NIM-SPIELE

EINE MATHEMATISCHE ANALYSE MIT COMPUTERCODE

Tony Hürlimann

Working Paper

Oktober 1999

INSTITUT D'INFORMATIQUE, UNIVERSITE DE FRIBOURG

INSTITUT FÜR INFORMATIK DER UNIVERSITÄT FREIBURG

Institute of Informatics, University of Fribourg

site Regina Mundi, rue de Faucigny 2, CH-1700 Fribourg / Switzerland

tony.huerlimann@unifr.ch

phone: ++41 26 300 2845 fax: ++41 26 300 9726

This research is supported by the Swiss National Science Foundation and financed by the project no. 12-55989.98.

NIM-Spiele
Eine mathematische Analyse
mit Computercode

Tony Hürlimann, PD. Dr. lic. rer. pol.

Keywords: Zweipersonen-Spiele, NIM-Spiele, azyklische Graphen.

Zusammenfassung:
Dieser Artikel gibt eine vollständige, mathematische Analyse gewisser Zwei-Personen-Spiele, die auch unter dem Namen *NIM-Spiele* bekannt sind. Auf Grund der Analyse wird ein Algorythmus und Programmcode präsentiert, welcher das Spiel immer unfehlbar gewinnt, falls die Ausgangs-Konfiguration eine Nicht-gewinnkonfiguration ist oder der Gegner auch nur einen Fehler begeht. Verschiedene Varianten des Spiels werden vorgestellt und alle werden auf dieselbe mathematische Theorie von azyklischen Graphen zurückgeführt.

1 Einleitung

Es gibt eine Vielzahl von Zweipersonen-Spiele, bei denen abwechslungsweise auf einem *Spielbrett* durch bestimmte *Regeln* ein *Zug* gemacht werden muss, und zwar solange bis kein solcher mehr möglich ist oder eine gewisse *Konfiguration* erreicht ist und damit ein Spieler zum *Gewinner* erklärt wird. Das Schachspiel, das Mühlenspiel, u.a. gehören in diese Klasse von Spielen.
Betrachten wir ein ganz einfaches Spiel (wir nennen es: das 16-Objekte-Spiel): 16 Objekte liegen auf einem Tisch. Zwei Spieler dürfen abwechslungsweise 1, 2, 3 oder 4 der verbleibenden Objekte entfernen. Der Gewinner ist derjenige Spieler, welcher die letzten Objekte (also höchstens 4) entfernt.

Das 16-Objekte-Spiel besitzt folgende Elemente:

1 Das "*Spielbrett*": die 16 auf dem Tisch liegenden Objekte
2 *Zug*: jeder Spieler abwechslungsweise darf Objekte entfernen
3 *Regeln*: Bei jedem Zug höchstens 4 und mindestens 1 Objekt
4 *Konfiguration*: Anzahl verbleibende Objekte
5 *Gewinner*: der Spieler, welcher die letzten Objekte abräumt

Eine ganze Reihe von Zweipersonen-Spiele besitzt dieselben Elemente. Zum Beispiel das Schachspiel:

1 Das "Spielbrett": das Schachspielbrett mit den Figuren
2 Zug: eine Figur verschieben
3 Regeln: die Schachregeln
4 Konfiguration: Eine Konfiguration der Schachfiguren
5 Gewinner: der Spieler, welcher als erster keinen Zug mehr tun kann

In diesem Artikel soll im zweiten Abschnitt eine in der Graphentheorie wohlbekannte Theorie [Tucker 1995] kurz vorgestellt werden, die geeignet ist, all diese Spiele zu analysieren. Im dritten Abschnitt wird das 16-Objekte-Spiel aus dem Lichte dieser Theorie beispielhaft anylysiert. Im vierten Abschnitt wird eine Variante des bekannten 16-Streichholz-Spiels gegeben und eine Gewinnstrategie angegeben. Der fünfte Abschnitt führt eine zweite Spielvariante ein, welche zu einer allgemeinen einfachen Theorie der NIM-Spiele führt.

2 Die Theorie

Gegeben sei ein Spiel, bei dem zwei Spieler abwechsungsweise die Spiel-Konfiguration nach vorgegebenen Regeln ändern können, d.h. eine Zug tun können, und zwar solange bis kein solcher mehr möglich ist. Wir nehmen an, dass bei jedem Zug nur eine finite Anzahl Veränderungen möglich sei, und dass eine finite Anzahl Ausgangs-Konfigurationen gegeben sei. Die Bedingung, dass das Spiel nach endlich vielen Zügen enden muss, ist zwar für die Theorie nicht wichtig, im praktischen Spielverlauf sind aber Spiele mit unendlich vielen Zügen, die durch eine zyklische Wiederholung einer bestimmten Zugsreihenfolge entstehen können, uninteressant.

Denn dadurch kann kein Gewinner ermittelt werden. Alle praktische Spiele müssen daher “progressiv” sein, d.h. jeder Zug muss das Spiel näher an das Ende des Spiels bringen.

Ziel einer Theorie ist, die optimale Strategie eines Spielers herauszufinden. Konkret: Mit welchem Zug muss ein Spieler jeweils antworten, damit er das Spiel eventuell gewinnt. Gibt es überhaupt eine solche Strategie? Gegeben eine optimale Strategie, ist dann die Ausgangs-Konfiguration entscheidend, um das Spiel immer zu gewinnen oder was sind die Bedingungen?

Man kann solche Zweipersonenspiele als gerichtete Graphen modellieren. Jede Spiel-Konfiguration wird als *Knoten* des Graphen, und jeder Zug, der von einer Konfiguration zur einer andern führt, wird als *gerichtete Kante (*auch *Bogen* genannt*)* abgebildet. Man beachte, dass “progressive” Spiele, wie sie oben definiert wurden, zu azyklischen Graphen führen, da des ausgehend von einer Konfiguration keine Möglichkeit gibt, durch eine endliche Anzahl von Zügen wieder in diesselbe Konfiguration überzugehen. Das bedeutet aber, dass es Knoten im entsprechenden Graphen gibt, deren Anzahl weggehende Bögen Null ist, welche die End-Konfigurationen definieren. Die End-Konfigurationen sind dann die Gewinn-Konfigurationen eines Spielers, weil kein Zug mehr möglich ist. Umgekehrt entsprechen die Ausgangs-Konfigurationen (oft ist nur eine gegeben) im Graphen Knoten, deren Anzahl eintreffende Bögen Null ist. Man kann sich so einen Graphen, wie in Abbildung 1 gezeichnet, vorstellen.

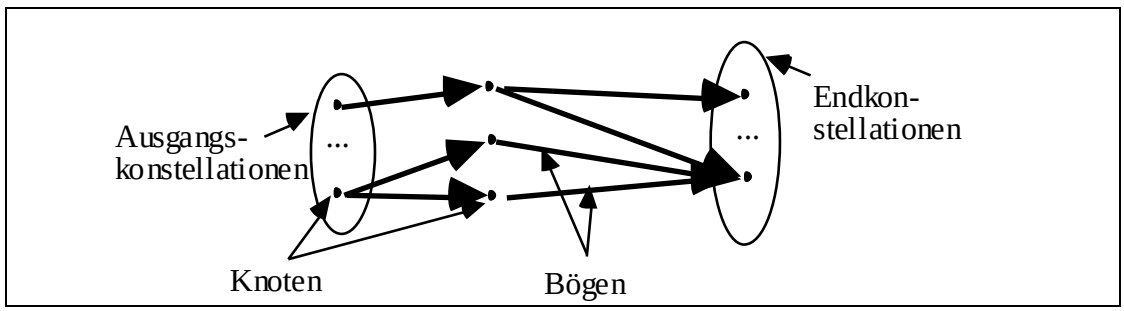

Abbildung 1

Wir nennen die Knoten, die die Untermenge der Ausgangs-Konfigurationen abbilden, die *Ausgangsknoten*. Die Knoten, welche die Untermenge der End-Konfigurationen abbilden, heissen *Gewinnknoten*. Ein Spieler,der den letzten Zug tun kann, im Graph also ein Bogen, die in einen Gewinnknoten hineingeht, gewinnt das Spiel. Jeder *Pfad,* ausgehend von einen Ausgangsknoten zu einem Gewinnknoten repräsentiert einen *Spielverlauf*. Abwechslungsweise jeder zweite Bogen in einem solchen Pfad, bildet genau einen Zug ein und desselben Spielers ab (Abbildung 2).

