Prüfungsvorleistung Codierungs-Theorie

David Tarnow, Michael Narkus, Armel Siewe, Lukas Köhler

26. 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, Matritzen und Kuben bereitgestellt. Im Rahmen dieses Projektes wurde Armadillo in der neusten stabilen Version (8.500.1) verwendet.

1.3 Gurobi

1.4 LAPACK

2 Schritt 1

Ziel von Schritt 1 war es, anhand der Werte für k und q, eine Matrix zur konstruieren. Die Anwendung nimmt zunächst die Werte entgegen und berechnet eine Sammlung von linear unabhängigen Vektoren.
Sind alle 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.

2.1 Generierung von linear unabhängigen Vektoren

Listing 1: Generierung der Vektoren

```
for (auto k_index = k - 1; k_index >= 0; --k_index)
{
  vec init_vec(k); // set size to k
  init_vec.fill(0); // init with zero
  init_vec.at(k_index) = 1;
  ret.push_back(init_vec);
  calc_lin_independ_vec(ret, &init_vec, 1, q, k_index, k);
}
```

Listing 2: Berechnung der linearen Unabhängigkeiten

```
k_index++;
auto construct_vec = *init_vec;
calc_lin_independ_vec(ret, &construct_vec, q_index, q,
   k_index, k);

while (construct_vec.at(k_index) < (q - 1))
{
  construct_vec.at(k_index) += 1;
  calc_lin_independ_vec(ret, &construct_vec, q_index + 1, q,
      k_index, k);
}
```

2.2 Erstellen der Matrix

Listing 3: Erstellen der Generatormatrix

```
arma::mat A(mat_size, mat_size);
for (auto i = 0; i < mat_size; ++i)
```

```
{
  for (auto j = 0; j < mat_size; ++j)
  {
    const auto res = static_cast<int>(dot(lin_independ_vecs.
        at(i), lin_independ_vecs.at(j))) % q;
    A(i, j) = 0 == res ? 1 : 0;
  }
}
```

2.3 Tests

Getestet wurde dieser Teil der Anwendung u.a. für die folgenden Wertepaare:

Ergebnis für q=2 und k =2: $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

q=2 und k=3: $\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$

q=3 und k=2: $\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

Neben den oben aufgeführten Werten wurden noch deutlich größere Eingaben, z.B. .. getestet. Die Matritzen 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.

3.1 Optimierung mit Gurobi

Die Lösung des eigentlichen 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. Die 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.

3.2 Tests

4 Schritt 3

Schritt 3 erweitert die, aus Schritt 2, bestehende Optimierungsproblem um ein wachsendes b ¿= 1. Ist das Optimierungsproblem 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.

4.1 Vorgehensweise

4.2 Tests

5 Schritt 4

6 Schritt 5

Listings