

Implementierung eines Simplex-Algorithmus

Präsentiert von Jean Sokolov

Inhaltsverzeichnis

- Problemstellung
- Simplex-Algorithmus
 - Initialisierung
 - Iteration
 - Stopbedingungen
- Implementation in Rust
- Praktische Demonstration

PROBLEMSTELLUNG

Problemstellung

- Lineare Optimierung
- Standardminimierungsprobleme

$$\begin{array}{ll}\text{Minimize} & Z = 12x_1 + 16x_2 \\ \text{Subject to:} & x_1 + 2x_2 \geq 40 \\ & x_1 + x_2 \geq 30 \\ & x_1 \geq 0; x_2 \geq 0\end{array}$$

- Simplex-Algorithmus in Rust

SIMPLEX-ALGORITHMUS

Simplex-Algorithmus

- Besteht aus drei Hauptschritten:
- Initialisierung
- Iteration
- Stopbedingungen

Simplex: Initialisierung

- Konvertierung Mini- zu Maximierung
- Erstellen einer Matrix mit Zielfunktion und Bedingungen

Minimize $Z = 12x_1 + 16x_2$
Subject to: $x_1 + 2x_2 \geq 40$
 $x_1 + x_2 \geq 30$
 $x_1 \geq 0; x_2 \geq 0$



1	2	40
1	1	30
<hr/>		<hr/>
12	16	0

- Transponieren der resultierenden Matrix

Simplex: Initialisierung

$$\begin{array}{cc|c}
 1 & 2 & 40 \\
 1 & 1 & 30 \\
 \hline
 12 & 16 & 0
 \end{array}$$

Transponieren

$$\begin{array}{cc|c}
 1 & 1 & 12 \\
 2 & 1 & 16 \\
 \hline
 40 & 30 & 0
 \end{array}$$

Standardmaximierungs-
problem

$$\begin{array}{ll}
 \text{Maximize} & Z = 40y_1 + 30y_2 \\
 \text{Subject to:} & y_1 + y_2 \leq 12 \\
 & 2y_1 + y_2 \leq 16 \\
 & y_1 \geq 0; y_2 \geq 0
 \end{array}$$

Simplex: Initialisierung

- Generieren von Slackvariablen

$$\begin{array}{ll}
 \text{Maximize} & Z = 40y_1 + 30y_2 \\
 \text{Subject to:} & y_1 + y_2 \leq 12 \\
 & 2y_1 + y_2 \leq 16 \\
 & y_1 \geq 0; y_2 \geq 0
 \end{array}$$



$$\begin{array}{ll}
 \text{Maximize} & Z = 40y_1 + 30y_2 + 0s_1 + 0s_2 \\
 \text{Subject to:} & y_1 + y_2 + s_1 = 12 \\
 & 2y_1 + y_2 + s_2 = 16 \\
 & y_1 \geq 0; y_2 \geq 0; s_1 \geq 0; s_2 \geq 0
 \end{array}$$

Simplex: Iteration

- Verbessern des Ergebnisses der Kostenfunktion
- Auswahl vielversprechendster Variable
- Vielversprechend = größter Vorfaktor in Kostenfunktion

$$\begin{array}{ll}\text{Maximize} & Z = 40y_1 + 30y_2 + 0s_1 + 0s_2 \\ \text{Subject to:} & y_1 + y_2 + s_1 = 12 \\ & 2y_1 + y_2 + s_2 = 16 \\ & y_1 \geq 0; y_2 \geq 0; s_1 \geq 0; s_2 \geq 0\end{array}$$

Simplex: Iteration

- Auswahl der limitierensten Bedingung

$$\begin{aligned}
 \text{Maximize} \quad & Z = 40y_1 + 30y_2 + 0s_1 + 0s_2 \\
 \text{Subject to:} \quad & y_1 + y_2 + s_1 = 12 \\
 & 2y_1 + y_2 + s_2 = 16 \\
 & y_1 \geq 0; y_2 \geq 0; s_1 \geq 0; s_2 \geq 0
 \end{aligned}$$

$$\begin{aligned}
 & \text{I} \quad y_1 + y_2 + s_1 = 12 \\
 & \text{II} \quad 2y_1 + y_2 + s_2 = 16 \\
 \Rightarrow & \text{I} \quad y_1 = 12 \\
 & \text{II} \quad 2y_1 = 16 \\
 \Rightarrow & \text{I} \quad y_1 = 12 \\
 & \text{II} \quad y_1 = 8
 \end{aligned}$$

