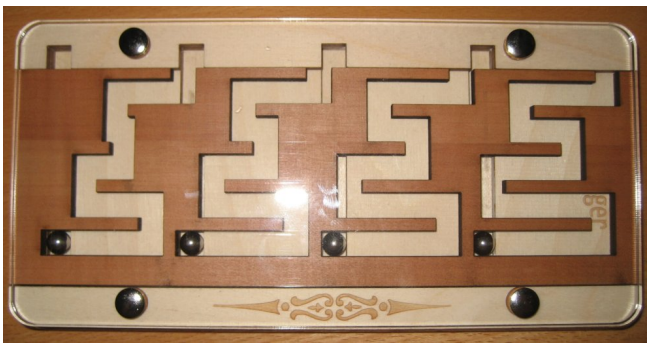
Some Thoughts on Jean-Claude Constantin's "Kugellager" Puzzle

A Short Analysis of Over 1200 Moves To Solve the Puzzle

by Dr. Goetz Schwandtner

23/04/2012

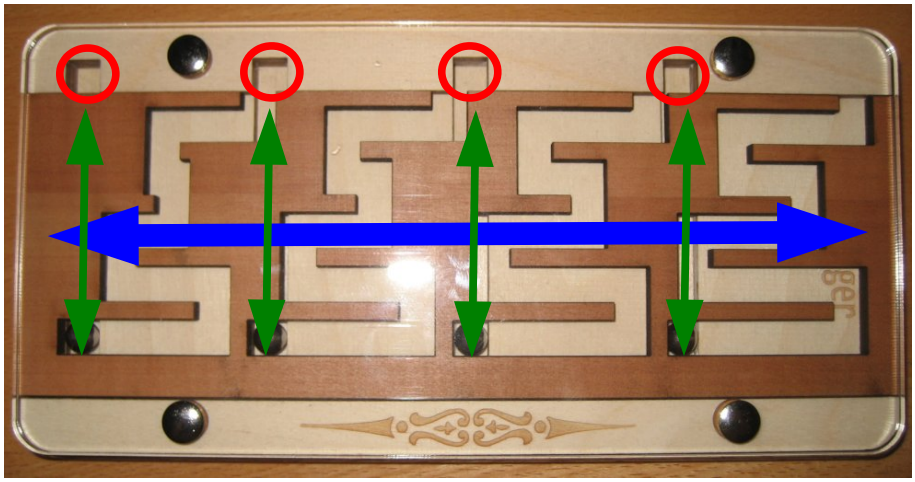


1 Introduction:

In this paper I will present the "Kugellager" puzzle and will give a brief discription showing how this puzzle works. The puzzle has a very high number of moves to solve (over 1200). After defining what a "move" is and after defining some notation, we will have a closer look at the number of moves that are needed. An analysis of the different situations (or configurations) in the puzzle will lead to an insight why this high number of moves is needed. This also leads to some parallels to existing puzzles (and other new puzzles) that we will discuss briefly.

2 Short Description of the Puzzle:

The Kugellager by Jean-Claude Constantin is a sliding puzzle consisting of laser-cut wood, four metal balls, an acrylic cover and some screws to keep erverything together. There is a slider that can slide horizontally (blue arrow) and four balls moving in some grooves allowing them to move only vertically and not allowing them to drop out of the puzzle (green arrows).



The slider can be moved by hand, the balls can be moved by tilting the puzzle. To play with the puzzle, the balls are guided through the labyrinth of the slider. In the following two pictures some moves have been carried out already: The slider is moved and the balls have moved – the leftmost ball is in the middle of a move, as we will define it below. The first picture shows the puzzle after the slider has moved to the left, and the leftmost ball is on the way up.

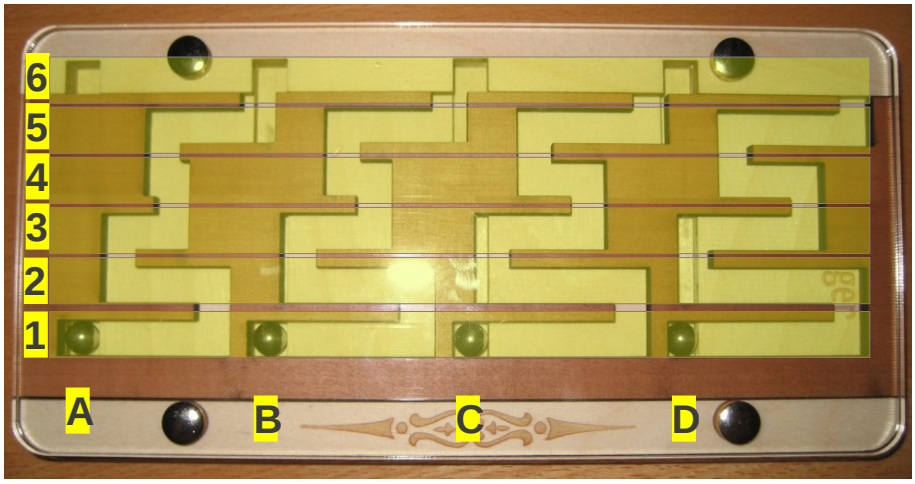
The second picture shows the puzzle after quite some more moves. Three balls have moved already and again the leftmost ball is in the middle of a move.

