jugend forscht
schüler experimentieren

2017

Tims MasterMind Solver - Lösungsstrategien zum Spiel „MasterMind“

Von Tim Palm

Klasse 9a

LGH Schwäbisch Gmünd

Betreuerin: Dr. Olga Lomonosova

Kurzfassung

Bei dem Gesellschaftsspiel „MasterMind“ handelt es sich um ein Logikspiel der Firma Haspro® & Parker-Brothers®. Das Spiel wird zu zweit gespielt. Ein Spieler ist der Codemaker, der andere der Codebreaker. Der Codemaker steckt zunächst einen vierstelligen Code aus sechs verschiedenen Farben (z.B. Gelb, Gelb, Grün, Rot). Dieser Code ist für den Codebreaker nicht sichtbar, welcher nun versuchen muss, ihn zu knacken. Dazu hat er zehn Versuche. Nach jedem Versuch bekommt er vom Codemaker eine Antwort auf seinen Versuch in Form von weißen und schwarzen Steckern. Die schwarzen Stecker stehen für eine richtige Farbe an richtiger Position und die weißen für eine Farbe, die zwar im Code vorkommt, allerdings noch nicht an der richtigen Position. Durch diese Stecker bekommt der Codebreaker nun Informationen, die er für seinen nächsten Zug verwenden kann.

Für das Spiel wurden verschiedene Lösungsstrategien entworfen und in einem Computerprogramm umgesetzt. Zudem wurde ein Algorithmus aus einer Veröffentlichung untersucht. Durch statistische Analysen konnten die eigenen sowie der veröffentlichte Algorithmus verbessert werden.

Die einzelnen Algorithmen wurden in folgenden Kategorien untersucht:

- Summe aller benötigten Versuche für alle 1296 Codes
- Maximum der benötigten Versuche für einen Code
- Durchschnittliche Anzahl der benötigten Versuche pro Code
- Rechenzeit, um alle 1296 Codes zu lösen

Darauf aufbauend wurde ein neuer Ansatz entworfen, der von allen behandelten Algorithmen der Beste ist (siehe Tabelle in Abschnitt 3.2.4).

Das Programm besteht aber nicht nur aus statistischen Untersuchungen, sondern auch aus einem Spielmodus, in dem man gegen den PC spielt. Auch wurde ein Modus implementiert, der eine Hilfestellung zu einem Spieldurchlauf in der realen Welt liefert.

Inhalt

1 Einleitung

Bei dem Gesellschaftsspiel „MasterMind“ handelt es sich um ein Logikspiel der Firma Haspro & Parker Brothers®. Das Spiel wird zu zweit gespielt. Ein Spieler ist der Codemaker, der andere der Codebreaker. Der Codemaker steckt zunächst einen vierstelligen Code aus sechs verschiedenen Farben (z.B. Gelb, Gelb, Grün, Rot). Dieser Code ist für den Codebreaker nicht sichtbar, welcher nun versuchen muss, ihn zu knacken. Dazu hat er zehn Versuche. Nach jedem Versuch bekommt er vom Codemaker eine Antwort auf seinen Versuch in Form von weißen und schwarzen Steckern. Die schwarzen Stecker stehen für eine richtige Farbe an richtiger Position und die weißen für eine Farbe, die zwar im Code vorkommt, allerdings noch nicht an der richtigen Position. Durch diese Stecker bekommt der Codebreaker nun Informationen, die er für seinen nächsten Zug verwenden kann. Das Spiel endet entweder, wenn nach zehn Zügen der Code immer noch nicht erraten wurde (dann hat der Codemaker gewonnen) oder wenn der Code erraten wurde (dann hat der Codebreaker gewonnen).

Da die Erfahrung gezeigt hat, dass es nicht immer einfach ist, den Code in zehn Versuchen zu lösen, hatte ich die Idee, dafür eine Hilfestellung in Form eines Computerprogramms zu entwerfen. Von dort aus kann man das Spiel mit drei Algorithmen lösen. Zwei habe ich mir selbst ausgedacht. Der andere entstammt aus einer Veröffentlichung von Donald E. Knuth.

Ich habe das Programm immer weiter verbessert und neue Inhalte hinzugefügt, sodass man schlussendlich aus vier Spielmodi, welche später noch erwähnt werden, auswählen kann. Des Weiteren habe ich es sogar geschafft, den Algorithmus von Knuth, welcher seit 1977 existiert, zu verbessern und einen komplett neuen Algorithmus zu entwerfen.

In der unten angegebenen Tabelle sind zum besseren Verständnis Begriffsdefinitionen, welche im Verlauf dieser Arbeit mehrfach verwendet werden, aufgeführt.

Codemaker	Der Codemaker erstellt den geheimen Code und gibt die Bewertungen an den Codebreaker.
Codebreaker	Der Codebreaker versucht, den geheimen Code des Codemakers zu knacken.
Code	Dies meint den geheimen Code des Codemakers.
Tipp	Rateversuch des Codebreakers
Bewertung	Dies bezeichnet die Antwort vom Codemaker an den Codebreaker in Form von weißen und schwarzen Steckern. Die Bewertung wird aus platzsparenden Gründen folgendermaßen formuliert: (schwarz, weiß) z.B. (2,1) steht für zwei Schwarze und einen weißen Stecker.
Farbcodes	0 1 2 3 4 5

Beispielhaft ist ein möglicher Spielverlauf aufgetragen:
Für den ersten Tipp gab es einen schwarzen Stecker für Grün, da Grün sich schon auf richtiger Position befindet. Rot dagegen noch nicht, weswegen es nur einen weißen Stecker gibt. Im zweiten Versuch gab es drei weiße Stecker als Antwort. Kein Tipp-Stecker ist an der richtigen Position, allerdings kommen Gelb, Rot und Grün im Code vor. Im dritten Tipp gab es zwei schwarze und einen weißen Stecker. Die Schwarzen repräsentieren die mittleren Tipp-Stecker (Grün u. Gelb), die schon an richtiger Position sind. Im

vierten Tipp wurden alle Farben bereits gefunden, wofür es als Antwort vier Stecker gab. Zwei Farben sind noch vertauscht. Im fünften Versuch wurde das Spiel gewonnen, da der Code erraten wurde und viermal Schwarz als Antwort zurückgegeben wurde.

2 Vorgehensweise

2.1 Lösungsalgorithmen

Bei sechs Farben und vier Steckplätzen ergeben sich insgesamt 6^4 = 1296 denkbare Codes. Der Codebreaker muss demnach aus allen diesen Codes den Gesuchten herausfinden. Dazu sind verschiedene Ansätze möglich. Ein denkbarer Algorithmus funktioniert folgendermaßen: Am ersten Steckplatz wird die Farbe solange durchgewechselt, bis ein schwarzer Stecker hinzukommt. Somit hat man die richtige Farbe im ersten Steckplatz gefunden. Dieses Vorgehen wiederholt man nun für die folgenden Positionen. Dies ist aber für den normalen Spieldurchlauf unbrauchbar, denn dieses Verfahren benötigt für einen Großteil der Codes weitaus mehr als zehn Versuche.

Mein eigener Ansatz verfolgt daher eine andere Strategie. Es wird von einer Liste mit allen 1296 möglichen Codes ausgegangen. Danach nimmt man den ersten Code in der Liste als ersten Rateversuch, welcher dann anschließend auch bewertet wird. Es sind demnach insgesamt die 14[1] folgenden Bewertungen möglich (siehe Tabelle in Kapitel 1).