Simplex: Iteration

$$\text{II } 2y_1 + y_2 + s_2 = 16$$

$$\rightarrow y_1 = 8 - \frac{1}{2} y_2 + \frac{1}{2} s_2$$

- Einsetzen in Gleichungssystem:

$$Z = 40 \left(8 - \frac{1}{2} y_2 + \frac{1}{2} s_2 \right) + 30 y_2 + 0 s_1 + 0 s_2$$

$$\text{I } \left(8 - \frac{1}{2} y_2 + \frac{1}{2} s_2 \right) + y_2 + s_1 = 12$$

$$\text{II } y_1 + \frac{1}{2} y_2 + \frac{1}{2} s_2 = 8$$

Simplex: Iteration

- Lösen des Gleichungssystems:

$$Z = 320 + 10 y_2 + 0 s_1 + 20 s_2$$

$$\text{I} \quad \frac{1}{2} y_2 + s_1 + \frac{1}{2} s_2 = 4$$

$$\text{II} \quad y_1 + \frac{1}{2} y_2 + \frac{1}{2} s_2 = 8$$

- Evaluieren der Kostenfunktion:
- $Z = 320$

Simplex: Stopbedingung

- Berechnung bis Optimum gefunden
=> Keine weitere Berechnung ist möglich
=> Alle Variablen der Kostenfunktion positiv
=> Wert der Kostenfunktion verschlechtert sich

Simplex: Stopbedingung

$$Z = 400 - 200 s_1 + 10 s_2$$

$$\text{I} \quad y_2 + 2 s_1 + s_2 = 8$$

$$\text{II} \quad y_1 - s_1 = 4$$

- Variablen der Kostenfunktion sind ≥ 0
- Kosten von 320 \rightarrow 400

IMPLEMENTATION IN RUST

Datei einlesen

```
275 fn main() {  
276     // Read file  
277     let mut file_content: Vec<String> = read_file().unwrap();  
278  
279     // separate objective function from rest of file content (constraints)  
280     let objective_function: &String = &file_content.clone()[0];  
281     file_content.remove(index: 0);
```

```
7  const DEFAULT_PATH: &'static str = "./target/release/KI_30.txt";  
8  
9  // Reading file and removing comments such as "// Objective function"  
10 fn read_file() -> std::io::Result<Vec<String>> {  
11     let mut args: Args = std::env::args();  
12     let file_path: String = args.nth(1).unwrap_or_else(|| DEFAULT_PATH.to_string());  
13     let file: File = File::open(file_path)?;  
14     let reader: BufReader<File> = BufReader::new(inner: file);  
15     let mut tmp: Vec<String> = Vec::new();  
16     for lines: Result<String, Error> in reader.lines() {  
17         let tmpstr: String = lines.unwrap();  
18         if !tmpstr.contains("//") {  
19             tmp.push(tmpstr);  
20         }  
21     }  
22     return Ok(tmp);  
23 }
```

Kostenfunktion parsen

```
283 // parse file contents
284 let mut objective_fn: Vec<f64> = generate_matrix_objective_fn(objective_function);
285 let constraints: Vec<Vec<f64>> = generate_matrix_constraints(&mut file_content);
286
```

```
43 /// Iterates through the substrings, which are split of the objective function at '+'
44 /// Then splits substrings at '*' to get coefficient of variables
45 /// Parses and transforms the variables into a 64-bit float vector
46 /// e.g. min: + 3*x0 + 2*x1; to [3.0,2.0]
47 // slack variables are added in function add_slack_variables_objective_fn
48 fn generate_matrix_objective_fn(objective_function: &String) -> Vec<f64> {
49     let mut objective_fn: Vec<f64> = Vec::new();
50     let variables: Vec<&str> = objective_function.split('+').collect();
51     for s: &str in variables.into_iter().skip(1) {
52         objective_fn.push(
53             s.trim().split('*').collect::() Result<f64, ParseFloatError>
55                 .unwrap(),
56         );
57     }
58     return objective_fn;
59 }
60
```