3 How to Play/Goal:

The starting position is as shown with the slider in place and all the balls in the bottom positions. The goal is to move all balls to the top and to remove the slider. To remove the slider, the balls have to be moved to the top and then can be moved to the holes marked with the red circles simultaneously. Only when all balls are in the topmost part of the slider they can be moved to the red circled positions.

4 Notation of positions/configurations:

To make the discussion about puzzle properties easier, I will use enumerate the balls from A (low) to D (high) and the positions from 1 (bottom) to 5 (top) and 6 (red circles):



5 Observations about the puzzle/definitions:

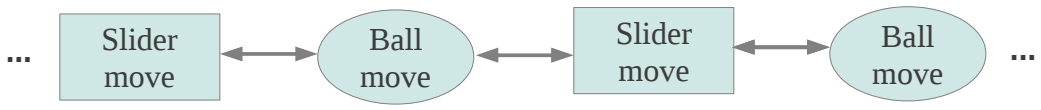
I will now state some observations about the puzzle – all without proof, but most of them can be verified easily by playing with the puzzle and close observation.

1. "The puzzle needs about 1250 moves to be solved." This number of moves was actually provided as part of the description on a German website where I bought the puzzle. But first you have to define what a move is. Similar to the moves found in burr puzzles, I will use the following *definition*:

   A move is a straight movement as far as possible either of one ball (or more balls simultaneously) or the slider.

2. Each puzzle configuration (positions of balls and slider) falls into exactly one of the following three cases:
   (a) All balls are in the bottom positions and cannot move.
   (b) All balls are in the top of the slider and can move to the red circled holes.
   (c) **Only one ball** can move and there is **only one possible move**, between two adjacent rows.

   These two observations lead to another one, about move sequences of the puzzle. At each point you can either move one ball or the slider. If you have moved the ball, you have to move the slider to move another ball, and alternate between slider and ball moves. The general structure of move sequences is always:



3. A ball can only move between row 1 and 2 if all lower balls are in row 5 and all higher balls are in rows 1, 2, or 5.
4. A ball can only move between row 2 and 3 if all lower balls are in row 1 and all higher balls are in rows 1, 2, or 3.
5. A ball can only move between row 3 and 4 if all lower balls are in row 1, 2, or 5 and all higher balls are in any row.
6. A ball can only move between row 4 and 5 if all lower balls are in rows 1, 2, or 3 and all higher balls in any row.

To get a clearer and more succinct view of these row conditions, please have a look at the following table:

Ball moving	Lower balls have to be in	Higher balls have to be in
1 --- 2	5	1 or 2 or 5
2 --- 3	1	1 or 2 or 3
3 --- 4	1 or 2 or 5	any
4 --- 5	1 or 2 or 3	any

6 Counting the moves:

During solving it feels like the very high number of moves (1250) could be correct. The easiest way to verify is to do some counting, but with that big number of moves and easy moving balls this method is error prone, of course. Pit Khiam Goh has written a program to simulate the puzzle and verify the counting and it seems now the numbers are correct.

- Let **T(n)** be the number of steps needed to move all balls from row 1 to row 5 for a

puzzle with **n** balls, or equivalently with the only **n** lower balls.

- Let **S(n)** be the number of slider moves needed in a move sequence to move all balls from row 1 to row 5.
  Note that by observation 2 above, the ball moves and slider moves alternate, hence **T(n)=2*S(n)**
- The number of steps **L(n)** to solve the puzzle with n balls is actually **L(n)=T(n)+2,** as we need to move the slider to the left and the balls to row 6.

We use **T(n**) and **S(n)**, as this function can also be determined for the lower **n** balls in a puzzle with more than **n** balls (the lower **n** balls cannot be moved to row 6 in this case).

With some counting we determine the values of **S(n)** to get (maybe with some little errors):

n	**S(n)**	**T(n)**	**L(n)**
1	4	8	**10**
2	**S(1)**+20=24	48	**50**
3	**S(2)**+101=124	248	**250**
4	**S(3)**+500=624	1248	**1250**

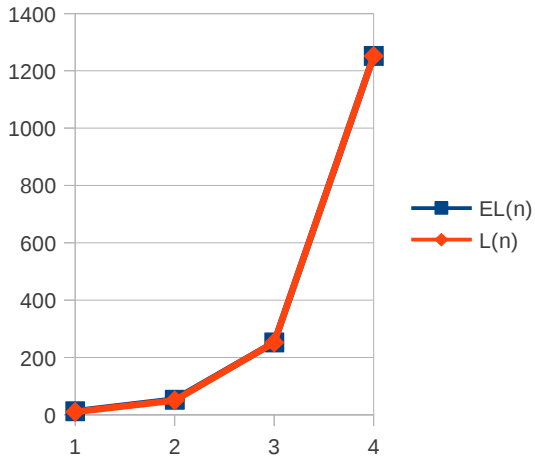
For **S(4)** there are 250 moves from the goal configuration of **S(3)** to the configuration, where Ball D first enters row 3, and then moves straight to row 5. This makes perfect sense, as for the ball D to move from row 1 to two the other balls have to be in row 5, while for D moving to row 3, the other balls have to be in row 1, which can be achieved the same way as solving for three balls, i.e. **S(3)**.