(0,4)	(0,3)	(0,2)	(0,1)	(0,0)	(1,3)	(1,2)	(1,1)	(1,0)	(2,2)	(2,1)	(2,0)	(3,0)	(4,0)

Im folgenden Schritt wird die Liste mit noch gültigen Codes geprüft. Es wird durchlaufend angenommen, dass jeder Code in der Liste der Gesuchte ist. Dazu wird der aktuelle Rateversuch mit jedem dieser Codes bewertet. Stimmt diese Bewertung nicht mit der überein, die man für den vorherigen Rateversuch bekommen hat, so kann dieser Code nicht der Gesuchte sein. In diesem Fall wird der Code aus der Liste entfernt. In der Tippliste befinden sich also immer nur Codes, die zu allen bisherigen Bewertungen konsistent sind. Im Anschluss wird der erste Code der reduzierten Liste als nächster Rateversuch verwendet und die Reduktion der Liste wiederholt sich. Wenn die Liste nur noch ein Element enthält, ist das der gesuchte Code und das Spiel ist gewonnen.

Dieser Algorithmus ist folgendermaßen strukturiert:

1. Erstelle eine Liste (Tippliste) mit allen 1296 möglichen Codes.
2. Starte mit einem (möglichst optimalen) Starttipp.
3. Wenn der Codebreaker vier schwarze Stecker als Bewertung zurückbekommt, ist das Spiel gewonnen und der Algorithmus endet.
4. Wenn nicht, dann entferne jedes Element aus der Tippliste, das, angenommen es wäre der Code, nicht dieselbe Bewertung in Bezug auf den aktuellen Tipp bekommen hätte.
5. Wähle das erste Element der reduzierten Tippliste als nächsten Tipp.
6. Wiederhole die Aktionen ab Schritt drei solange, bis vier schwarze Stecker als Antwort zurückgegeben werden.

[1] Die Bewertung (3,1) ist nicht möglich, da bei drei korrekten Steckern die vierte richtige Farbe bereits auch an richtiger Position sein muss.

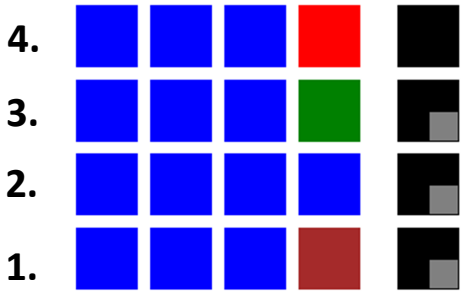
Zum Vergleich wurde noch ein Algorithmus aus einer 1977 erschienenen Veröffentlichung des amerikanischen Informatikprofessors Donald E. Knuth[2] herangezogen. Dieser Algorithmus gilt als einer der effizientesten zur Lösung des MasterMind-Spiels. Er garantiert die Lösung für jeden beliebigen Code in maximal fünf Versuchen. Zu Beginn ähnelt der Ansatz meinem Algorithmus. Ebenfalls wird eine Tippliste erstellt, die in ihrer Größe immer weiter reduziert wird. Der Unterschied liegt in der Wahl des neuen Tipps. Dieser wird auf folgendem Wege gefunden.

1. Bewerte alle möglichen noch unbenutzten 1296 Tipps nach folgenden Kriterien.
    a. Bestimme für jede der 14 möglichen Bewertungen die Größe der dadurch reduzierten Tippliste. (man erhält eine Datenliste, gemäß Kap.3.2.3)
    b. Die Wertigkeit eines Tipps ist das Maximum der Größe der potenziellen Tippliste.
    c. Der Tipp mit dem kleinsten Maximum, d.h. mit der größten Reduktion der aktuellen Tippliste, wird als neuer Tipp herangezogen.
    d. Wenn es mehrere Tipps gleicher Wertigkeit gibt, wird vorzugsweise ein Mitglied aus der aktuellen Tippliste verwendet.

Die wesentlichen Unterschiede zwischen meinem und Knuths Algorithmus liegen zum einen darin, dass ich als neuen Tipp den ersten Eintrag der Tippliste verwende, während Knuth jeden möglichen Tipp zunächst anhand des Minmax-Prinzips bewertet und daraufhin auswählt. Er schaut sozusagen in die Zukunft. Zum anderen ist es bei Knuths Algorithmus möglich, dass auch ein Tipp, der nicht mehr in der Tippliste vorhanden ist, als neuer Rateversuch genommen wird, da dieser die Tippliste am stärksten reduziert. Dies kann, wie das folgende Beispiel zeigt, ein Vorteil sein.

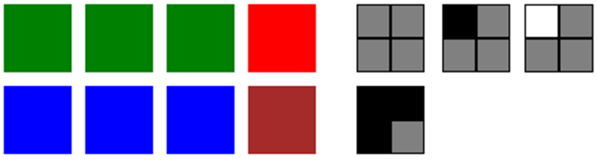
Ausgangspunkt ist die folgende Spielsituation. Für die Positionen 1 bis 3 ist bereits geklärt, dass es sich um blaue Stecker handeln muss. Für Position 4 kommen nur noch die drei Farben (Blau, Grün, Rot) Frage.

Mein Algorithmus würde die oben genannte Situation folgendermaßen angehen.



Da ausschließlich Tipps aus der aktuellen Tippliste verwendet werden, prüft der Algorithmus die drei verbliebenden Möglichkeiten nacheinander. Im ungünstigsten Fall benötigt dies drei Versuche.

Im Algorithmus von Knuth würde zunächst der Tipp (1, 1, 1, 2) (s. Tab. in Kap. 1) ausgewählt werden.



[2] Donald E. Knuth, ”The computer as master mind”, J. of Recreational Mathematics, vol. 9(1) , 1977

Dieser ist offensichtlich nicht mehr Mitglied der Tippliste, da er im Widerspruch zu vorherigen Bewertungen steht. Allerdings ist er in Hinsicht auf die Anzahl der verbleibenden Rateversuche effektiver. Für diesen Tipp kommen drei mögliche Bewertungen in Frage:

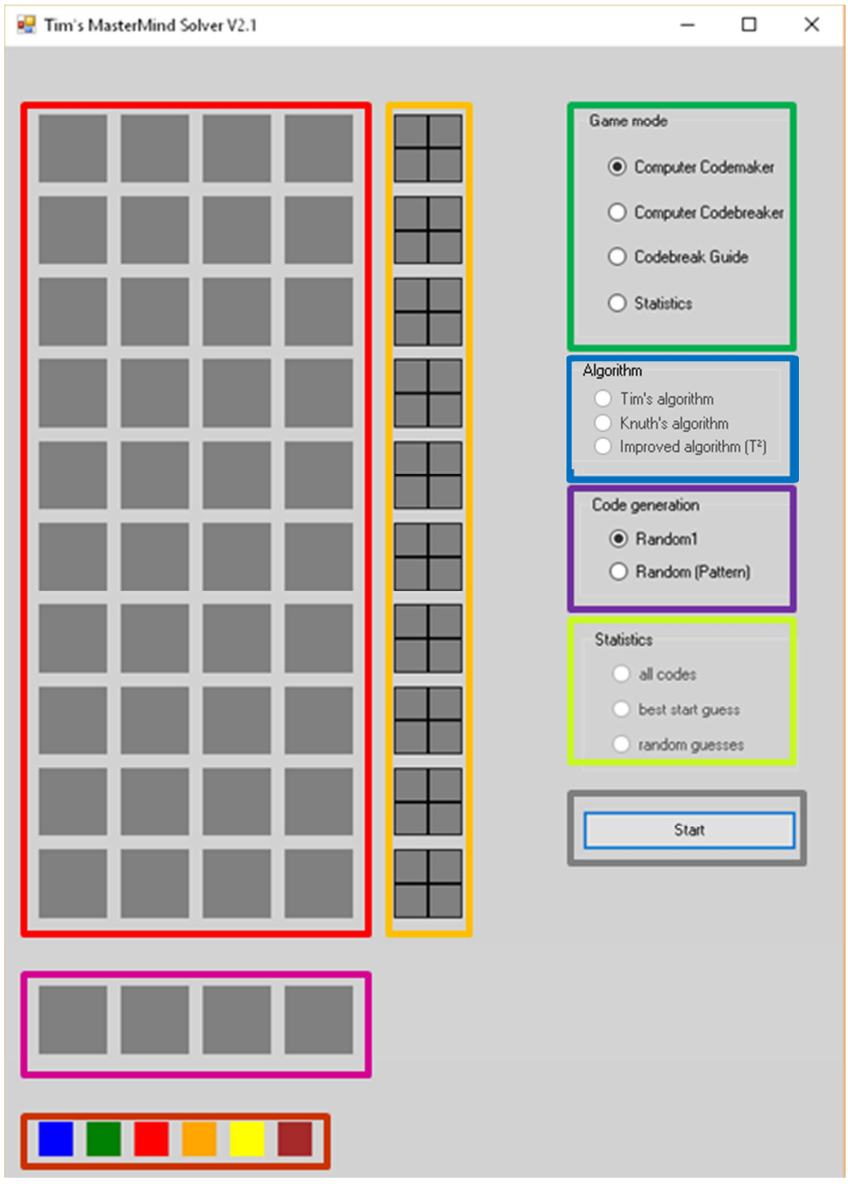
1. **Kein Stecker:** Die 4. Position muss Blau sein.
2. **Ein schwarzer Stecker:** Die 4. Position muss Rot sein.
3. **Ein weißer Stecker:** Die 4. Position muss Grün sein.