Der gesamte Graph fasst also die Gesamtheit aller Spielverläufe zusammen. Eine Gewinnstrategie ist eine Regel, welche einem Spieler sagt, welcher Zug (Bogen im Graph) ausgehend von einer Konfiguration (Knoten im Graph) er verfolgen muss, um das Spiel eventuell zu gewinnen, d.h. als letzter Zug zu einer EndKonfiguration (Gewinnknoten) überzugehen. Natürlich kann nur ein Spieler eine Gewinnstrategie haben.

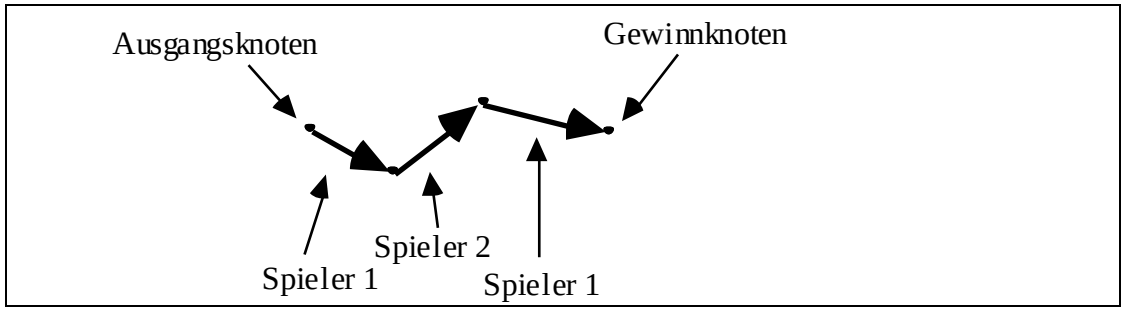

Abbildung 2: Ein Spielverlauf

Es scheint offensichtlich, dass alle Gewinnknoten untereinander nicht durch Bögen verbunden sind, da es sonst möglich wäre von einem Gewinnknoten in einen andern Gewinnknoten überzugehen. Ausserdem sind alle Knoten, die durch einen Bogen mit einem Gewinnknoten verbunden sind, *Verlierknoten*, in dem Sinne, dass ein Spieler der in einen solchen Knoten hineingeht, das Spiel verlieren kann, da der nächste Zug des andern Spieler in einen Gewinnknoten übergehen kann – aber nicht muss, da Verlierknoten unter Umständen untereinander durch einen Bogen verbunden sind.

Betrachten wir jetzt alle Verlierknoten. Sie sind also alle Knoten im Graphen, die mit mindestens einem Gewinnknoten durch einen Bogen verbunden sind. Wenn wir weiter einen Zug zurückgehen, dann erhalten wir die Menge aller Knoten (nennen wir sie X), die durch einen Bogen, mit einem Verlierknoten als Endpunkt des Bogens verbunden sind. Diese Menge X kann in zwei Untermengen partitioniert werden: (1) diejenigen, von denen aus alle ausgehenden Bögen zu einem Verlierknoten führen, und (2) diejenigen, die zwar Bögen zu Verlierknoten haben, aber daneben auch noch andere ausgehende Bögen, die zu Knoten in der Menge X zeigen. Wir nehmen die erstere Menge die *Vorgewinnknoten*. Es ist klar, dass ein Spieler, der auf eine Vorgewinnknoten trifft, das Spiel gewinnen kann. Beim nächsten Zug muss der andere Spieler zu einen Verlierknoten kommen, während der erste Spieler beim übernächsten Zug in eine Gewinnknoten hineinläuft.
Wir erhalten das Schema in Abbildung 3.

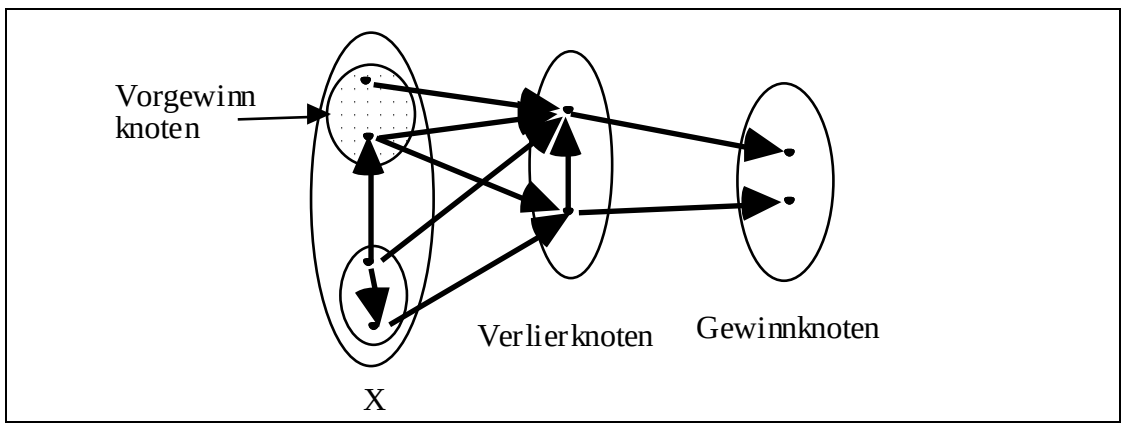

Abbildung 3

Eine Theorie der Gewinnstrategie beruht auf dieser rekursiven Ausdehnung des

Graphen, indem immer mehr Knoten eingezogen werden und diese in weitere Vorgewinnknoten und andere partitioniert werden, bis alle Knoten partitioniert sind. Wir nennen die Vereinigung aller Vorgewinnknoten (zusammen mit den Gewinnknoten) die *guten Knoten*, und die andern die *schlechten Knoten*. Alle Knoten des Graphen werden somit in gute und schlechte Knoten eingeteilt. Die optimale Strategie für einen Spieler A muss also sein, wenn möglich immer einen guten Knoten zu erreichen, da der Spieler B dann gezwungen ist, beim nächsten Zug in einen schlechten Knoten hineinzulaufen, hernach wird es für den Spieler A immer möglich sein, beim übernächsten Zug wieder einen guten Knoten zu erreichen. Dies kann dann bis zum Spielende wiederholt werden und Spieler A gewinnt das Spiel.

Das Konzept der guten Knoten soll nun mit einer Definition formalisiert werden.

***Defintion**: Der Kernel eines gerichteten Graphen ist die Menge der Knoten so,*
(1) dass kein Bogen zwischen den Kernelknoten existiert,
(2) dass es einen Bogen von jedem Nicht-Kernelknoten zu einem Kernelknoten gibt.

Theorem: Wenn ein azyklischer Graph einen Kernel hat, dann ist die Gewinnstrategie für den ersten Spieler bei jedem Zug zu einem Kernelknoten zu gelangen. Wenn der Ausgangsknoten im Kernel ist, so kann der zweite Spieler diese Gewinnstrategie anwenden.
Beweis: Die Gewinnknoten sind im Kernel, denn die Anzahl weggehender Bögen ist Null. Gemäss der Eigenschaft (1) der Kerneldefinition, muss der zweite Spieler aus dem Kernel hinausgehen, wenn der erste Spieler zuvor in den Kernel gelangte; gemäss der Eigenschaft (2) kann der erste Spieler immer in den Kernel ausgehend von einem beliebigen Nicht-Kernelknoten gelangen. Das Spiel schreitet also so fort, dass der erste Spieler immer in den Kernel gelangt, während der zweite gezwungen ist zu einem Nicht-Kernelknoten zu gelangen. Da die Gewinnknoten im Kernel sind muss der erste Spieler das Spiel gewinnen.