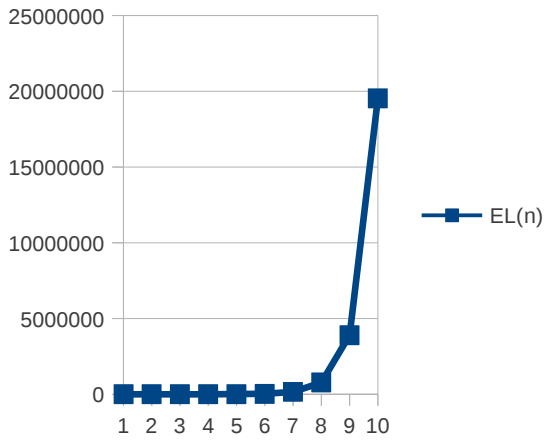
7 Explicit form for S(n), T(n), and L(n):

If we have a look at the values for **S(n)** we might get an idea how to estimate the number of moves and how to derive an explicit form for **S(n), T(n),** and **L(n)**:

Function	**n=1**	**n=2**	**n=3**	**n=4**	**explicit**
S(n)	$4 = 5^1-1$	$24 = 5^2-1$	$124 = 5^3-1$	$624 = 5^4-1$	$\mathbf{5^n-1}$
T(n)	$8 = 2*5^1-2$	$48 = 2*5^2-2$	$248 = 2*5^3-2$	$1248 = 2*5^4-2$	$\mathbf{2*5^n-2}$
L(n)	$10 = 2*5^1$	$50 = 2*25 = 2*5^2$	$250 = 2*5^3$	$1250 = 2*5^4$	$\mathbf{2*5^n}$

Following this scheme, the number of moves to solve the puzzle for **n** balls is exponential with base 5, leading to a fast growth with growing **n**. We shortly visualize this for **L(n)** and an explicit form $\mathbf{EL(n)=2*5^n}$. In the left diagram, the known values for **L(n)** are shown together with **EL(n);** in the right diagram **EL(n)** is drawn for **n** up to 10, showing the quick growth of this function.





8 Recursion or explicit formula:

We have counted some values for **L(n).** However, four values seem to be too little to derive a sensible recursion, yet even an explicit formula for **L(n)**. Maybe a closer look at the restrictions for moves presented above leads to a provable recursion scheme and confirm the above assumption.

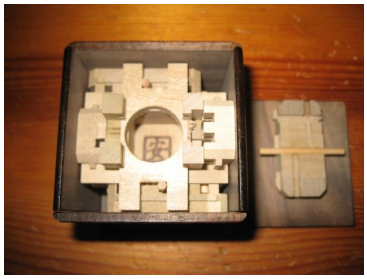
9 Similarities with well known puzzles (binary and ternary):

There are some well known puzzles with n moving pieces where a piece can only move depending on a certain configuration of the lower pieces, sometimes also of the higher pieces. Examples for this are the SpinOut puzzle by William Keister where each piece has two states and can therefore be called a binary puzzle. Equivalent to SpinOut is the much older Chinese Rings puzzle, coming with different numbers of rings, most commonly between 4 and 9. There is also the "The Brain" puzzle, shaped completely different, but equivalent to Chinese Rings with 8 rings. There is also an equivalent burr puzzle, the Binary Burr by Bill Cutler, a description of which can be found on his home page [8]. A two dimensional version is the Key Puzzle by Pit Khiam Goh. Bill Cutler has a description and pictures on his home page at [9]. A binary puzzle with some additional settings for solution length and difficultiy is the Hexadecimal Puzzle, also by William Keister. There is also a Japanese puzzle box with a binary scheme: The "Cubi" by Akio Kamei from the Karakuri Group [10]. A nice diagonnaly moving puzzle with binary scheme is the "Barcode Burr", demonstrated in the YouTube-video [11] by its maker Lee Krasnow [12].

A description of the oldest of these puzzles, the Chinese Rings puzzle, can be found in the book by Slocum and Botermans [4], including solution and building instructions.

Three of these puzzles can currently be found in my private collection, first picture in the starting configuration and in the second after some moves: SpinOut (top), The Brain (left), Chinese Rings (right).

The CUBI is also available as "Small CUBI (M-11-6)", a Karakuri Christmas present by Akio Kamei. It does not look very spectacular from the outside, but is a very nice design with a tricky mechanism involving springs (as shown in the picture to the right).