Bedingungen parsen

```
61 /// Iterates through the file content, which contains the constraints
62 /// Parses and transforms the individual lines/strings into 64-bit float vectors
63 /// e.g. + 3*x0 + 2*x1 >= 12 to [3.0,2.0,12.0]
64 /// slack variables are added in function add_slack_variables_constraints
65 fn generate_matrix_constraints(file_content: &mut Vec<String>) -> Vec<Vec<f64>> {
66     let mut constraints: Vec<Vec<f64>> = Vec::new();
67     for s: &mut String in file_content {
68         let tmp_vec_str: Vec<&str> = s.split('+').collect::<Vec<&str>>();
69         let mut tmp_vec_f64: Vec<f64> = Vec::new();
70
71         // grabbing lhs
72         for t: &str in tmp_vec_str.clone().into_iter().skip(1) {
73             tmp_vec_f64.push(
74                 t.trim().split('*').collect::<Vec<&str>>()[0] &str
75                     .parse::<f64>() Result<f64, ParseFloatError>
76                     .unwrap(),
77             );
78         }
79
80         // grabbing rhs
81         tmp_vec_f64.push(
82             tmp_vec_str Vec<&str>
83                 .clone() Vec<&str>
84                 .last() Option<&str>
85                 .unwrap() &&str
86                 .trim() &str
87                 .split(">=") Split<'_', &str>
88                 .collect::<Vec<&str>>()[1] &str
89                 .replace(from: ";", to: "") String
90                 .replace(from: " ", to: "") String
91                 .parse::<f64>() Result<f64, ParseFloatError>
92                 .unwrap(),
93         );
94         constraints.push(tmp_vec_f64);
95     }
96
97     return constraints;
98 } fn generate_matrix_constraints
```

Matrix anlegen und transponieren

```
287 // transpose matrix with constraints and objective function
288 let mut transposed_matrix: Vec<Vec<f64>> = constraints.clone();
289 transposed_matrix.push(objective_fn);
290 transposed_matrix = transpose_matrix(transposed_matrix);
291
292 // grab transposed objective_fn
293 objective_fn = transposed_matrix.pop().unwrap();
294
```

```
237 /// transposes the matrix consisting of constraints and objective function
238 /// this process turns a standard minimization problem into a standard maximization problem
239 fn transpose_matrix(m: Vec<Vec<f64>>) -> Vec<Vec<f64>> {
240     let mut t: Vec<Vec<f64>> = vec![Vec::with_capacity(m.len()); m[0].len()];
241     for r: Vec<f64> in m {
242         for i: usize in 0..r.len() {
243             t[i].push(r[i]);
244         }
245     }
246     t
247 }
```

Slackvariablen: Kostenfunktion

```
295 // add slack variables to transposed matrix
296 objective_fn = add_slack_variables_objective_fn(m: objective_fn, n: transposed_matrix.len());
297 transposed_matrix = add_slack_variables_constraints(transposed_matrix);
298
299 //println!("\n\nObjective f: {:?}", objective_fn);
300 //println!("Constraints: {:?}", transposed_matrix);
```

```
265 /// adds slack variables to the objective function
266 fn add_slack_variables_objective_fn(m: Vec<f64>, n: usize) -> Vec<f64> {
267     let mut t: Vec<f64> = m;
268     // add n slack-variables to objective fn, n being the number of constraints
269     for i: usize in 0..n {
270         t.push(0.0);
271     }
272     t
273 }
274
```