Q.E.D.

Theorem: Jeder azyklische Graph besitzt einen nicht-leeren, eindeutigen Kernel.
Beweis: Sei

$l(x) = 0 \Leftrightarrow s(x) = \varnothing$ und $L_0 = \{x | l(x) = 0\}$

$l(x) = k \Leftrightarrow (x \notin L_{k-1} \wedge s(x) \subseteq L_k)$ und $L_k = L_{k-1} \cup \{x | l(x) = k\}$, mit $\forall k{:}k > 0$,

wobei $s(x)$ definiert ist als die Menge der Nachfolgerknoten von *x*. Sei weiter K_k die Menge der Kernelknoten in L_k. Dann ist L_0 die Menge der Gewinnknoten. Da alle Gewinnknoten im Kernel sind, muss gelten $K_0 = L_0$. Durch die Induktionhypothese nehmen wir an, dass für alle $n>1$ K_{n-1} der wohl-definierte Kernel für L_{n-1} ist. Dann müssen wir zeigen, dass es möglich ist durch Hinzufügen von Knoten aus L_n einen wohl-definierten Kernel K_n für L_n zu finden. Wenn ein Knoten $x \in L_n - L_{n-1}$ zu keinem Knoten in K_{n-1} adjazent ist, dann muss dieser Knoten in K_n sein. Andererseits, wenn ein Knoten $x \in L_n - L_{n-1}$ adjazent zu einen Kernelknoten in K_{n-1} ist, dann kann *x* nicht in K_n sein. Das heisst aber, dass $K_n = K_{n-1} \cup (\{x | l(x) = n \wedge s(x) \cap K_{n-1}\})$ wohl-definiert ist und in L_n liegt. Daraus folgt durch Induktion, dass der Graph einen wohl-geformten Kernel hat. Zudem ist er nicht leer, da L_0 nicht-leer ist.

Q.E.D.

3 Das 16-Objekte-Spiel

Die Theorie soll jetzt anhand eines einfachen Spiels illustriert werden.
Sechszehn Objekte (z.B. Steichhölzer) sind auf einem Tisch plaziert. Zwei Spieler dürfen abwechslungsweise 1, 2, 3 oder 4 Streichhölzer wegnehmen. Gewinner ist derjenige, der die letzten Steichhölzer wegnimmt.

Lösung: Man zeichnet einen Graphen mit 17 Knoten, für jede Konfiguration. Diese Konfigurationen sind: (1) '16 Objekte auf dem Tisch', (2) '15 Objekte auf dem Tisch', usw., bis (17) 'Null Objekte auf dem Tisch'. Eine Kante (Bogen) wird gezogen, wenn von einem Zustand (einer Konfiguration) in den andern übergegangen werden darf. Laut Spielregeln darf 1, 2, 3 oder 4 Streichhölzer entfernt werden, das heisst: z. B. vom Zustand '16 Objekte sind auf dem Tisch', darf übergegangen werden in den Zustand '15 Objekte sind auf dem Tisch' oder '14 Objekte sind auf dem Tisch', '13 Objekte sind auf dem Tisch' oder '12 Objekte sind auf dem Tisch'. Daher ziehen wir einen Bogen von 16 zu 15 , von 16 zu 14, von 16 zu 13 und von 16 zu 12. Dasselbe tun wir mit Knoten 15, dann 14 usw. So entsteht der abgebildete, azyklische Graph (Abbildung 4).

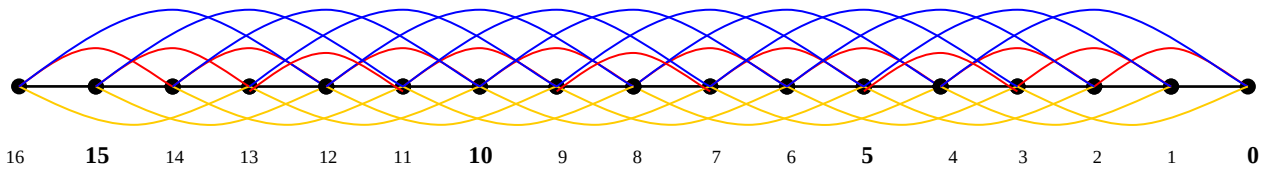

Abbildung 4

Ein Spielverlauf kann nun durch eine sequentielle Bogenmenge beschrieben werden, wie z. B. im nächsten Graph (Abbildung 5), bei dem der erste Spieler 4 Streichhölzer, der zweite 1, der erste wieder 3, der zweite 2, der erste 4 (selber Schuld!), und schliesslich der zweite Spieler 2 entfernt. Der zweite Spieler hat gewonnen.

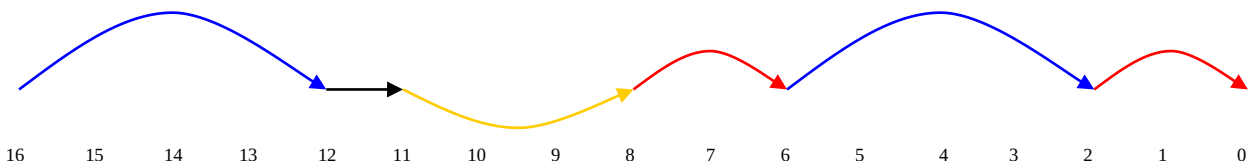

Abbildung 5

Das Spiel gewinnt derjenige, welcher zum Knoten Null gelangt. Dies ist der einzige *Gewinnknoten*. Ein Knoten, von dem aus ein Gewinnknoten erreicht werden kann, muss demnach ein *Verlierknoten* sein. Also sind Knoten 4, 3, 2, 1 alle Verlierknoten.

Das heisst, wer zu diesen gelangt, muss damit rechnen zu verlieren, wenn der andere Spieler entsprechend 4, 3, 2 oder 1 Steichhölzer wegnimmt. Wie steht es mit Knoten 5? Wer dorthin gelangt, kann gewinnen. Wieso? Weil der andere Spieler dann 1, 2, 3 oder 4 Streichhölzer entfernen muss, und damit unweigerlich auf den Knoten 4, 3, 2 oder 1 – also auf einem Verlierknoten – landet. Knoten 5 ist also ein Vorgewinnknoten. Knoten 6, 7, 8 und 9 sind natürlich auch wieder schlechte Knoten und Knoten 10 ist ein guter Knoten. Wieso? Wenn ein Spieler 10 erreicht (d.h. nach seinem Zug 10 Streichhölzer auf dem Tisch zurücklässt), kann der andere Spieler gar nicht anders als zum Knoten 9, 8, 7 oder 6 zu gelangen (d.h. er muss 9, 8, 7 oder 6 Streichhölzer zurücklassen, worauf der erste Spieler seinerseits dann 4, 3, 2 oder 1 Streichholz entfernt, um nur noch 5 übrig zu lassen.
Wir können also die Knoten in gute und schlechte Knoten einteilen. 15, 10, 5 und 0 sind gute Knoten, die andern sind schlechte Knoten. Die Strategie, um das Spiel zu gewinnen, muss demnach sein: “Versuche bei jedem Zug einen guten Knoten zu erreichen.”
Der Kernel dieser Graphen ist demnach die Knotenmenge: {0, 5, 10, 15}.
Man beachte im Beispiel, dass – gemäss der Kerneldefinition, (1) keine der guten Knoten miteinander verbunden sind: {5, 10, 5, 0} sind alle nicht durch einen Bogen verbunden); und (2) jeder schlechte Knoten {16, 14, 13,...,1} hingegen ist mit mindestens einem guten Knoten verbunden.

Die Gewinnstrategie für dieses Spiel muss daher für den ersten Spieler sein, (1) beim ersten Zug in den Kernel gelangen, indem er ein Streichholz entfernt, (2) immer im Kernel zu bleiben.