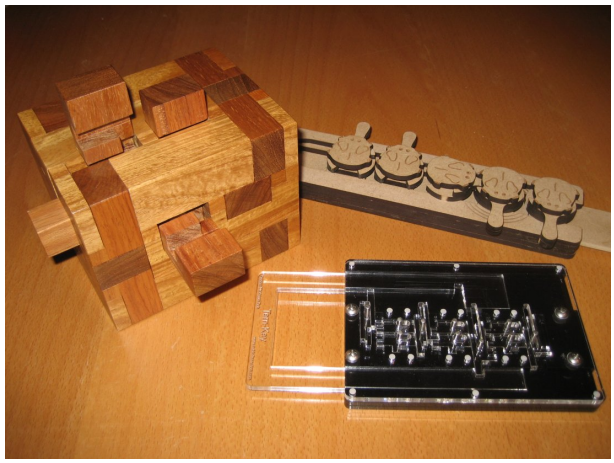


I do not have the Binary Burr in my collection, which is a burr puzzle with usually 6 ring pieces, designed by Bill cutler and created by Jerry McFarcland. However my puzzle friend Dirk Weber was visiting me recently and he had a special version of the Binary Burr to play with him: a Binary Burr with 10 ring pieces, which takes really long to solve it and bring it back to the starting position. Here are some pictures of the puzzle and during solving:

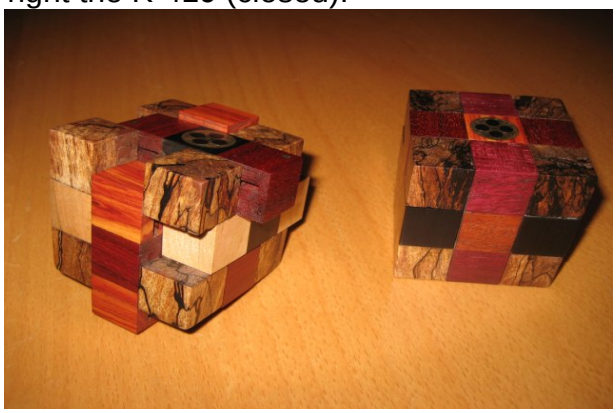
There are also some ternary puzzles, where each "ring" piece can have three different states, like the Tern Key and the new Ternary Burr, both by Pit Khiam Goh, and the Crazy Elephant Dance by Markus Goetz. There is a nice Javascript animation and even a solver for the Crazy Elephant Dance on Markus' Homepage [5]. Maybe the oldest ternary puzzle is the Loony Loop puzzle, a metal and string disentanglement puzzle presented in the fourth volume of [14]. A previously unpublished sliding block puzzle [15] by Bob Hearn is also ternary and a totally different puzzle. It was developed during the research of the space complexity[1] of sliding block puzzles.
A detailed description of the Ternary Burr, including a recursion formula for the number of moves for this puzzle is presented on the manufacturer's homepage (Mr. Puzzle Australia) [6], in the Limited Edition section of the webpage, Limited Edition 2009. There is a ternary

1 Space complexity: Field in Complexity Theory (in Theoretical Computer Science) classifying decision problems for their computability within certain asymptotic space bounds.

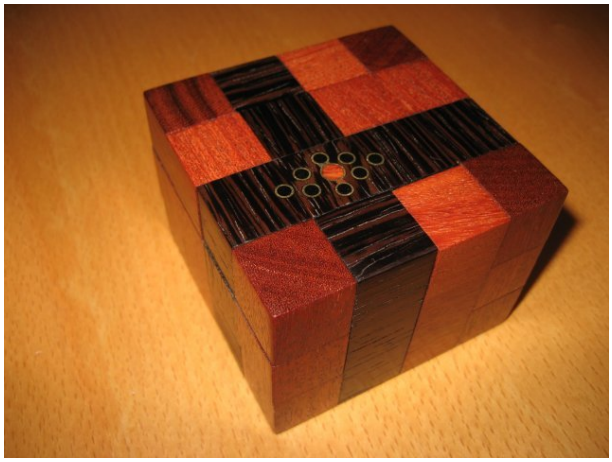
version of the above mentioned Cubi: The Super Cubi with 324 moves by Hiroshi Iwahara (Karakuri Group [10]). A much smaller ternary puzzle is the K-323 by Kim Klobucher [13], which seems to have a move sheme similar to the Super Cubi. There is another new Kcube puzzle, the K-419, which seems to have a quaternary move scheme and opens in 419 moves. A bigger version with an astounding number of moves is the MMMDXLVI below.
I am proud to have some of them in my private collection, and these are shown in the following picture: Ternary Burr (left), Crazy Elephant Dance (right), Tern Key (bottom) – in the starting position and after some moves showing the three different states a "ring" can have in each of the puzzles.



In the following picture two KCubes from my collection are shown, on the left the K-323 partially opened, on the right the K-419 (closed):



In July 2010 a new addition to this KCube series came out: the MMMDXLVI. It has a simple but very descriptive name – the name is just the number 3546 of moves to open the puzzle in Roman numerals. This "beast" looks quite innocent from the outside:



10 Mathematics behind these puzzles:

For most of these puzzles, easy recursive formulae can be derived for the solution length and may also be converted to an explicit form. Binary puzzles are related to Gray codes, binary codes of words of the same length, ordered in a way such that adjacent words differ exactly in one bit. The bits in these code words correspond to the ring positions of these puzzles. More details are discussed on Jaap Scherphuis' page [7], in the article about the SpinOut puzzle.

As we have a look at the Kugellager puzzle in particular, we will not discuss this here.

11 "Kugellager" – as a quinary puzzle:

The "Kugellager" can under this impression be counted as a quinary (5-ary) puzzle, for the following reasons:

- Each of the balls can be in one of 5 positions (ignoring the red circles once more).
- In each configuration of the puzzle there is exactly one move possible, and after moving the slider exactly one other move, two moves alltogether – leading to a "linear" solution without branching points.
- The movements of the slider and the currently moving ball are influenced by the lower and higher balls, where there is a homogenous scheme.