Slackvariablen: Bedingungen

```
249 /// adds slack variables to the constraints
250 fn add_slack_variables_constraints(m: Vec<Vec<f64>>) -> Vec<Vec<f64>> {
251     let mut t: Vec<Vec<f64>> = m.clone();
252     for index: usize in 0..m.len() {
253         for _i: usize in 0..m.len() {
254             // add n slack-variables to each constraint, n being the number of constraints
255             t[index].push(0.0);
256         }
257         // swap initial rhs with last element of constraint, as rhs value is not the right-most value after appending the slacks
258         t[index].swap(a: m[0].len() + m.len() - 1, b: m[0].len() - 1);
259         // setting this constraint's slack variable to 1
260         t[index][m[0].len() + index - 1] = 1.0;
261     }
262     t
263 }
264
```

Ausführen des Algorithmus

```
302 // Run simplex algorithm
303 solve(objective_fn, constraints: transposed_matrix);
304
```

```
100 fn solve(mut objective_fn: Vec<f64>, mut constraints: Vec<Vec<f64>>)) {
101     let mut i: usize = 0;
102     let mut cost: f64 = 0.0;
103     let mut init_cost: f64 = INFINITY;
104     let mut init_obj_fn: Vec<f64> = objective_fn.clone();
105     loop {
106         // find most promising variable, ignoring variables with value of 0
107         let mut tmp_obj_fn: Vec<f64> = objective_fn.clone();
108         for n: usize in 0..tmp_obj_fn.len() {
109             if tmp_obj_fn[n].eq(&0.0) {
110                 tmp_obj_fn[n] = -INFINITY;
111             }
112         }
113
114         let index_max_factor: usize = [tmp_obj_fn Vec<f64>
115             .iter() Iter<'_, f64>
116             .enumerate() impl Iterator<Item = (usize, ...)>
117             .max_by(compare: |(_, a: &&f64), (_, b: &&f64)| a.total_cmp(b)) Option<(usize, &f64)>
118             .map(|(index: usize, _)| index)] Option<usize>
119         .unwrap();
```

Finden der limitierendensten Bedingung

```

125 // find most limiting constraint
126 let mut lhs: Vec<f64> = Vec::new();
127 let mut rhs: Vec<f64> = Vec::new();
128 for constraint: Vec<f64> in constraints.clone() {
129     lhs.push(constraint[index_max_factor]);
130     rhs.push(constraint.last().unwrap().to_owned());
131 }
132 let mut n: usize = 0;
133
134 // iterate through the this iteration's constraints,
135 // dividing rhs value by coefficient of this iteration's variable e.g. 3*x0=12 to x0=4
136 // while doing so, check that the variables respect the non-negative constraint
137 while n < lhs.len() {
138     //println!("Pre Calc {:?}={:?}", lhs[n], rhs[n]);
139     if lhs[n].ge(&0.0) {
140         if rhs[n].ge(&0.0) {
141             rhs[n] = rhs[n] / lhs[n];
142             lhs[n] = 1.0;
143         } else {
144             rhs[n] = INFINITY;
145             lhs[n] = INFINITY;
146         }
147     } else {
148         if rhs[n].ge(&0.0) {
149             rhs[n] = INFINITY;
150             lhs[n] = INFINITY;
151         } else {
152             rhs[n] = rhs[n] / lhs[n];
153             lhs[n] = 1.0;
154         }
155     }
156     //println!("Post Calc {:?}={:?}", lhs[n], rhs[n]);
157     n += 1;
158 }
159
160 // determine index/row of most restrictive constraint for most significant variable in objective function of this iteration
161 let most_significant_constraint_index: usize = (rhs Vec<f64>
162     .iter() Iter<'_, f64>
163     .enumerate() impl Iterator<Item = (usize, ...)>
164     .min_by(compare: |(<_, a: &&f64>), (<_, b: &&f64>)| a.total_cmp(b)) Option<(usize, &f64)>
165     .map(|(index: usize, _)| index) Option<usize>
166     .unwrap());
167 // grab value/vector behind determined index
168 let mut most_significant_constraint: &mut Vec<f64> = &mut constraints[most_significant_constraint_index];