4 Das 16-Streichholz-Spiel (erste Variante)

Problem: 16 Streichhölzer liegen in vier Reihen zu eins, drei, fünf und sieben, wie folgt, vor:

I
III
IIIII
IIIIIII

Jeder Spieler darf pro Zug aus einer Reihe beliebig viele Hölzchen – aber mindestens eins – wegnehmen, aber nicht aus zwei oder mehr Reihen. Wer das letzte Hölzchen nehmen muss, hat verloren.

Lösung: Dieses Spiel hat genau dieselbe mathematische Struktur, wie das zuvor beschriebene, einfache Streichholzproblem. Der Graph ist ein azyklischer gerichteter Graph, daraus folgt (wie gezeigt), dass ein Kernel existiert. Also gibt es gute Knoten, die der Spieler anpeilen muss. Wenn er sich also an diese guten Knoten hält, muss er das Spiel immer gewinnen. Das Problem ist allerdings jetzt, dass der Graph nicht mehr gezeichnet werden kann, da er 384 Knoten enthält und dadurch unübersichtlich wäre. Wir nehmen die Abstraktion zu Hilfe.

Zuerst führen wir eine Notation für Konfigurationen, bzw. Knoten, ein. Anstelle der umständlichen Darstellung:

I
III
IIIII
IIIIIII

schreiben wir ein Vierer-Tupel von Zahlen, also in diesem Falle: (1,3,5,7). Die erste Zahl steht für die Anzahl Streichhölzer in der ersten Reihe, die zweite Zahl für die Streichhölzer der zweiten Reihe, usw. Wir lassen der Einfachheit halber noch die Klammern und die Kommata weg. Dann ergibt es einfach eine Zahl: 1357. Jede Konfiguration kann so durch eine Zahl dargestellt werden. Zum Beispiel:

I
II
III
IIIIII

ergibt die Zahl: 1236, und

I
III
II

ergibt die Zahl: 1320 (die letzte Reihe enthält kein Streichholz).
Jetzt haben wir eine bequeme Art, jede Konfiguration darzustellen. Da es in der ersten Reihe zwei (nämlich 0 oder 1), in der zweiten 4 (nämlich 0, 1, 2 oder 3), in der dritten Reihe 5 (nämlich 0, 1, 2, 3, 4, oder 5) und in der vierten Reihe 7 (nämlich 0, 1, 2, 3, 4, 5, 6 oder 7) Möglichkeiten gibt, haben wir insgesamt 2•4•6•8 = 384 mögliche Konfigurationen. Soweit zur Grösse des Graphen.

Exerzieren wir das Problem zunächst einmal an einem kleinen Beispiel durch: Angenommen wir haben nur zwei Reihen mit maximal 1 in der ersten Reihe und 3 Streichhölzer in der zweiten Reihe. Die Ausgangslage ist dann:

I
III

Es gibt dann 8 (=2•4) Konfigurationen. Diese sind: 13, 12, 11, 10, 03, 02, 02, 01 und 00. Der Graph ist in Abbildung 6 visualisiert.

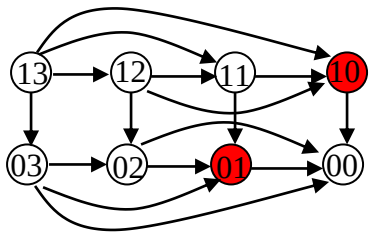

Abbildung 6

Der Kernel besteht aus den Knoten {01, 10}. Das heisst aus allen Konfigurationen muss der Spieler diese Knoten erreichen, um das Spiel zu gewinnen. Warum bilden die beiden Knoten einen Kernel? Ganz einfach. Es sind (1) die einzigen beiden Knoten, welche unter sich nicht verbunden sind und (2) von allen andern Knoten erreicht werden können. So war die Definition des Kernels.
Damit ist dieses einfache Spiel vollständig beschrieben. Sein Kernel ist {01, 10}. Dieses Spiel ist natürlich zu trivial, um ein sinnvolles Spiel zu geben. Es geht hier nur um die Veranschaulichung der Theorie.

Gehen wird zum etwas komplizierteren Fall mit drei Reihen, bei der die Ausgangslage

I
III
IIIII

ist. Jetzt haben wir 2•4•6 = 48 Knoten. Diese sind: {135, 134,..., 130, 125,..., 120,...,..., 001, 000}. Um den Graphen überhaupt noch vernünftig veranschaulichen zu können, greifen wir zu einem Trick. Wir zeichnen ihn drei-dimensional. Das heisst, wir machen ein Gitter und tragen die Anzahl Streichhölzer in der ersten Reihe in der *x*-Achse, die Anzahl Streichhölzer der zweiten Reihe in der *y*-Achse, und die Anzahl Steichhölzer der dritten Reihe in der *z*-Achse ab, in der Art von Abbildung 7.

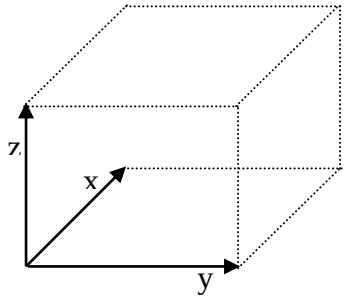

Abbildung 7

Daraus entsteht dann folgendes drei-dimensionales Gitter (Abbildung 8).

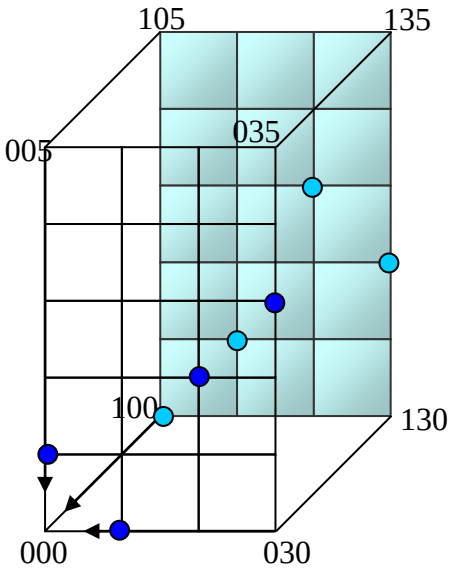

Abbildung 8

Um die Uebersicht zu behalten, wurden nur die 8 Eckpunkte mit den Positionsnamen bezeichnet. Die Bögen können wir uns nun folgendermassen vorstellen: (1) Von jedem Gitterpunkt (es sind insgesamt 48) führt ein Bogen vertikal zu jedem andern Gitterpunkt (*z*-Achse), (2) von jedem Gitterpunkt führt ein Bogen nach links zu jedem andern Gitterpunkt (*y*-Achse), und (3) von jeden Gitterpunkt auf der hinteren Fläche führt ein Bogen zu seinem entsprechenden Punkt in der vorderen Ebene (*x*-Achse).
Dieser Graph besitzt einen Kernel, der aus acht Gitterpunkten gebildet ist (4 Punkte auf der hinteren Fläche; 4 Punkte auf der vorderen Fläche). Der Kernel ist also: {001, 010, 022, 033, 100, 111, 123,132}.