However, this scheme of restrictions on ball movements by other balls is not as simple and clear as for the other (binary/ternary) puzzles, which poses the question, whether we should really call this puzzle a "quinary puzzle".

12 "Kugellager 7" – a new world record of 4802 moves:

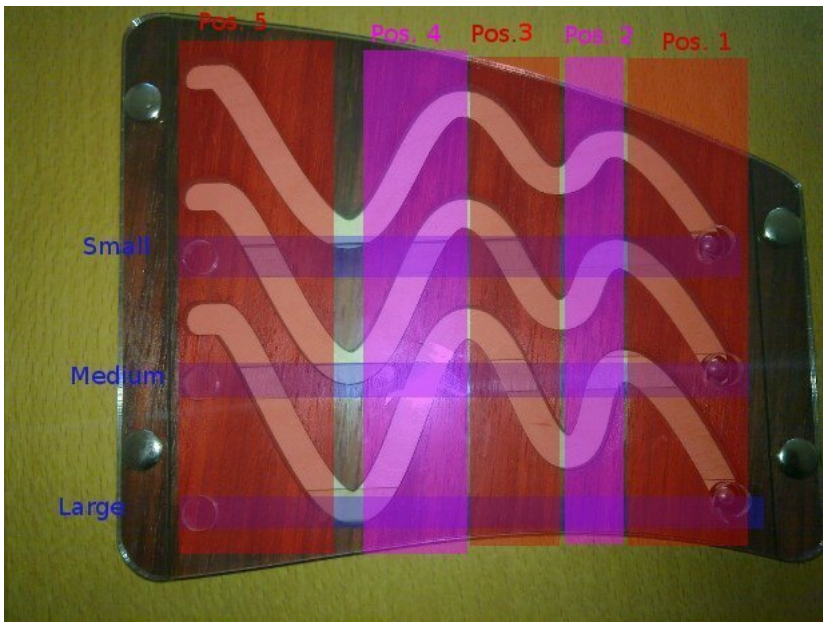
Following the Kugellager presented so far, Jean-Claude Constantin has created a larger prototype version of the puzzle. It still has 4 balls, but the base was raised from 5 to 7. Also, it does not have balls and an acrylic cover, but some sliders to be found in other of his puzzles as well. Actually, this makes the puzzle faster to be solved than if it hat balls to be moved by tilting. However, this advantage of faster movements is eaten up by the large number of moves to solve the puzzle – the interesting part of the puzzle. Using the formula from above, and by replacing base 5 by 7, getting $2*7^n$ for n balls, we can calculate the total number of moves to be solved, which is an astonishing $2*7^4=2*2401=4802$ moves. I used Pit Khiam Goh's program to verify and could confirm this number. Although it seems to be a puzzle of only theoretical interest, I have solved it completely already and it took me about 45 minutes. Here is a picture of this first septenary (7-ary) puzzle and the highest level puzzle in existence at the moment (both to my knowledge):

A closer look may pose the question whether this puzzle is really a base-7 variant of the original Kugellager. Compared that one, this puzzle is upside-down and the balls/pins would move from top to bottom instead from the bottom up. However, there are two variants of the Kugellager – one bottom-up and the other top-down – and the Kugellager is a base-7 variant of the latter one. For the number of moves, it does not matter which variant you choose and both types are equivalent.

13 Another truly quinary (5-ary) puzzles with balls

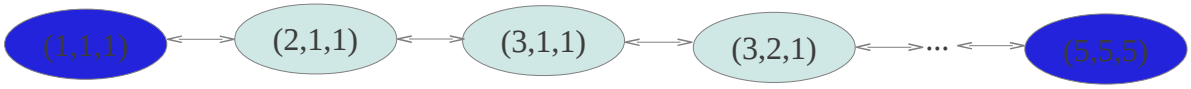
A newer puzzle by Jean-Claude is: "Die Welle" ("The Wave"), which does not seem to follow a strict scheme as the puzzles mentioned above. On the other hand, it seems a very interesting puzzle, as there are 3 balls in horizontal grooves, and some wavelike grooves on a vertical slider, and the balls move simultaneously directed by the slider. Only at some "decision points", you can move a ball into one of the two possible directions by tilting:

This was my first impression. A closer analysis showed that it is as close to a quinary puzzle as one can get, and in some way very similar to the Kugellager. There are three balls moving horizontally and acting as three "ring pieces", like the four balls of the Kugellager, and the puzzle has one slider (moving vertically on the picture above), like the Kugellager. What may be confusing at first, is the lack of clearly distinguished positions. Having a closer look, you will find out that there are different sections of the grooves that are separated by little "hilltops" of different heights. The following picture gives an overview of these different sections/positions:



The positions are named "Pos.1" to "Pos.5". All balls (denoted as "Small", "Medium" and "Large" in the picture) start in Pos.1 and the goal is to get all of them to Pos.5 at the same time.