Wie man sieht, würde mein Algorithmus im ungünstigsten Fall noch drei weitere Versuche benötigen, während bei Knuth nach nur zwei Versuchen das Spiel garantiert gewonnen wäre.

Aus diesem Beispiel wird deutlich, dass es unter manchen Umständen effektiver sein kann, einen Tipp zu wählen, der gar nicht mehr der gesuchte Code sein kann.

2.2 Funktionalität und Umsetzung

Zu den oben genannten Algorithmen wurde ein Computerprogramm in der Microsoft .NET Sprache C# geschrieben. Das Programm besteht aus 33 Methoden und umfasst über 1500 Zeilen Quelltext. Im folgenden Bild kann man die Benutzeroberfläche erkennen.



Das Programm enthält vier Spielmodi. Man kann diese im **grünen** Kasten auswählen. Im ersten Modus ist der Spieler der Codebreaker und bekommt die Bewertung vom Computer. Im zweiten Modus ist der Computer der Codebreaker und der Spieler der Codemaker. Hier kommen die implementierten Lösungsalgorithmen zur Anwendung. Im Spielmodus „Codebreak Guide“ kann das Programm bei einem realen Spiel behilflich sein, indem es den idealen nächsten Rateversuch nach dem ausgewählten Algorithmus zeigt. Der letzte Spielmodus sind statistische Auswertungen, welche in Kapitel 3 im Detail erwähnt werden.

Mit dem Startbutton bestätigt man einen Spielmodus. Er befindet sich im **grauen** Kasten.

Im **blauen** Kasten kann man den Algorithmus auswählen, mit dem das Spiel gelöst werden soll.

Im **violetten** Kasten kann durch Radiobuttons bestimmt werden, welche Codegenerierung vom Computer im Spielmodus 1 verwendet werden soll. „Random“ steht für einen komplett zufälligen Code mit zufälliger Reihenfolge. „Random Pattern“ soll die eventuelle Strategie eines menschlichen Codemakers nachbilden. Dazu wird zunächst zufällig eines von fünf möglichen Farbmustern ausgewählt. Danach wird dieses Muster zufällig mit Farben gefüllt.

A:A:A:A
A:A:A:B
A:A:B:B
A:A:B:C
A:B:C:D

Im **hellgrünen** Kasten kann man Optionen für statistische Auswertungen auswählen. Im ersten Modus werden alle Codes mit dem ausgewählten Algorithmus gelöst. Als Ergebnis bekommt man die Summe und den Durchschnitt aller benötigten Versuche zurück. Im zweiten Modus kann man sich den besten Starttipp bestimmen lassen, indem das Programm jeden Code mit jedem Starttipp nach dem ausgewählten Algorithmus durchläuft. Im letzten Modus kann untersucht werden, welche die beste Gewinnstrategie in Bezug auf die Codeerstellung für den Codemaker ist.

Im **rot** markierten Kasten befindet sich das Spielfeld. Hier kann der Codebreaker seine Versuche markieren. Der darunterliegende **pinke** Kasten ist das Feld für den gesuchten Code. Daneben befinden sich die Antwortfelder (**orangener** Kasten). Daran gibt der Codemaker die Bewertung ab.

Am unteren Ende des Spielfeldes sind alle sechs möglichen Farben aufgetragen (**brauner** Kasten).

2.3 Programmbeschreibung

Um einen besseren Einblick in das von mir geschriebenem Programm zu erhalten, ist hier ein Quelltextausschnitt aus der Solver-Methode zu sehen. Diese Methode ist das Kernstück des Programms. Da sie natürlich noch um einiges größer ist, wurde sie aus Platzgründen nur auf den wichtigsten Teil reduziert. Bevor die **while-Schleife** beginnt, wurde bereits der vorgegebene Starttipp bewertet und daraufhin die **Tippliste** ein erstes Mal reduziert. Die wichtigsten Erläuterungen der einzelnen Programmzeilen sind in Tabelle 1 zusammengestellt.

```
1   while ((hits[0] != 4) && (tippliste.Count > 1 ))
2   {
3       versuche++;
4       member2 = false;
5
6       if (versuche == zeilen + 1)
7       {
8           ok2.Enabled = false;
9           lblWinLoose.Text = "Verloren!";
10          lblWinLoose.Visible = true;
11          return;
12      }
13      for (int i = tippliste.Count - 1; i >= 0; i--)
14      {
15          hitstemp = new int[2];
16          hitstemp = bewerten(tippliste[i], tipp);
17          if (hits[0] != hitstemp[0] || hits[1] != hitstemp[1])
18              tippliste.RemoveAt(i);
19      }
20      if (rBAlgo2.Checked) // Solver Knuth
21      {
22         int min = 999999;
23
24         foreach (int[] guess in templiste)
25         {
26             int max = 0;
27             foreach (int[] hit in allehits)
28             {
29                 int count = 0;
30                 for (int i = tippliste.Count - 1; i >= 0; i--)
31                 {
32                     hitstemp = bewerten(tippliste[i], guess);
33                     if (hit[0] == hitstemp[0] && hit[1] == hitstemp[1])
34                         count++;
35                 }
36                 if (count > max)
37                     max = count;
38             }
39             member = false;
40
41             for (int i = 0; i < tippliste.Count; i++)
42             {
43                 if (tippliste[i].SequenceEqual(guess))
44                     member = true;
45             }
46             if ((max < min) || ((max == min) && (member==true) && (member2==false)))
47             {
48                 if (member)
49                     member2 = true;
50
51                 min = max;
52                 tipp = new int[4];
53                 tipp = guess;
54             }
55         }
56      }
57      else // Solver Tim
58       tipp = tippliste[0];
```