Um diesen Kernel zu finden, gehen wir wie folgt vor: Zunächst ist klar, dass die drei Punkte 001, 010 und 100 Gewinnknoten sind (000 ist per definitionem ein schlechter

Knoten oder wird vom Graphen ausgeschlossen). Welche Gitterpunkte werden durch die drei Gewinnknoten abgedeckt, d.h. welchen Knoten erreichen diese drei Gewinnknoten durch einen Bogen? Auf der Vorderfläche alle Punkte 00*, 0*1, 0*0, 01*, auf der Hinterfläche alle Punkte 10*, 1*0 (wobei * für eine beliebige Zahl im Rahmen der Notation steht). Die nächst-tiefsten (oder lexiograghic kleinsten) Punkte, die nicht abgedeckt sind, sind: auf der Vorderseite 022 und auf der Hinterseite 111. Beide sind nicht miteinander verbunden, also gehören diese zum Kernel. Der Punkt 022 deckt alle Punkte *22, 0*2 und 02* ab; der Punkt 111 deckt alle Punkte *11, 1*1 und 11* ab. Die beiden nächst-tiefste unabgedeckte Punkt sind somit: auf der Vorderseite 033, und auf der Hinterseite 123. Diese sind ebenfalls nicht miteinander verbunden. Also gehören sie zum Kernel. Der Punkt 033 deckt die Punkte *33, 0*3 und 03* ab, und 123 deckt *23, 1*3 und 12* ab. Die einzigen noch unabgedeckten Punklte sind 132, 134 und 135. Somit gehört 132 zum Kernel, welcher auch 134 und 135 abdeckt. Der Kernel ist somit vollständig.

Das Problem, den Kernel eines Graphen zu finden, ist im allgemeinen ein nicht-triviales Problem. Es gehört zur Klasse der Knotenabdeckung (vertex-cover problem genannt), welches die Komplexität von NP hat, d.h. ein nicht-deterministisch polynomiales Problem ist. In unserem Fall – von azyklischen Graphen – kann man allerdings einen (bezüglich der Knotenmenge) linearen Algorithmus angeben, welcher den Kernel findet. Der Algorithmus ist:

```
PROCEDURE FindKernel(Graph):KernelSet
Kernel =  end-winning nodes
RestNodes = AllNodes \ end-losing nodes
  (stored in lexicographic order)
For AllNodes (in lexicographic order) do
  Remove all nodes from RestNodes
    which are adjacent to any node in Kernel
  If RestNodes is not empty then
    Put the first node in RestNodes into the Kernel
Endfor
Return Kernel
```

Folgendes Programm in Pascal implementiert diesen Algorythmus, welches den Kernel zum Streichholzproblem mit vier Reihen findet:

```
Program FindKernel; (* steichholzproblem *)
  const NUM=4; MAX : array[1..NUM] of integer = (1,3,5,7); TOT=384;
  type TPosition = record a:array[1..4] of integer; end;
  type Tkernel = record n:integer; PP:array[1..100] of TPosition; end;
  var P:TPosition; K:TKernel; i:integer; fil:text;
```

```
procedure NextP(var P:TPosition); var i:integer;
begin
  i:=NUM; inc(P.a[i]);
  while P.a[i]>MAX[i] do begin
    P.a[i]:=0; dec(i); if i>0 then inc(P.a[i]) else exit;
  end;
end;

function CheckP(var P:TPosition; var K:TKernel):boolean;
var i,j,c,cmin:integer;
begin
  cmin:=100;
  for i:=1 to K.n do begin
    c:=0;
    for j:=1 to NUM do if P.a[j]<>K.PP[i].a[j] then inc(c);
    if c<cmin then cmin:=c;
  end;
  if (cmin>=2) then begin inc(K.n); K.PP[K.n]:=P; end;
end;

procedure writeP(var P:TPosition); var i:integer;
begin for i:=1 to NUM-1 do write(P.a[i]:1); write(P.a[NUM]:1,' '); end;

procedure packP(var P:TPosition; a,b,c,d:integer);
begin P.a[1]:=a; P.a[2]:=b; P.a[3]:=c; P.a[4]:=d; end;

begin
  K.n:=4;
  PackP(K.PP[4],0,0,0,1); PackP(K.PP[3],0,0,1,0);
  PackP(K.PP[2],0,1,0,0); PackP(K.PP[1],1,0,0,0);
  PackP(P,0,0,0,0);
  for i:=1 to TOT do begin NextP(P); CheckP(P,K); end;
  assign(fil,'k.txt'); rewrite(fil);
  for i:=1 to K.n do writeP(fil,K.PP[i]);
  close(fil);
end.
```

Das Programm fand, dass der Kernel aus den folgenden 48 Konfigurationen besteht:

{1000 0100 0010 0001 0022 0033 0044 0055 0111 0123 0132 0145 0154 0202 0213 0220 0231 0246 0257 0303 0312 0321 0330 0347 0356 1011 1023 1032 1045 1054 1101 1110 1122 1133 1144 1155 1203 1212 1221 1230 1247 1256 1302 1313 1320 1331 1346 1357}.

Interessanterweise ist 1357 – also die Ausgangskonfiguration – im Kernel. Das bedeutet, dass derjenige Spieler, der beginnt, verlieren muss, wenn sein Gegenspieler keinen Fehler begeht. Denn der erste Zug geht notwendigerweise von einer Kernelkonfiguration weg, und damit in eine Nicht-Kernelkonfiguration.

Ein Problem bleibt noch offen. Wie kann ein Spieler all diese Gewinnknoten im Kopf gehalten? Es scheint keine einfache Regel zu geben. Zunächst muss festgehalten werden, dass die Summe der Streichhölzer nur eine gerade Zahl ist (mit Ausnahme der vier ersten Gewinnknoten (Summe ist 1) , und allen Kombinationen, bei der drei 1-er erscheinen (Summe ist 3). Führen wir eine Notation ein: [xx,yy] bedeute, dass eine Permutation xx gemischt mit einer Permutation yy vorkommen kann. Also beispielsweise [22,00] bedeute: alle Permutationen (0022, 0202, 0220) (2200 ist natürlich nicht erlaubt, da sie gar keine legale Konfiguration ist). Oder [0,145] bedeute: (0145, 0154, 1045, 1054). Dann können wir den Kernel folgendermassen memorisieren:

Summe	gute Knoten
1	[000,1]
3	[111,0]
4	[00,22]
6	[00,33],[11,22], [0,123]
8	[00,44], [11,33]
10	[00,55], [11,44], [0,145]
12	[11,55], [0,246]
14	[0,257], [0,347], [0,356], [1256], [1346]
16	[1357]

Wir werden gleich sehen, dass es eine für eine zweite Spielvariante des Spiels eine einfache Formel gibt, um den Kernel vollständig zu beschreiben.

Eine erfahrene Spielerin, welche die mathematische Theorie nicht kennt, schrieb mir folgendes, nachdem ich ihr den Kernel mitteilte:

> "Wie Du richtig vermutest, gewinne ich dieses Spiel immer, sofern der andere anfängt oder, wenn ich anfangen muss, sofort einen Fehler macht. Dabei merke ich mir die Gewinnpositionen natürlich nicht in dieser Form:
>
> (1000 0100 0010 0001 0022 0033 0044 0055 0111 0123 0132 0145 0154 0202 0213 0220 0231 0246 0257 0303 0312 0321 0330 0347 0356 1011 1023 1032 1045 1054 1101 1110 1122 1133 1144 1155 1203 1212 1221 1230 1247 1256 1302 1313 1320 1331 1346 1357),
>
> sondern etwas einfacher. Die Grundüberlegung ist die, dass es ja keine Rolle spielt, in welchen Reihen wieviele Hölzchen liegen. Demzufolge sind z.B. die Varianten 0123, 0132, 0231, 0231, 0312, 0321, 1023, 1032, 1203, 1230, 1302 und 1320 von der Gewinnposition aus betrachtet identisch. Was ich mir hier merken muss, ist einzig die Gedankenstütze 1-2-3, die alle obigen Varianten abdeckt. Ebenso ist die Eselsbrücke 1-4-5 für 0145, 0154, 1045, 1054 zuständig.
>
> Die zweite Regel ist die der Paare. Alle Paarpositionen sind Gewinnpositionen. Damit wären weitere 13 Positionen (0022, 0033, 0044, 0055, 0220, 0303, 0330, 1122, 1133, 1144, 1155, 1221 und 1331) durch das Merkwort Paare charakterisiert.
>
> Die banalen Endpositionen 1000, 0100, 0111 etc. kann man auch ohne Gedankenstütze als Gewinnpositionen erkennen.
>
> Bleiben nur noch die wenigen Anfangspositionen mit vielen Hölzchen wie 1346, 1256, 1247, 1256, 1346, auf die man kommen muss, wenn der Gegner zu Beginn nur ein Hölzchen weggenommen hat. Man kann sie sich leicht merken, weil man dann immer auch nur ein Hölzchen, nämlich dasjenige in der nächstfolgenden Reihe, wegnehmen muss. Legt mir mein Gegner also 1347, reagiere ich mit 1346, bei 0357 antworte ich mit 0257 usw. Nimmt er eins aus der letzten Reihe, antworte ich mit dem Wegnehmen eines Hölzchens aus der ersten Reihe, 1356 wird 0356. Von da an gelten dann wieder die obigen Regeln.
>
> Also ist man mit vier Regeln, die man im Kopf behalten muss, immer der Gewinner, sofern der andere beginnt: "1-2-3", "1-4-5", "Paare" und "1 am Anfang". That's it!
>
> Wie ich Dir gesagt habe, hat mir und meiner Freundin dieses Spiel zu seiner Zeit etliche Gewinne eingebracht. Natürlich liessen wir die andern anfangs nach alter Taschenspielermanier ein paarmal gewinnen."

