r47], [r52]}
S([r39]) = {[r39]}

Found cycles for rows
-----------------------------------------------------
ST([r0]) = {[r0], [r9], [r11], [r12], [r13], [r14]}
ST([r1]) = {[r1], [r7], [r24], [r25], [r35], [r44]}
ST([r2]) = {[r2], [r18], [r26], [r42], [r43], [r51]}
ST([r3]) = {[r3], [r4], [r15], [r20], [r32], [r39]}
ST([r5]) = {[r5], [r38], [r53]}
ST([r6]) = {[r6], [r33], [r46], [r48], [r50], [r54]}
ST([r8]) = {[r8]}
ST([r10]) = {[r10]}
ST([r16]) = {[r16], [r21], [r22], [r31], [r52], [r55]}
ST([r17]) = {[r17], [r29], [r40], [r41], [r45], [r47]}
ST([r19]) = {[r19], [r23], [r49]}
ST([r27]) = {[r27], [r28], [r30], [r36], [r37], [r56]}
ST([r34]) = {[r34]}
-----------------------------------------------------
```

```
Re-calculate matrix A with cylces
-----------------------------------------------------
0 1 0 1 1 0 1 1 1 0 1 1 0
1 0 0 2 2 1 1 1 0 0 0 0 0
1 0 0 1 1 0 0 1 1 0 1 1 1
1 0 0 2 0 0 1 0 0 1 2 1 0
0 0 1 2 0 1 0 0 2 0 0 2 0
1 0 0 1 0 0 1 2 2 1 0 0 0
6 1 1 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 3 0 0 0 3 0 0 1
1 0 0 0 1 1 1 0 2 0 2 0 0
1 0 0 0 0 1 1 2 0 0 1 2 0
0 0 1 0 2 0 0 2 0 1 2 0 0
1 0 0 0 2 0 1 0 1 1 0 2 0
0 0 1 0 0 0 6 0 0 0 0 0 1
-----------------------------------------------------

Calculate vector c
-----------------------------------------------------
6 1 1 6 6 3 6 6 6 3 6 6 1
-----------------------------------------------------
Calculating vector x
-----------------------------------------------------
0 1 1 0 0 0 0 0 0 0 1 1 1
-----------------------------------------------------

We have now following code
-----------------------------------------------------
[n,k,d]_q
[15,3,12]_7
-----------------------------------------------------

Generate Generatormatrix G
-----------------------------------------------------
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 2 3 3 4 6 6 2 2 4 5 5 6 4
0 5 4 2 6 5 0 1 5 6 1 0 4 2 3
-----------------------------------------------------

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