Zeile	Funktion
1	**while-Schleife:** Der Algorithmus wird ausgeführt, solange keine vier Schwarze als Antwort erhalten wurden und noch mehr als ein Tipp in der **Tippliste** vorhanden ist.
3	Die Rate**versuche** werden um eins erhöht.
6-12	Wird aufgerufen, wenn **versuche** größer als 10 ist. d.h. das Spiel ist verloren.
13-19	Die **Tippliste** wird gemäß Punkt 4 in Abschnitt 2.1 reduziert.
20-56	Tippfindung nach Knuth.
24	**foreach-Schleife:** über alle 1296 Codes (Punkt 1 in Knuths Tippfindung).
27	**foreach-Schleife:** über alle möglichen Bewertungen (Punkt 1a in Knuths Tippfindung)
30	**for-Schleife:** über jedes Element der aktuellen **Tippliste**.
32-37	Bestimmung der Größen der reduzierten **Tipplisten** für jede mögliche Bewertung und Findung des Maximums. (Punkt 1b in Knuths Tippfindung).
41-45	**for-Schleife:** Bestimmung, ob der aktuelle Code Mitglied der **Tippliste** ist. (Mitglieder der **Tippliste** werden bei gleicher Wertigkeit bevorzugt.)
46-53	**if-Anweisung:** Das minimale Maximum der vorgesehenen Tipplistengröße wird bestimmt (Punkt 1c in Knuths Tippfindung). Die zweite Bedingung der **if-Anweisung** verkörpert Punkt 1d in Knuths Tippfindung. Der Code mit dem minimalen Maximum ist der neue Tipp.
57-58	Im Falle von „Tims Algorithmus“ wird lediglich der erste Eintrag der aktuellen **Tippliste** als neuer Tipp verwendet.

Tabelle 1: Erläuterungen zum Programmcode

3 Ergebnisse

In diesem Abschnitt werden mithilfe des Programms statistische Auswertungen zum besseren Verständnis der benutzen Algorithmen durchgeführt.

3.1 Leistungsbeschreibung der Algorithmen

Hier wurden alle 1296 Codes mit dem ausgewählten Algorithmus durchgerechnet. Im Ergebnis wird ein Balkendiagramm erstellt, in dem auf der X-Achse die Anzahl der benötigten Versuche (1 bis 10) und auf der Y-Achse die Anzahl der Codes, die in der jeweiligen Anzahl der Versuche gelöst wurden, aufgetragen sind. Des Weiteren wird die Summe aller benötigen Versuche sowie die durchschnittliche Anzahl von Versuchen pro gelöstem Code zurückgegeben.

3.1.1 Tims Algorithmus

Abb. 1 zeigt, dass die Vielzahl der 1296 Codes in sechs Versuchen gelöst werden. Die höchste Versuchsanzahl liegt bei neun und wird von sechs Codes benötigt. Nur ein Code wird bereits im ersten Versuch erraten. Hierbei handelt es sich offensichtlich um den Starttipp (0, 0 ,0 ,0) des Algorithmus. Die Summe aller benötigten Versuche liegt bei 7471. Die durchschnittliche Versuchsanzahl pro Code beträgt demnach 5.764.

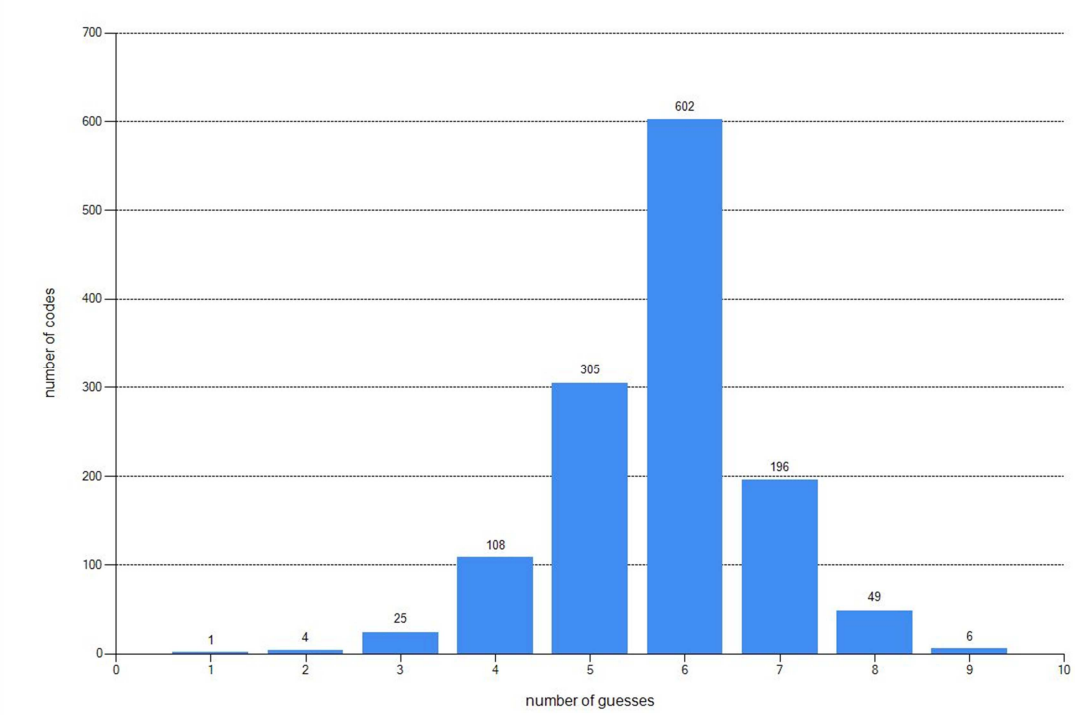


Abb.1: Verteilung der Codes auf die verschiedenen Versuche nach Tims Algorithmus

3.1.2 Knuths Algorithmus

In Abb.2 sind die Ergebnisse für Knuths Algorithmus aufgetragen. Man kann deutlich erkennen, dass die meisten Codes in nur fünf Versuchen gelöst wurden. Dies ist auch gleichzeitig die maximale Anzahl aller benötigten Versuche pro Code. Die Summe aller Versuche beträgt hier 5808 und deren Durchschnitt 4.481.

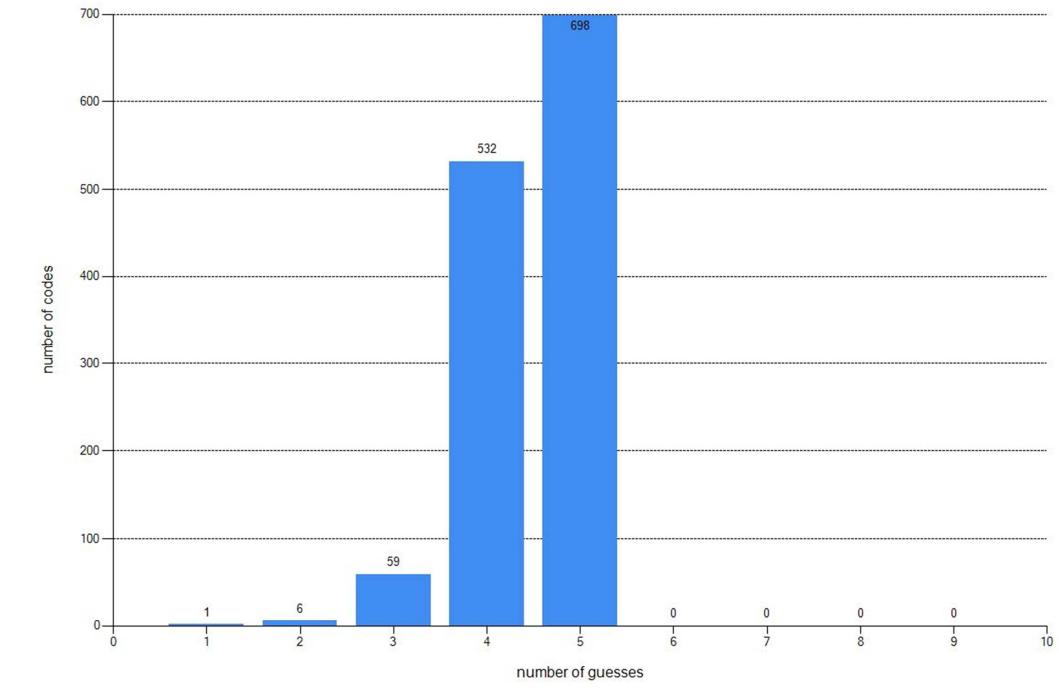


Abb.2: Verteilung der Codes auf die verschiedenen Versuche nach Knuths Algorithmus

3.1.3 Vergleich beider Algorithmen

In diesem Abschnitt werden die beiden Algorithmen verglichen. Tabelle 2 fasst die wesentlichen Merkmale der unterschiedlichen Algorithmen zusammen.