Die Spielerin merkte sich also den Kernel durch die vier Regeln:

1) Alle 1-2-3 und alle 1-4-5 Kombinationen
2) Alle Paar-Konfigurationen 00-22,00-33,00-44,00-55, 11-22,11-33,11-44,11-55

3) Die 'banalen' Konfigurationen 0001, 0010, 0100, 1000, und 0-111
4) Die fünf Ausgangskonfiguratione 1346,1256,1247, 1256, 1346 merkst sie sich einzeln.

Dies entspricht den von mir vorgeschlagenen Memorisationsregeln:
1) [0,123], [0,145]
2) [00,22], [00,33], [00,44], [00,55], [11,22],[11,33], [11,44], [11,55]
3) [000,1], [0,111]
4) [0,257], [0,347], [0,356], [1256], [1346]

Ich fragte dann die Spielerin, was sie macht, wenn sie mit der Konfiguration 0346 konfrontiert werde. Dann muss sie nämlich in die Kernelkonfiguration 0246 gelangen. Aber diese fehlt in ihrer Auflistung!

Interessant ist auch, dass das Spiel mit 5 Reihen (also 9 Hölzchen in der fünften Reihe) uninteressant ist. Da 1357 ein guter Knoten ist, kann derjenige gewinnen, welche beginnt (sofern er keinen Fehler macht), indem er einfach zuerst die fünfte Reihe abräumt! Dieses Spiel hat übrigens 384 gute Knoten und der gesamte Graph hat 3840 Knoten.

5 Das 16-Streichholz-Spiel (zweite Variante)

Eine weitverbreitete Variante des 16-Streichholz-Spiels ist folgende: Das Spiel ist dasselbe wie das im letzten Abschitt beschriebene mit der Ausnahme, dass die Gewinn-Konfiguration 0000 ist, d.h. derjenige, welcher die letzten Streichhölzer entfernt, gewinnt (in der vorigen Variante verliert er).

Diese zweite Variante ist übrigens unter dem Namen 'Nim-Spiel' bekannt. Es wurde anfangs der sechsiger Jahren durch den Spielfilm “Letztes Jahr in Marienbad” im deutschen Raum äusserst populär (1961). Auf dem Internet kann man es an verschiedenen Adressen spielen. Z.B.

www.mannheim.de/gsg/marie.htm

oder auch bei

zaphod.uchicago.edu/~bryan/nim/index.html

Auch auf der Internetseite

www.cut-the-knot.com/nim_st.html

wird eine Variante des Spiels vorgestellt: Gewinner ist immer derjenige, welcher das letzte Hölzchen wegnimmt, d.h. die Konfiguration 0000 ist jetzt die einzige Endgewinn-Konfiguration. Auch auf der Internet-Seite

www.journey.sunsb.edu/Wise/Games/Nim.html

werden theoretische Ueberlegungen zum Spiel angestellt und auf weitere Seiten

verwiesen. In der Spick-Ausgabe vom Oktober 1999 wird das Spiel kurz vorgestellt. Die Aussagen sind aber zum Teil falsch. Die Konfiguration 1156 gehört nicht in den Kernel, wie fälschlicherweise angegeben, sondern 1256.

Für das Spiel in dieser zweiten Variante mit dem Gewinnknoten 000 à drei Reihen ergibt sich jetzt folgender Kernel: {000 011 022 033 101 110 123 132}.

Für das Spiel mit der Gewinn-Konfiguration 0000 à vier Reihen ergibt sich der Kernel wie folgt: {0000 0011 0022 0033 0044 0055 0101 0110 0123 0132 0145 0154 0202 0213 0220 0231 0246 0257 0303 0312 0321 0330 0347 0356 1001 1010 1023 1032 1045 1054 1100 1111 1122 1133 1144 1155 1203 1212 1221 1230 1247 1256 1302 1313 1320 1331 1346 1357}.

Es ist interessant festzustellen, dass die beiden Kernel dieser Spielvariante sich nur geringfügig vom ursprünglichen Spiel (bei dem derjenige mit dem letzten Zug Verlierer ist) unterscheidet. Der Unterschied ist folgender: Bei dieser Variante sind alle Konfigurationen im Kernel, bei der eine gerade Anzahl von Nullen und Einsen vorkommt, während bei der ursprünglichen Variante alle Konfigurationen vorkommen, bei der eine ungerade Anzahl von Nullen und Einzen vorkommt. Alle andern Konfigurationen im Kernel sind gleich. Das heisst, der Spielverlauf ist genau gleich ausser in der Endphase des Spiels. Diese Tatsache ist einigermassen erstaunlich. Die Erklärung für diese auf den ersten Blick erstaunliche Tatsache ist einfach. Wenn auf bestimmten Anzahl von Reihen nur noch ein Streichholz sich befindet, dann ist das Spiel determiniert in dem Sinne, dass beide Spieler keine Wahl mehr haben. Isi zum Beispiel die Konfiguration 0111 gegeben, dann kann in den nächsten drei Zügen jeder Spieler nur ein ein Streichholz wegnehmen. Gewinner oder Verlierer ist dann derjenige, der das letzte nimmt. Bei beiden Spielvarianten ist also je nachdem 0111 ein guter Knoten oder ein schlechter Knoten. Denn die Anzahl der Züge ist notwendigerweise eine ungerade Zahl. Wenn mehr als ein Streichholz in der Reihe liegt, besteht noch die Wahlfreiheit. Ist also beispielsweise der in beiden Spielvarianten definierte Kernelknoten 0022 erreicht, so muss der andere Spieler mit 0012 oder 0002 antworten. Je nach Variante hat jetzt der erste Spieler die Wahl, ob er nach 0001 (Kernelkonfiguration in der ersten Variante) oder 0011 (Kernelkonfiguration in der zweiten Variante) gelangen will. Konfigurationen mit mehr als einem Streichholz in einer Reihe haben somit keinen Einfluss auf den Spielverlauf in der Endphase. Es ist als, ob der Kernel in der Endphase lokal leicht verschoben ist.

Auf jeden Fall gibt es in dieser neuen Variante eine extrem elegante Lösung, um alle Konfigurationen des Kernels zu beschreiben. Dazu muss eine neue Notation für die Konfigurationen eingeführt werden.