To avoid some trouble counting the moves, this time I employed the help of a computer program. I did a similar analysis to the one for the Kugellager puzzle (Chapter 5) and created a graph of configurations: Each vertex of the graph is a configuration denoting the current position of each ball. Each edge of the graph is a valid transition, describing the move of one or more balls. If you denote each configuration by: (s,m,l) – a tuple of three ball positions, the graph would look like:



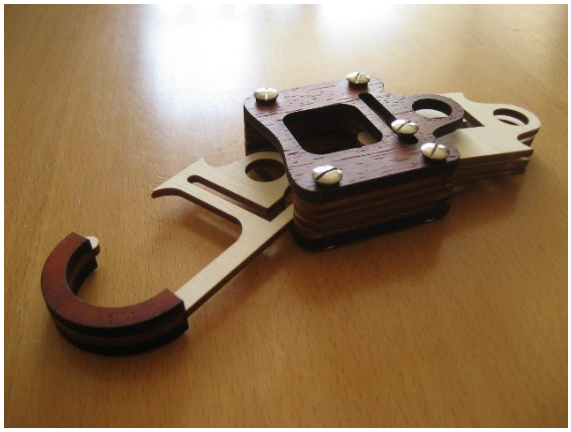
For building this graph, I only checked for each ball transitions, where all "lower" and "higher" balls have to be located. The rest is done by my program and it turns out to be a deterministic puzzle, with only one next move forward as shown in the graph above.

After building the graph, it is easy to determine the shortest solution and its length by simply doing a breadth-first-search in this graph for the shortest path from (1,1,1) to (5,5,5). It turns out that all configurations are needed for the solution, leading to a solution of 124 steps. This coincides with S(5)=124 of Chapter 5. Since the ball and slider moves for this puzzle occur simultaneously, unlike the Kugellager puzzle, where ball and slider moves happened in an alternating fashion, it is no surprise that "Die Welle" has the same number of moves, as a Kugellager with only one of the move types, e.g. slider moves.

14 Short big Sliding Lock without balls

A new puzzle introduced to the autor on the international games fair "Spieletage 2010" in Essen in October 2010 by Jean-Claude Constantin is a rather small puzzle, which looks like a pad lock with some internal sliders. In fact, it is a close relative to the Kugellager puzzle and "Die Welle", and is a quinary (5-ary) puzzle with three ring pieces and alltogether 125 ring piece moves, plus an additional 125 moves of a sliding screw. Unlike the Kugellager, the ring pieces are itself sliders now, and instead of the Kugellager's slider, there is a screw reaching trhough grooves in all three ring pieces, and the shackle-piece. After all three ring pieces have been slid from starting position 1 to goal position 5, the screw can be slid to the very end of the groove in the lock and the piece connected to the

shackle can be pulled out. The following pictures show the puzzle in the starting configuration, a configuration in the middle of solving, and the goal configuration with the shackle removed.



15 A new puzzle similar to the ones presented above:

There is also another puzzle by Jean-Claude Constantin which follows a similar scheme as some of the others mentioned above. In this sliding lock, the lowest "ring piece" only has two different states and acts like the Kugellager's slider, while the others have six positions – a generalization of e.g. the ternary puzzles. Alltogether, a little over 100 moves are used to solve this lock:

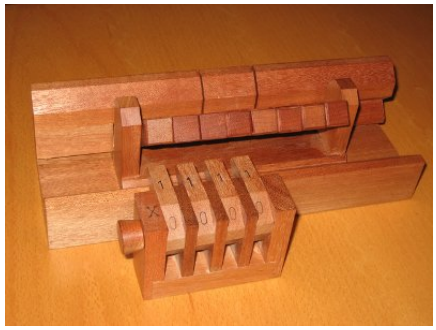
Looks like we might have a senary (6-ary) puzzle here, but that has yet to be confirmed.

16 Some new finds:

There are some recent finds that deserve an own chapter. The first three of these are not in my collection (all are on my wishlist) which explains the lack of pictures. The first one to mention is a tiny chest with four drawers, the "**Bin Laden**". The name says it: It is a binary puzzle and you have to know which drawer to move in and out during the solution. As a

reward, you will get some Euro cent coins from the drawers. This was an IPP exchange puzzle by Rik van Grol.

The next one is also an exchange puzzle designed in the Netherlands: The "**Mysterians**" designed by Oskar van Deventer and exchanged by Nick Baxter in 2003 at IPP23. As it turns out, this puzzle is equivalent to the "Die Welle" above. It has three moving pieces (="balls"), each has five regions on each piece, and for the solution, all 125 states are required. Hard to believe when you only see the picture to the right (provided by Nick Baxter). The full exchange puzzle documentation (see [17]) on Oskar's page, however shows the five regions per piece and a cubic graph for the solution. The solution path is a Hamiltonian path through the solution graph from start to end state, which (by definition) runs through all states and uses all available moves. Historically, this is the oldest 5-ary puzzle in existence, existing more than five years before the Kugellager.