	Tim	**Knuth**
Summe aller Versuche	7471	5808
Max. benötigte Versuche	9	5
Durchschnitt pro Versuch	5.764	4.481
Rechenzeit für alle Codes [s]	9	1614

Tims Algorithmus bleibt mit maximal neun Versuchen unterhalb der zehn Erlaubten. Allerdings benötigt er 1663 Versuche mehr als Knuths Algorithmus und ist deshalb nicht so effizient. Dagegen ist die Rechenzeit bei Knuth mit 1614s ca. 180-mal größer als die bei Tims Algorithmus. Dafür gibt es zwei wesentliche Gründe:

Zum einen wird nicht, wie bei Tims Algorithmus, das erste Element der Tippliste als neuer Tipp gewählt, sondern in einem aufwendigen Verfahren der optimale Tipp ermittelt.

Weiterhin kommen bei Knuths Algorithmus auch Tipps in Frage, die nicht mehr Mitglied der Tippliste sind und somit nicht der gesuchte Code sein können, allerdings am meisten Informationen enthalten. Folgendes Beispiel in Abb. 3 verdeutlicht dies.

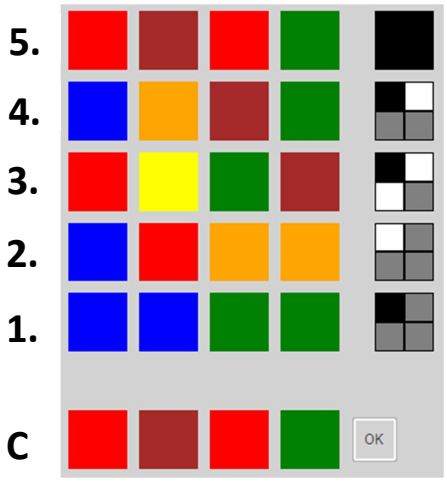


Abb.3: Beispielhafter Verlauf eines Lösungsablaufes nach Knuth

Die ersten drei Tipps reduzieren die Tippliste von ursprünglich 1296 Codes über 256 und 44 Codes auf 7 Codes. Zu Versuch 4 befinden sich noch folgende Elemente in der Tippliste.

Tippliste
(2, 5, 2, 1)
(2, 5, 5, 1)
(3, 4, 5, 1)
(3, 5, 1, 4)
(4, 4, 2, 1)
(5, 3, 1, 4)
(5, 5, 1, 2)

Keiner dieser Tipps kann die Tipplistengröße auf 1 reduzieren und somit den gesuchten Code im nächsten Versuch sicher finden. Deshalb wird für den vierten Versuch der Tipp (0, 3, 5, 1) außerhalb der Tippliste herangezogen. Dieser ist zwar nicht geeignet, das Spiel im nächsten Versuch zu gewinnen, stellt aber sicher, dass der Code in fünf Versuchen erraten ist.

3.2 Verbesserung der Algorithmen

In diesem Kapitel wird die zweite statistische Funktion von meinem Programm benutzt. Hier wird der beste Starttipp bestimmt, indem der jeweilige Algorithmus mit jedem der 1296 möglichen Starttipps jeden Code gemäß Abschnitt 3.1 durchläuft. Danach kann man den besten Starttipp nach zwei Kriterien bestimmen. Zum einen danach, welcher Starttipp in allen durchgerechneten Codes das minimale Maximum der Versuche benötigt hat, und zum anderen danach, welcher Starttipp, für die 1296 Codes insgesamt am wenigsten Versuche gebraucht hat.

3.2.1 Tims Algorithmus

In Abb. 4 ist die Versuchsanzahl in Abhängigkeit vom Starttipp aufgetragen. Darin kann man gut die sechs Spitzen erkennen, welche für die sechs Starttipps mit nur gleichen Farben stehen. Der ursprünglich gewählte Starttipp (0, 0, 0, 0) war also einer der schlechtesten. Die optimalen Starttipps sind (4, 3, 5, 2) und (4, 3, 5, 5), welche beide den ersten Platz belegen.

Kategorie	**Starttipp**	**Anzahl**
kleinste Summe der Gesamtversuche	(4, 3, 5, 2) und (4, 3, 5, 5)	6021
min. Maximum der benötigten Versuche pro Code	(4, 3, 5, 2) und (4, 3, 5, 5)	7

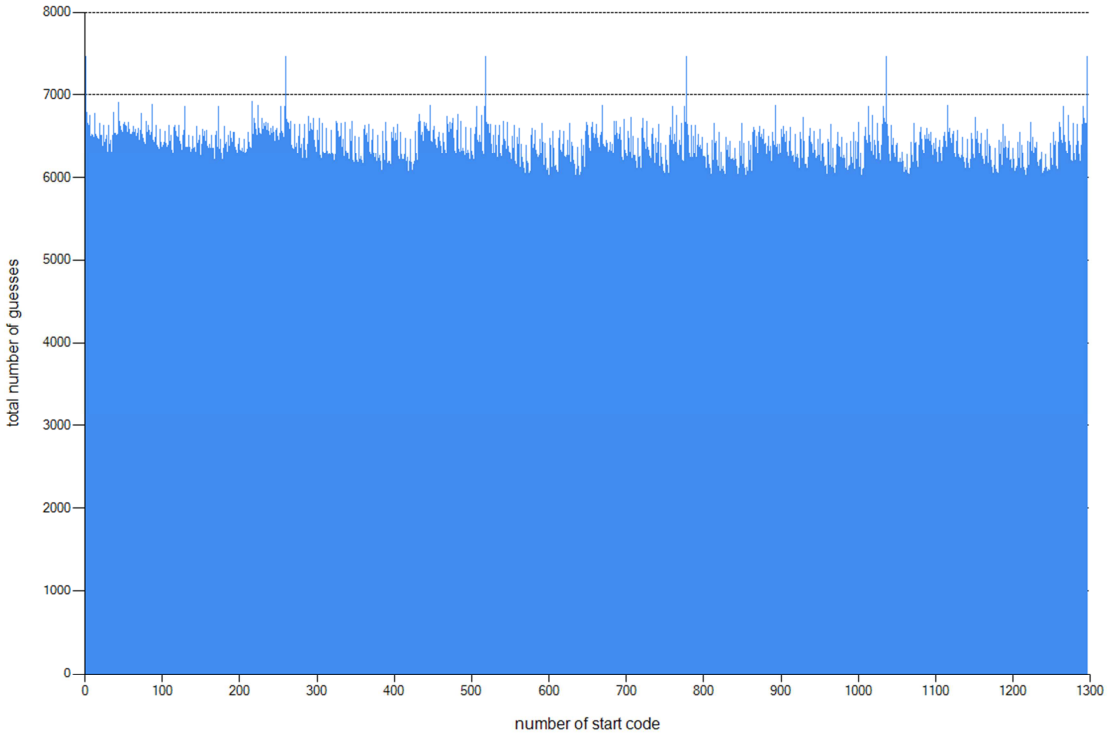


Abb.4: Versuchsanzahl in Abhängigkeit vom Starttipp für Tims Algorithmus

Dieser Starttipp verbessert den Algorithmus enorm. Es lohnt sich daher, die Analyse von Kapitel 3.1 noch einmal zu wiederholen. Die Ergebnisse dafür zeigt Abb. 5.