Zunächst repräsentieren wir die Konfigurationen in binärer Zahlenform. Das heisst, jede Ziffer in einer Konfiguration stellen wir als binäre Zahl dar: (0=000, 1=001, 2=010, 3=011, 4=100, 5=101, 6=110, 7=111). Die Konfiguration 1054 würde also jetzt repräsentiert als: 001 000 101 100. Wir nennen diese Darstellung die *binäre Darstellung* einer Konfiguration. Wenn wir die vier erhaltenen binären Zahlen

untereinander schreiben, so erhalten wir:

001
000
101
100

Betrachten wir die vertikalen Kollonnen, so stellen wir fest, dass die Anzahl '1' immer gerade ist. Dies gilt für alle Konfigurationen im Kernel. Zum Beispiel: 1357 ergibt:

001
011
101
111

Auch hier ist die Anzahl '1' immer gerade. Mathematisch ausgedrückt bedeutet das, dass die XOR-Operation (exklusive Oder-Operation) angewandt auf die Bits in den Kollonnen immer '0' ergibt. Zum Beispiel: 0 XOR 0 XOR 1 XOR 1 = 0.

Andererseits ergibt eine Konfiguration, die nicht im Kernel ist, immer in mindestens einer Kollonne eine ungerade Anzahl '1'. Zum Beispiel: 1234 ergibt:

001
010
011
100

Die erste Kollonne enthält eine ungerade Anzahl '1' (nämlich eine '1').
Damit haben wir eine elegante Beschreibung aller Kernel-konfigurationen, die jetzt als Theorem zusammengefasst werden soll.

Theorem: Genau diese Konfigurationen sind im Kernel, bei der die paarweise XOR-Operation der Bit-Positionen in der binären Darstellung '0' ergibt.

Beweis: Gemäss der Kerneldefinition müssen wir zeigen, dass
a) Jeder Zug mit Ausgangspunkt einer Konfiguration im Kernel muss zu einer Konfiguration führen, die nicht im Kernel ist.
b) Für jeden Ausgangspunkt einer Konfiguration, die nicht im Kernel ist, gibt es einen Zug, der zu einer Konfiguration im Kernel führt.
Zu a) Wir nehmen also an, dass die Anzahl der '1' in jeder Kollonne gerade ist. Ein Zug bedeutet, eine Zeile verändern. Dies heisst aber, dass in mindestens einer Kollonne die Anzahl '1' Bits ungerade wird.
Zu b) Angenommen, in mindestens einer Kollonne existiert eine ungerade Anzahl '1' Bits. Sei *j* die erste Kollonne von links, in der eine ungerade Anzahl '1' vorkomme. Wir suchen eine Zeile, in der eine '1' in Kollonne *j* vorkommt. Eine solche Zeile muss existieren, da die Anzahl '1' in dieser Kollonne mindestens eins sein muss. Sei *i* die gefundene Zeile. In dieser Zeile *i* verwandeln wir alle '1' in eine '0' und eine '0' in eine '1' , wenn immer die Kollonne eine ungerade Anzahl '1' enthält. Dadurch erhalten alle Kollonnen eine gerade Anzahl '1', was einer Konfiguration im Kernel entspricht.

Q.E.D.

Der Beweis gibt auch eine einfache Spielstrategie (vorausgesetzt man denkt sich die Zahlen binär!). Ansonsten kann folgende Funktion `GetNextPosition` ausgeführt werden, welche den optimalen nächsten Zug generiert. Die Funktion produziert also aus jeder Konfiguration, die nicht im Kernel ist, eine Kernelkonfiguration, die im nächsten Zug erreicht werden kann. Wenn der Funktion eine Kernelkonfiguration präsentiert wird, dann wird diejenige Nicht-Kernelkonfiguration generiert, die aus der Reihe mit den meisten Hölzchen nur eines wegnimmt.

```
procedure GetNextPosition(var P:TPosition);
    {works only if winning position is 0000,
     overwrites P with the result}
  var i,j,n,h:integer;
      bits: array[1..NUM] of string[4];
      res:string[4];
  function pow(b,e:integer):integer; var i,n:integer;
  begin n:=1; for i:=1 to e do n:=n*b; pow:=n; end;

begin
  res[0]:='    ';
  for i:=1 to NUM do begin
    bits[i,0]:='    ';
    for j:=1 to 4 do
      if P.a[i] mod pow(2,j)>=pow(2,j-1) then
        bits[i,j]:='1'
      else
        bits[i,j]:='0';
  end;
  for j:=1 to 4 do begin
    n:=0;
    for i:=1 to NUM do
      if bits[i,j]='1' then inc(n);
    if odd(n) then res[j]:='1' else res[j]:='0';
  end;

  if res<>'0000' then begin {P is not in Kernel}
    j:=4; while res[j]='0' do dec(j);
    i:=1; while bits[i,j]='0' do inc(i);
    bits[i,j]:='0';
    for h:=j-1 downto 1 do begin
      if res[h]='1' then begin
        if bits[i,h]='1' then bits[i,h]:='0' else bits[i,h]:='1';
      end;
    end;
    P.a[i]:=0;
    for j:=1 to 4 do
      if bits[i,j]='1' then P.a[i]:=P.a[i]+pow(2,j-1);
  end else begin
    {P is in Kernel, so remove only one from the largest number}
    n:=0; for i:=1 to NUM do if P.a[i]>n then begin n:=P.a[i]; j:=i; end;
    dec(P.a[j]);
  end;
end;
```

Die Funktion `GetNextPosition` ist das komplette Programm, welches zu jeder Konfiguration (ausser 0000) und zu jedem Nim-Spiel immer den optimalen Zug zurückgibt. Damit haben wir das Spiel vollständig beschrieben.

Aus dem letzten Theorem folgt auch, dass sämtliche Nim-Spiele dadurch abgedeckt sind: Die Anzahl Reihen kann geliebig sein, und auch die Anzahl Streichhölzer pro Reihe kann eine beliebige positive Zahl sein.

6 Eine weitere Spielvariante (Nachtrag)

Eine weitere interessante Spielvariante ergibt sich dadurch, dass nachfolgende Spielzüge voneinander abhängig sind. Zum Beispiel könnte man die Beschränkung auferlegen, dass ein nachfolgender Zug nicht dieselbe Anzahl Objekte entfernen darf wie der Zug des Gegenspielers. Diese Variante soll zunächst am Beispiel des 16-Objekte Spiels aufgezeigt werden. Jeder Spieler darf maximal 4 Objekte und muss mindestens ein Objekt entfernen. Wer das oder die letzten Objekte nimmt, gewinnt. Nun also mit der Einschränkung, dass ein Spieler 1, 2, 3 oder 4 nehmen darf, aber nicht gleichviel wie bein Vorzug. Zum Beispiel, wenn der Gegenspieler 2 Objekte entfernt hat, darf der Spieler nur 1, 3 oder 4 Objekte entfernen, aber nicht 2.

Auf den ersten Blick scheint es, dass dieses Spiel mit der in Abschnitt 2 dargelegten Theorie des Kernels nicht analysiert werden kann. Denn ein Zug ist ja abhängig von der vorausgehenden Zugsequenz. Eine wichtige, nicht explizit geschriebene Voraussetzung, um einen Kernel zu finden, ist ja eben, dass jeder Zug unabhängig und lokal definiert ist und an keinerlei Abhängigkeiten einer Zugsequenz gebunden ist. Ist ein Zug getan, kann der Gegenspieler unabhängig seinen Zug wählen. Jeder hat bei jedem Zug immer dieselben Regeln. Diese Voraussetzung ist hier verletzt.