```


Limitierenste Bedingung umformen

```
170 | let x: Vec<f64> = transform_equation(&mut most_significant_constraint, current_index: index_max_factor);
25 | /// Transformes an equation/singular constraint to have the lhs be a singular variable, which then can be inserted into the other constraints and objective function
26 | /// e.g. + 3*x0 + 2*x1 + s1 = 12 to x0 = 12 - 2/3*x1 - 1/3*s1
27 | /// or rather as matrix:
28 | /// [3.0,2.0,1.0,12.0] to [1.0,-0.66667,-0.33333,-12.0]
29 | /// the rhs value can be inverted aswell, as upon insertion (if rhs>=0) you end up with a positive value on the lhs, which has to be subtracted from the rhs value anyways
30 | fn transform_equation(
31 |     most_significant_constraint: &mut Vec<f64>,
32 |     current_index: usize,
33 | ) -> Vec<f64> {
34 |     let mut transformed_equation: Vec<f64> = Vec::new();
35 |     for mut values: f64 in most_significant_constraint.clone() {
36 |         values = values / most_significant_constraint[current_index] * -1.0;
37 |         transformed_equation.push(values);
38 |     }
39 |
40 |     return transformed_equation;
41 | }
```

Neuaufstellen des Gleichungssystem

```
172 // insert transformed equation/value of currently selected variable into all constraints other than currently selected constraint/row
173 // for currently selected constraint: normalize values, so that coefficient of currently selected variable is 1
174 // e.g. + 3*x0 + 2*x1 + 1*s1 = 12 to + x0 + 2/3*x1 + 1/3*s1 = 4
175 for j: usize in 0..lhs.len() {
176     let mult: f64 = constraints[j][index_max_factor];
177     if j.ne(&most_significant_constraint_index) {
178         for k: usize in 0..constraints[j].len() {
179             constraints[j][k] += x[k] * mult;
180         }
181         constraints[j][index_max_factor] = 0.0;
182     } else {
183         for k: usize in 0..constraints[j].len() {
184             constraints[j][k] /= mult;
185         }
186     }
187 }
188
189 let mult: f64 = objective_fn[index_max_factor];
190
191 // insert transformed equation/value of currently selected variable into objective function
192 for k: usize in 0..x.len() {
193     if k.ne(&x.len().checked_sub(1).unwrap()) {
194         objective_fn[k] += x[k] * mult;
195     } else {
196         cost += x.last().unwrap() * mult;
197     }
198 }
199 objective_fn[index_max_factor] = 0.0;
202
```

Stopbedingung

```
203 println("Initial rhs of objective fn {:?}, changed to {:?} with this iteration. Value changed by: {:?}", init_cost, cost, cost - init_cost);
204 // if stop condition (minimization -> cost of objective function increasing) is triggered, print the values of the objective function and variables that were calculated
205 if cost.gt(&init_cost) {
206     println(
207         "\n_____ \n\nOptimal solution found.\nVariables are:\n"
208     );
209     // grab all non-slack-variables
210     let mut variables: Vec<f64> = vec![0.0; objective_fn.len()];
211     for n: usize in 0..init_obj_fn.len() {
212         if n.ge(&(init_obj_fn.len() - constraints.len())) {
213             variables[n] = init_obj_fn[n];
214         }
215     }
216     // print all variables with a value, writing their name and value to the output/console
217     for n: (usize, &f64) in variables.iter().enumerate() {
218         if n.1.ne(&0.0) {
219             println(
220                 "x{} = {}",
221                 n.0 - (init_obj_fn.len() - constraints.len()),
222                 n.1.abs()
223             );
224         }
225     }
226     // print final cost of objective function
227     println("p = {}", init_cost.abs());
228     break;
229 }
230 //println!("\nNEW OBJECTIVE FUNCTION: {:?}= {:?}", objective_fn);
231 init_cost = cost;
232 init_obj_fn = objective_fn.clone();
233 i += 1;
234 }
235 println!("\nDone after {} iterations", i + 1);
236 } fn solve
```

Quellcode

- https://github.com/JeanSokolov/sat_solver_v2/blob/master/src/main.rs#L275

PRAKTISCHE DEMONSTRATION

Vielen Dank für Ihre Aufmerksamkeit