The last one is a puzzle that came into my collection quite recently and that has been noted in Chapter 9 above, the "**Hexadecimal Puzzle**" designed by William Keister. Originally, this puzzle was produced in 1985 as one of the first puzzles by "Binary Arts" (now "Think Fun"), but these originals are hard to find nowadays. At the moment, the Australian Bill Wylie creates very few copies of a remake of this puzzle which allowed me to study the Hexadecimal puzzle and provide some insight of it's binary nature. There is much documentation on the web about this puzzle, so only a short overview will be provided: It has a central slider with 8 switch pieces on it. These switch pieces can be flipped between two positions, namely inside a channel in the back block of the puzzle, or above. At the beginning all eight are in the channel and for removing the slider (and hence solving the puzzle) all have to be flipped to sit on top of the block. However, fipping a switch is only possible at one position in the back block and to a allow this, a small block in the front of the puzzle has to be pushed back. This small block carries four bit switches that are labelled "0" and "1" and which interact with some of the switches and do not always allow to move this block. Before solving the puzzle, the solver has to select the challenge and set these bits according to his selection. After this initial setting these bits remain untouched until the puzzle is solved, only the slider and it's switch pieces are moved. There are four bits, adding up to 16 different possibilities of varying challenge level – hence the name "Hexadecimal puzzle". The challenge "1110" is a binary puzzle and requires the longest solution sequence of all the challenges.

17 Auf dem Holzweg

The last is a recent novelty by Siebenstein: The puzzle "**Auf dem Holzweg**". It very much lokes like a Kugelllager where the slider has been cut into four parts, one for each of the channels, and the balls have been replaced by rivets that each go through two of the slider parts. The four rivets are connected so that they can only be moved simultaneously. So, it's a Kugellager the other way round: The "balls" move simultaneously and over the same

distance, the channels on the slider parts are moved so that the "balls" are on the correct level. Last notable thing is that the channels have six levels, so this puzzle seems to be equivalent to a level 6 Kugellager with four balls. The German name is a very good description of the puzzle: "Holz" means "wood", and "weg" translates into "way", which together describes the wooden puzzle with the channels in it. However, there is also a proverbial meaning of the name: If someone is "auf dem Holzweg", it means that someone has taken the wrong turn and is walking the completely wrong direction, or is using some flawed reasoning that will lead to the wrong conclusion.

In fact, this is a very good name, as I had to find out: Just after playing with the real puzzle, the deception induced by the pictures was uncovered. The puzzle is **not 6-ary**, but a new type. It combines two halves of a 6-ary puzzle and consists of **two ternary (3-ary) parts**. In the picture shown to the right, the upper part shaded in yellow is the first ternary puzzle, the bottom part shaded in green the second ternary puzzle. After solving the yellow part, the two rivets have to go through the channels highlighted in blue simultaneously to continue with the green part. If you have a close look at these blue parts, you will see that they exactly overlap to allow this transition from yellow to green.

The number of moves m(n) for a "Holzweg" with k sliders can be calculated by using the known formula for each part and addition:

$$m(k) = 2 * 3^k + 2 * 3^k = 4 * 3^k$$

For the actual puzzle in the picture we have **m(4)=4*81=324** moves. The puzzle is ternary in some way, but as a combination of two ternary puzzles, **double-ternary** might be the right notation.

18 The beginning – the Kugellager ancestor

Quite recently, Michel van Ipenburg (a true Kugellager enthousiast) pointed me to an ancestor of the Kugellager. It is a disc puzzle from 1913, so Kugellager history has been going on for quite a while, although there are no proven connections between this puzzle and the recent n-ary Kugellagers. The puzzle is disc shaped, has a cross on it and four rivets which are equivalent to the balls of the 5-ary Kugellager. A picture of the puzzle "Cross and Crown" can be seen at [18] and the original patent US1071874 by Louis S. Burbank from 1913 at [19]. It's not as old as the Chinese rings, but the oldest puzzle with a strong resemblance to the original Kugellager.

19 Overview of puzzles by type:

The puzzles presented so far can be grouped by their "arity" (including assumptions):

Arity	Puzzle Examples
2 - binary	Chinese Rings, SpinOut, The Brain, Binary Burr, Key Puzzle, Cubi, Barcode Burr, Hexadecimal Puzzle
3 - ternary	Crazy Elephant Dance, Tern Key, Ternary Burr, Super Cubi, K-323, Loony

	Loop, Sliding Block Puzzle, Auf dem Holzweg*
4 -quaternary	K-419, MMMDXLVI
5 - quinary	Kugellager, Die Welle, Short big sliding lock, Mysterians, Cross and Crown
6 - senary	Sliding Lock
7 - septenary	Kugellager 7

* = double-ternary

20 Conclusion:

The "Kugellager" is a very interesting puzzle, may it be a genuine quinary puzzle or not. If you have a recursion scheme and a constructive proof for the number of moves, I would be very interested to see your considerations and calculations.

21 Notes and Remarks:

My mail address and a gallery of my whole puzzle colletion [1] including many interesting items – also most of the ones mentioned above – can be found via my homepage, or directly via:

http://puzzles.schwandtner.info

Although I checked the information presented in this paper, it might well be the case that there are incorrect details. If you run over such a flaw, please let me know. The current version of this paper is always hosted on my homepage [2].
The pictures in this paper have been created by the author. If you would like to use them, please feel free to ask.