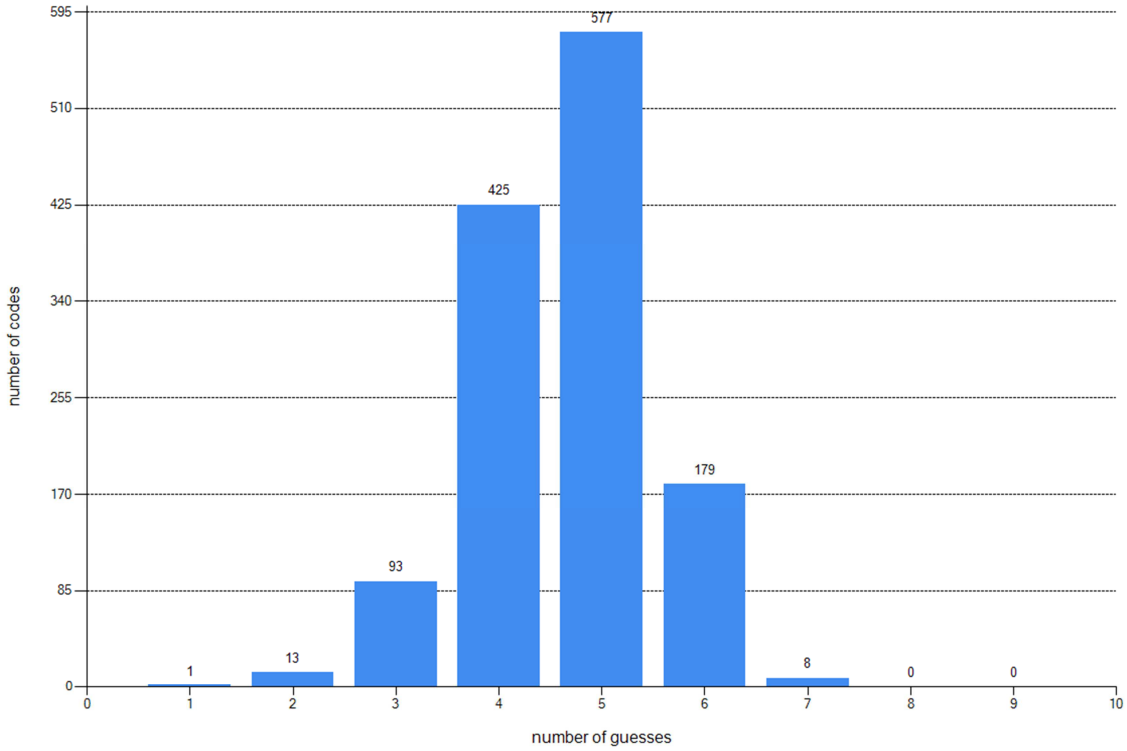


Abb.5: Verteilung der Codes auf die verschiedenen Versuche nach angepasstem Starttipp in Tims Algorithmus.

Nun erkennt man wesentliche Unterschiede zwischen den Diagrammen in Abb.1 und Abb. 5. Während davor die meisten Codes nur in fünf oder sechs Versuchen gelöst werden konnten, ist es jetzt möglich, einen Großteil davon schon in vier bis fünf Versuchen zu lösen, was dazu beiträgt, die durchschnittliche Versuchsanzahl zu verringern. Auch ist das Maximum von neun auf sieben Versuche abgesunken, wobei nur für acht Codes sieben Versuche gebraucht werden.

3.2.2 Knuths Algorithmus

Auch hier kann man wieder die sechs Spitzen erkennen (Abb.6). Ein weiterer Unterschied zwischen Abb. 6 und dem Diagramm in 3.2.1 (Abb. 4) liegt darin, dass die Schwankungen in den Versuchsanzahlen schwächer ausfallen. Die Abhängigkeit vom Starttipp ist also nicht so stark.

Für die beiden Bewertungskategorien gibt es nun auch zwei verschiedene Codes:

Bewertungskategorie	**Starttipp**	**Anzahl**
kleinste Summe der Gesamtversuche	(0, 5, 4, 2) und (4, 0, 1, 5)	5783
min. Maximum der benötigten Versuche pro Code	(0, 1, 1, 0)	5

Zwar haben die Codes (0, 5, 4, 2) und (4, 0, 1, 5) die geringste Anzahl an Gesamtversuchen, allerdings benötigen sie bis zu sechs Versuche pro Code. Der Starttipp (0, 1, 1, 0) hingegen löst alle Codes in nur fünf Versuchen, braucht aber dafür mit 5795 eine größere Summe an Gesamtversuchen. Dies liegt aber immer noch unterhalb des Ergebnisses vom originalen Algorithmus.

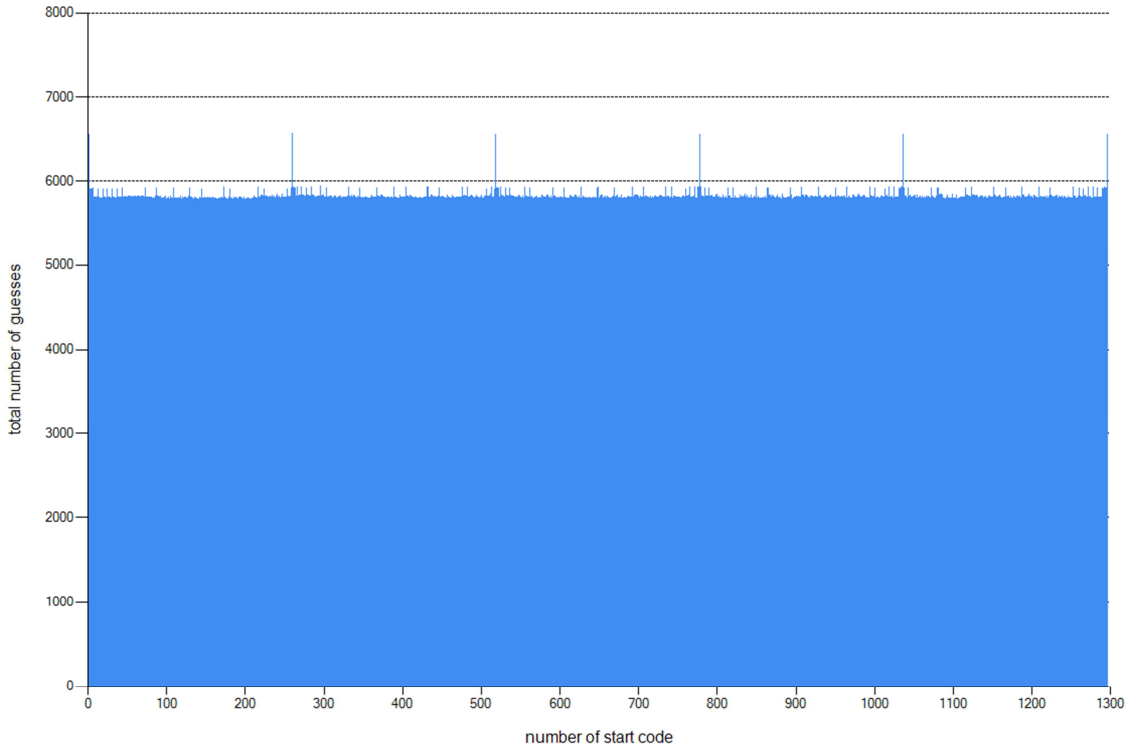


Abb.6: Versuchsanzahl in Abhängigkeit des Starttipps für Knuths Algorithmus

Die geänderte Statistik mit den neuen Starttipps sieht wie folgt aus:

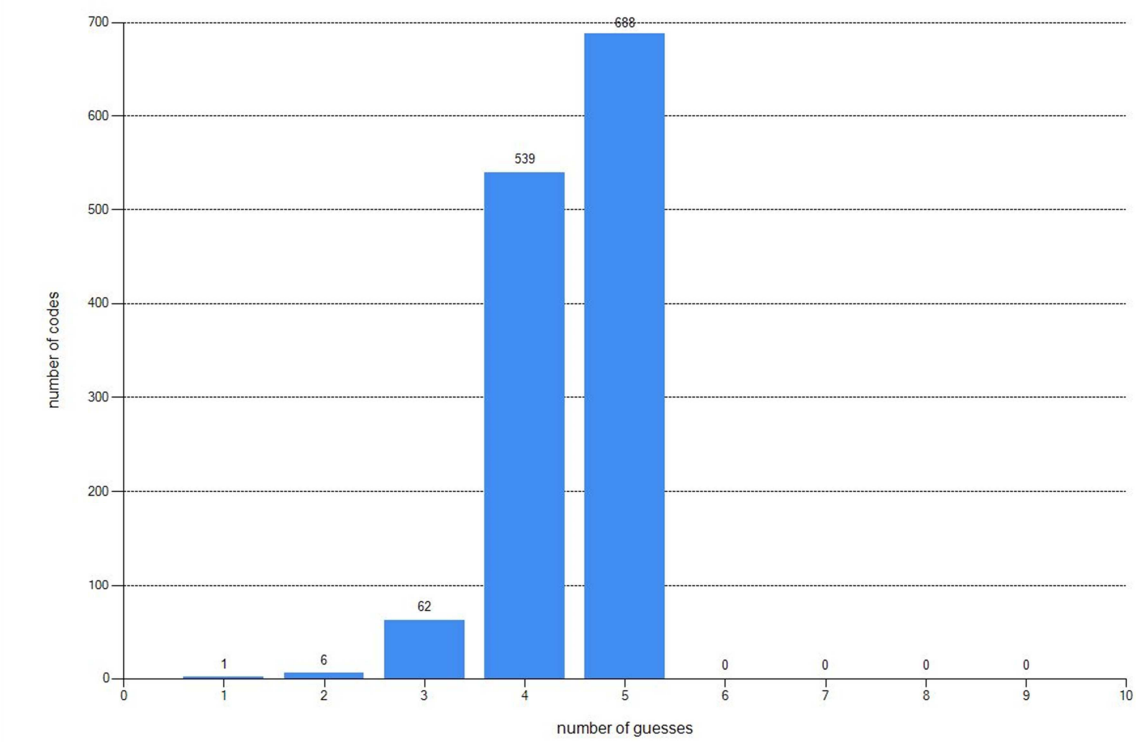


Abb.7: Verteilung der Codes auf die verschiedenen Versuche nach angepasstem Starttipp (0, 1, 1, 0) in Knuths Algorithmus.

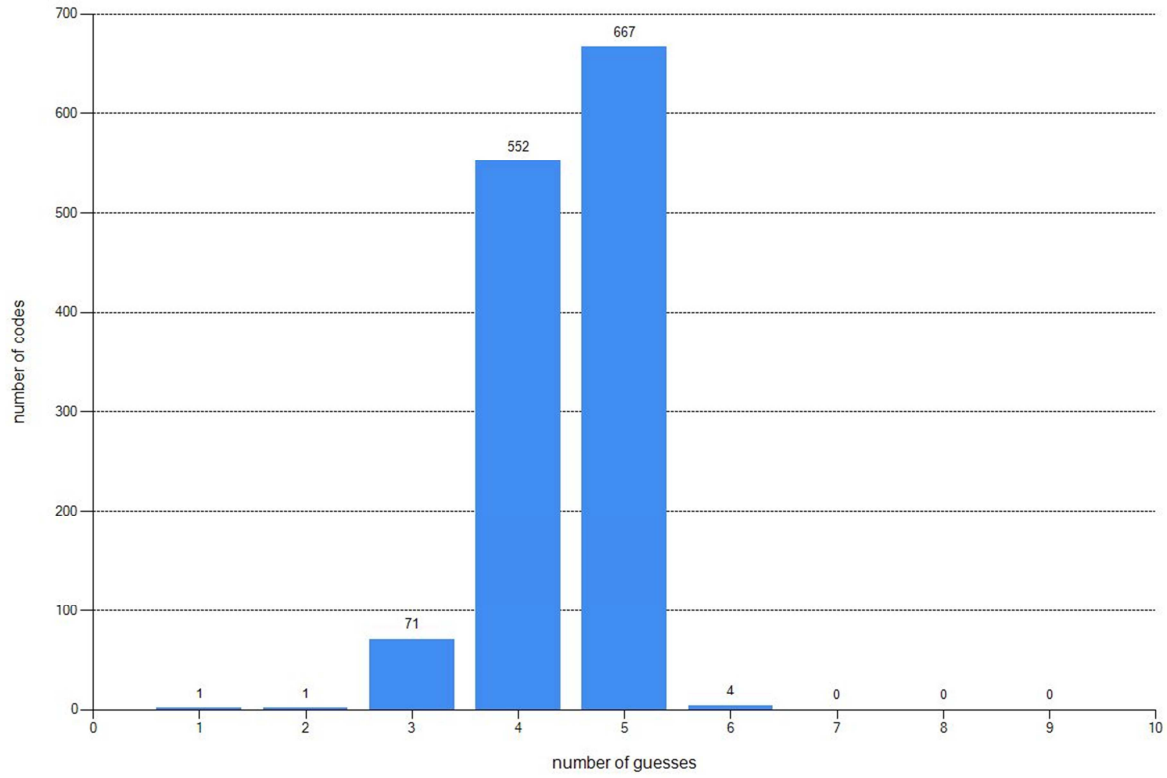


Abb.8: Verteilung der Codes auf die verschiedenen Versuche nach angepasstem Starttipp (0, 5, 4, 2) bzw. (4, 0, 1, 5) in Knuths Algorithmus.

3.2.3 Verbesserter Algorithmus T^2

Es wurde eine Analyse der Schritte in Knuths Algorithmus vorgenommen. Die folgende Tabelle zeigt exemplarisch die Wertigkeiten aller 1296 in Frage kommenden Tipps für die erste Spielrunde, in der die Anzahl der noch möglichen Codes ebenfalls 1296 beträgt. Aus Platzgründen sind jedoch nur die Auswertungen für sieben Tipps dargestellt.

	mögliche Bewertungen														**Wertigkeit**	
mögl. Tipps (7 aus 1296)	**00**	**01**	**02**	**03**	**04**	**10**	**11**	**12**	**13**	**20**	**21**	**22**	**30**	**40**	Max.	σ
(0, 0, 0, 0)	625	0	0	0	0	500	0	0	0	150	0	0	20	1	625	204.5
(0, 0, 0, 1)	256	308	61	0	0	317	156	27	0	123	24	3	20	1	317	119.5
(0, 0, 1, 1)	256	256	96	16	1	256	208	36	0	114	32	4	20	1	256	105.6
(0, 0, 1, 2)	81	276	222	44	2	182	230	84	4	105	40	5	20	1	276	96.1
(2, 4, 1, 3)	16	152	312	136	9	108	252	132	8	96	48	6	20	1	312	97.6
(3, 0, 3, 3)	256	308	61	0	0	317	156	27	0	123	24	3	20	1	317	119.5
(5, 5, 5, 5)	625	0	0	0	0	500	0	0	0	150	0	0	20	1	625	204.5