Eine etwas nähere Betrachtung jedoch erlaubt uns, dieses Spiel in der genau gleichen Weise zu betrachten wie die bisher behandelten NIM-Spiele. Die Kernidee besteht darin, einen Zug (also eine Kante im Graphen) *anders* zu betrachten. Nun, wie anders? Wir müssen die Sequenzabhängigkeiten irgendwie loswerden, um wieder zu einen "Zustandgraphen" zu kommen. Es muss also ein *anderer* Graph konstruiert werden.
Betrachten wir, was ein Zustand (Konfiguration oder Knoten im Graph) des Spieles ist. Dies führt uns dann zur Knotenmenge des Graphen. Ein Zustand ist eine Konfiguration im früheren Sinne *zusammen* mit der Anzahl Objekte, die vom Gegenspieler entfernt wurden. Im 16-Objekte-Spiel können wir also eine Konfiguration durch zwei Zahlen beschreiben: die verbleibenden Anzahl Objekte und die im Vorzug weggenommen Objekte, d.h. als Zahlentupel. Die Konfiguration, bei der 12 Objekte verbleiben, wobei 2 weggenommen wurden, wird also durch <12,2> beschrieben.
Die Gewinnpositionen können also durch die vier Knoten <0,4>, <0,3>, <0,2>, 0,1> beschrieben werden, d.h. alle Konfirurationen, bei denen kein Objekt übrigbleibt. Als Gewinnposition muss auch die Konfiguration <1,1> betrachtet werden. Denn im letzten Zug wurde ein Objekt entfernt, jetzt bleibt eins übrig, das laut Regel nicht mehr genommen werden darf, da Spieler und Gegenspieler sonst hintereinander dieselbe Anzahl Objekte entfernen würden.
Die Knoten des Graphen sind also alle Zahlentupel <n,m>, welche die Bedingungen

$$0 \le n \le 16,$$
$$1 \le m \le 4, \text{ wenn } 0 \le n \le 12$$
$$1 \le m \le 16 - n, \text{ wenn } 13 \le n \le 15$$
$$m = 0, \text{ wenn } n = 16$$

erfüllen. Die Anzahl Knoten, die sich daraus ergeben sind: 12·4+3+2+1+1=55. Von jedem Knoten <n,m> führt eine Kante zu einem andern Knoten <p,q>, wenn p+q=n und q≠m erfüllt ist. Damit ist der Graph vollständig beschrieben. Abbildung 9 gibt einen Teil des Graphen wieder. Die Gewinnposition sind <1,1>, <0,1>, <0,2>, <0,3> und <0.4> und gehören somit zum Kernel. Alle Knoten links von diesen fünf markierten Positionen haben eine Kante, die in eine Gewinnknoten hineinführen. Zum Beispiel führt eine Kante von <4,1> zu <0,4>. Diese Knoten gehören somit alle nicht zum Kernel. Die nächsten sechs Knoten, die wieder zum Kernel gehören sind <4,4>, <5,4>, <5,3>, <5,2>, <5,1>, <6,1>, da sie alle unter sich nicht verbunden sind und nur Kanten besitzen, die rechts in Knoten hineinführen, die nicht zum Kernel gehören.

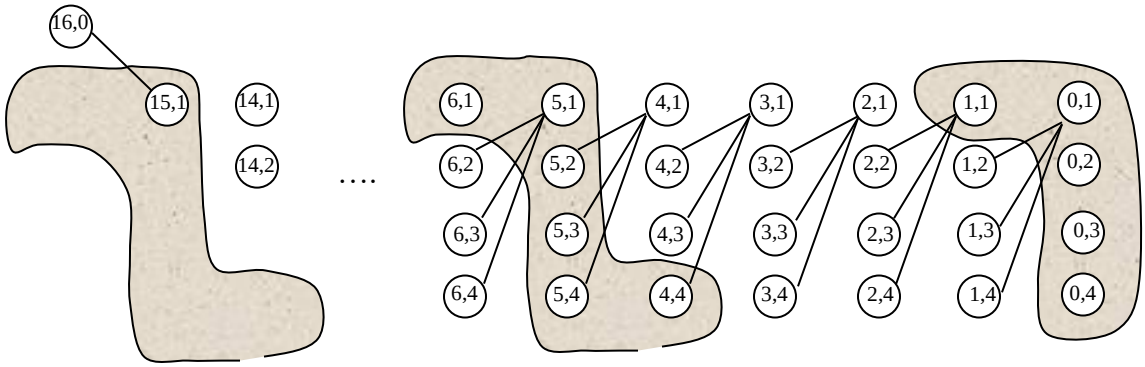

Abbildung 9

Es ist interessant festzustellen, dass diese Variante des Spiels sich gar nicht so stark vom ursprünglichen 16-Objekte Spiel unterscheidet. All Konfigurationen, die 0, 5, 10 oder 15 Objekte zurücklassen, sind im Kernel. Zudem gehören in den Kernel alle Konfigurationen, die 4, 9, … Objekte zurücklassen, wobei 4 Objekte im Vorzug entfernt wurden, und diejenigen, die 1, 6, 11, … Objekte zurücklassen, wobei 1 Objekt im Vorzug entfernt wurde. Ist der Spieler also mit der Konfiguration <8,3> konfrontiert, so kann er nicht 3 Objekte entfernen, da sein Gegenspieler soeben 3 Objekte entfernt hat, sondern er muss 4 Objekte entfernen, um auf die Kernelposition <4,4> zu gelangen. Ist der Spieler mit <7,2> konfrontiert, so muss er ein Objekt entfernen, um auf Konfiguration <6,1> zu gelangen. Ausser mit diesem wenigen Ausnahmen, bleibt die Gewinnstrategie also dieselbe, wie beim ursprünglichen 16-Objekte Spiel. Dies ist einigermassen erstaunlich, da die neue Variante einen Graphen besitzt, der doch sehr verschieden vom ursprünglichen Graphen in Abbildung 4 ist.

Wnden wir uns nun dem Streichholz-Spiel zu drei Reihen zu. Also dem Spiel mit der Ausgangskonfiguration

I
III
IIIII

Das ursprüngliche Spiel führte uns zum Graphen in Abbildung 8. Diese Variation führt uns in ähnlicher Weise zu einem Graphen, bei dem jeder jede Konfiguration durch ein Tupel <abc,m> von Zahlen beschrieben werden kann, wobei abc die Konfiguration im ursprünglichen Spiel ist und *m* die Anzahl der Streichhölzer, die im Vorzug entfernt wurden. Sei A=max(a)=5, B=max(b)=3 und C=max(c)=1. Dann gibt es genau eine Konfiruration (nämlich 531), bei der m=0 sein darf. Und für 531=*abc* gilt: max(A-a,B-b,C-c)=0. Für 7 Konfigurationen *abc* gilt: max(A-a,B-b,C-c)=1, nämlich für 431, 421, 420, 521, 430, 530, 520. Für diese haben wir m=1. Für 10 Konfigurationen *abc* gilt: max(A-a,B-b,C-c)=2, für diese haben wir: $1 \le m \le 2$. Für 14 weitere Konfigurationen *abc* gilt: max(A-a,B-b,C-c)=3, und für diese haben wir: $1 \le m \le 3$. Für weitere 8 Konfigurationen *abc* gilt: max(A-a,B-b,C-c)=4, und für diese haben wir: $1 \le m \le 4$. Schliesslich bleiben noch 8 Konfigurationen *abc* mit: max(A-a,B-b,C-c)=5, und für diese haben wir: $1 \le m \le 5$. Macht zusammen 48 Konfigurationen *abc*. Insgesamt besitzt der Graph für diese neue Spielvariante somit: $1+7+10 \cdot 2+14 \cdot 3+8 \cdot 4+8 \cdot 5 = 142$ Knoten.

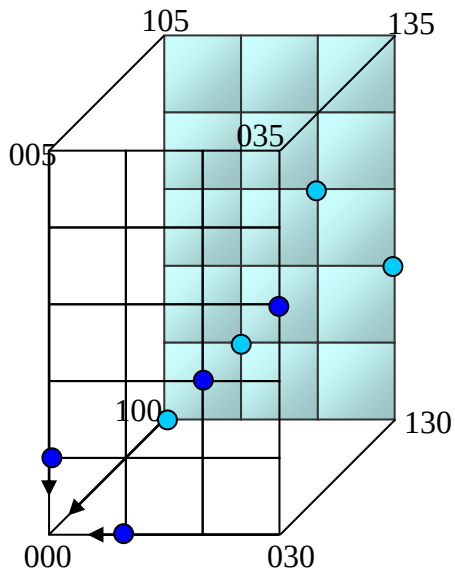

Abbildung 8

Bibliographie

TUCKER A, [1995], Applied Combinatorics, North-Holland.