Der Algorithmus von Knuth wählt den Code (0, 0, 1, 1) als neuen Tipp aus, da er mit 256 das geringste Maximum der größten Partition hat. In der letzten Spalte ist zudem die Standardabweichung σ der einzelnen Partitionen aufgetragen, die sich wie folgt berechnet.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

Man kann erkennen, dass die Standardabweichung beim optimalen Tipp nach Knuth (0, 0, 1, 1) größer ist als beim Code (0, 0, 1, 2). Aus dieser Beobachtung heraus wurde ein neuer Algorithmus implementiert, welcher den optimalen Tipp nicht durch das geringste Maximum findet, sondern durch die kleinste Standardabweichung auswählt. Es ist zu erwarten, dass möglichst gleichverteilte Partitionen, was durch eine geringe Standardabweichung ausdrückt wird, eine effiziente Lösungsstrategie bieten.

In Abbildung 9 kann man erkennen, dass auch dieser Algorithmus mit dem Starttipp (0, 0, 1, 1), genau wie bei Knuth, alle Codes in maximal fünf Versuchen löst. T^2 hat zum Lösen aller 1296 Codes insgesamt 5766 und damit 42 Versuche weniger als Knuth gebraucht. Im Schnitt sind dies 4.449 benötigte Versuche pro Code. Wenn man darauf verzichtet, alle Codes in fünf Versuchen zu lösen, liefert der Starttipp (0, 4, 5, 0) mit 5687 sogar einen noch kleineren Wert. Der Durchschnitt liegt hier dann bei 4.388.

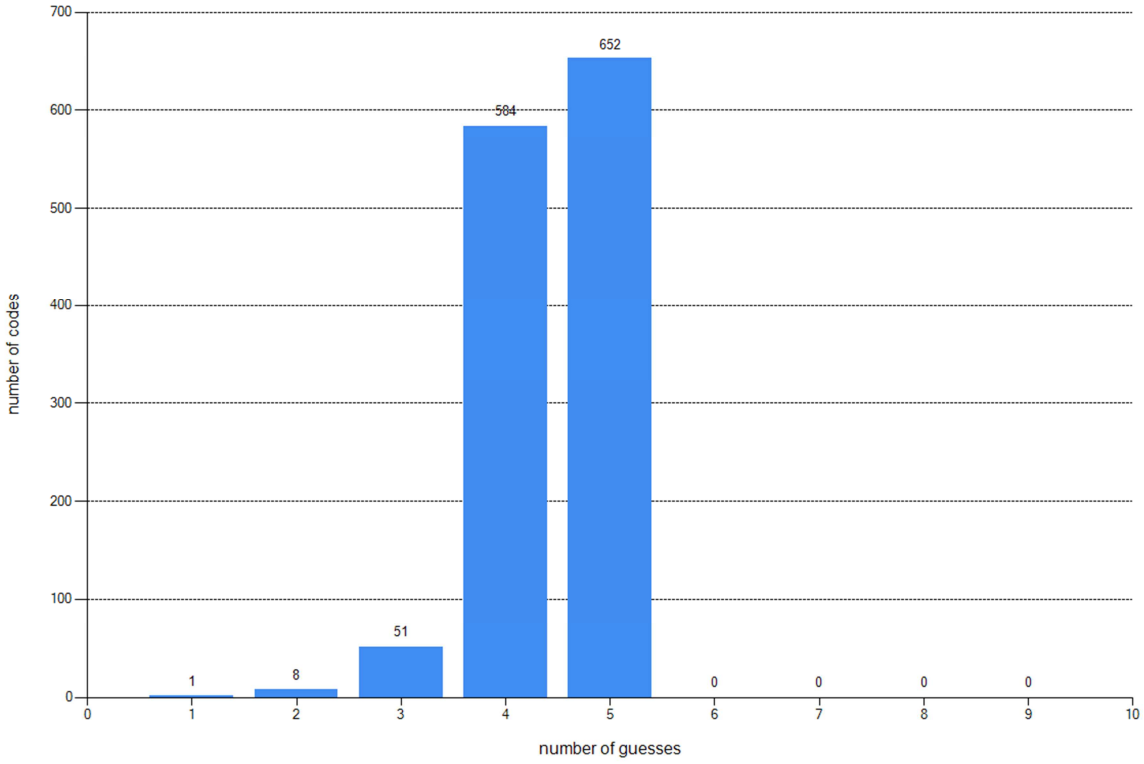


Abb.9: Verteilung der Codes auf die verschiedenen Versuche nach dem T^2 Algorithmus.

3.2.4 Vergleich aller Algorithmen

In der folgenden Tabelle sind die Kennzahlen der verschieden Algorithmen aufgetragen. T^2 ist unter den behandelten Ansätzen von seiner Leistungsfähigkeit der Beste.

	Tim (original)	Tim (verbessert)	Knuth (original)	Knuth (verbessert)		T^2	
Starttipp	(0, 0, 0, 0)	(4, 3, 5, 5)	(0, 0, 1, 1)	(0, 1, 1, 0)	(0, 5, 4, 2)	(0, 0, 1, 1)	(0, 4, 5, 0)
Summe aller Versuche	7471	6021	5808	5795	5783	5766	5687
max. Versuche	9	7	5	5	6	5	6
Durchschnitt Versuche/Code	5.764	4.464	4.481	4.471	4.462	4.449	4.388
Rechenzeit für alle Codes [s]	9	7,25	1614	1610	1607	1576	1554

4 Zusammenfassung

Für das Spiel MasterMind wurden verschiedene Lösungsstrategien entworfen und in einem Computerprogramm umgesetzt. Zudem wurde ein Algorithmus aus einer Veröffentlichung untersucht. Durch statistische Analysen konnten die eigenen und der veröffentlichte Algorithmus verbessert werden.

Die einzelnen Algorithmen wurden in folgenden Kategorien untersucht:

- Summe aller benötigten Versuche für alle Codes
- Maximum der benötigten Versuche für einen Code
- Durchschnittliche Anzahl der benötigten Versuche pro Code
- Rechenzeit, um alle 1296 Codes zu lösen

Durch systematische Untersuchungen wurde zu jedem Algorithmus der optimale Starttipp ermittelt, so dass er noch weiter verbessert werden konnte. Die obige Tabelle in Abschnitt 3.2.4 fasst alle wesentlichen Ergebnisse zusammen. Die größte Verbesserung in seiner Effizienz erfuhr „Tims Algorithmus“ mit einer Reduktion von 7471 auf 6021 Versuche, um alle 1296 Codes zu lösen. Auch in dem aus der Literatur entnommenen Algorithmus von Knuth konnte eine leichte Verbesserung um 13 bzw. 25 Versuche erzielt werden, wobei nur die erste Reduktion weiterhin eine Lösung aller Codes innerhalb von 5 Versuchen gewährleistet.

Als effizientester Algorithmus hat sich T^2 erwiesen. Im Vergleich zum Algorithmus von Knuth löst T^2 alle 1296 Codes in nur 5766 Versuchen und benötigt damit 29 Versuche weniger als die bereits verbesserte Version des Algorithmus‘ von Knuth. Nimmt man in Kauf, dass 3 der 1296 Codes in 6 Versuchen gelöst werden, reduziert sich die Anzahl aller benötigten Versuche um weitere 79 Versuche.