22 Acknowledgements:

I would like to thank the designer and manufacturer of this nice puzzle: Jean-Claude Constantin. However, his homepage [3] does only include older puzzles and does not contain any reference to the Kugellager or Sliding Lock. My thanks go to Dan Feldman (inventor of the very interesting DanLock puzzle lock, which includes several creative ideas and seems to be the most interesting puzzle lock to me) for providing valueable feedback on drafts of this document, and for the nice discussion about the puzzle. While we were discussing this puzzle I had the idea to take a closer look at "Kugellager" and to write this short paper. Thanks to Nick for the recent discussion about the "Mysterians". Thanks for Michel to find and point out the ancestral Kugellager roots.

23 Updates:

This article will be extended when knew facts and ideas come up. Here is a short overview list of changes:

27/02/2010 – Recounting of moves; additional functions S(n), L(n), estimation

20/03/2010 – Added some puzzles, recounted moves – should be correct now

06/06/2010 – Added three more puzzles, from KCube and "Die Welle"

18/09/2010 – Added new KCube and Loony Loop, Sliding Block

17/10/2010 – Bugfixing

02/03/2011 – Added pictures of Small Cubi

22/05/2011 – Added chapters for "Die Welle" (with program) and "Big short sliding lock"

23/10/2011 – Added chapter for new "Kugellager 7"

16/11/2011 – Minor bugfix for table on p. 4

04/12/2011 – Added pictures of Binary Burr

26/02/2012 – Added chapter for "Some new Finds"

20/03/2012 – Added chapter for Kugellager ancestor

23/04/2012 – Added separate chapter for "Auf dem Holzweg" (removed old paragraph)

24 References:

[1] My puzzle gallery: http://puzzles.schwandtner.info
[2] My home page: http://www.schwandtner.info (both German and English)
[3] Jean-Claude Constantin: http://www.constantin-jean-clau.de
[4] Jerry Slocum, Jack Botermans. Puzzles Old and New – How to Make and How to Solve Them. Village Games. 1999 (Third Paperback Edition)
[5] Markus Goetz (Crazy Elephant Dance): http://www.markus-goetz.de
[6] Mr.Puzzle (Ternary Burr): http://www.mrpuzzle.com.au
[7] Jaap's Puzzle Page: http://www.jaapsch.net/puzzles/
[8] Bill Cutler's Page: http://home.comcast.net/~billcutler/
[9] Key Puzzle on Bill's Page: http://home.comcast.net/~billcutler/stock/key.html
[10] Karakuri Creation Group: http://www.karakuri.gr.jp/
[11] YouTube-video showing some of Lee Krasnow's puzzles: http://www.youtube.com/watch?v=OH9JhRalzoY
[12] Lee Krasnow's homepage: http://www.pacificpuzzleworks.com/
[13] KCube Designs: http://kcubedesigns.com/
[14] Elwyn R. Berlekamp, John H. Conway, Richard K. Guy. Winning Ways for Your Mathematical Plays. A K Peters, Wellesley, Massachusetts. 2004.
[15] Sliding Block Puzzle: http://www.hearn.to/block-rings.pdf
[16] Robert A. Hearn. The complexity of sliding block and plank puzzles. In Tribute to a Mathemagician, pages 173-183. A K Peters, 2004.
[17] Exchange puzzles 2003 by Oskar: http://oskarvandeventer.nl/2003-exchanges.html
[18] Cross and Crown on Rob's puzzle page: http://www.robspuzzlepage.com/routefind.htm#mazemulti
[19] Patent for Cross and Crown: http://www.google.com/patents/US1071874

25 Appendix: Program Source for "Die Welle":

The program is written in Python and can be run after downloading and installing the Python interpreter (at least version 2.5) from: http://www.python.org

To run it, simply copy the text below into a file, e.g. named WelleSolver.py with a plain text editor and run the program file with: `python WelleSolver.py`

The output will be purely in a textual form, showing the solution path and the length of the solution.

```
#!/usr/bin/python
# WelleSolver.py -- python program to solve the puzzle "Die Welle" by Jean-Claude Constantin
# This program was created by and is (c) by Goetz Schwandtner 03/2011
# Reference: http://www.schwandtner.info
class ConfAdj:
          """Configuration graph of puzzle as adjacency matrix"""
          def __init__(self):
                    # dimensions of the configuration graph
                    self.numPositions= 5
                    self.numDimensions= 3
                    self.numConfigurations= self.numPositions**self.numDimensions
                    # transitions - lower[i] contains all allowed positions for lower balls, for transition i <-> i-1
                    self.lower= [ [3,4,5], [1,4,5], [5], [1] ]
                    # transitions - higher[i] contains all allowed positions for higher balls, for transition i <-> i-1
                    self.higher= [ [1,2,3,4,5], [1,2,3,4,5], [3,4,5], [1,4,5] ]
                    # configuration graph of puzzle represented as symmetric adjacency list
                    self.confAdj= []
                    # call data structure initializer
                    self.setupConfAdj()

          def setupConfAdj(self):
                    """internal helper function to set up adjacency list"""
                    for conf in range(self.numConfigurations):
                              self.confAdj.append([])
                    for conf in range(self.numConfigurations):
                              # iterate over all neigbors and check for edges (i.e. transitions)
                              # determine coordinates (i.e. positions of sm, md, lg ball)
                              (psm, pmd, plg)= (conf % self.numPositions, conf / self.numPositions % self.numPositions, \
                                                  conf / self.numPositions**2)
                              # smallest ball transition -- possible/valid?
                              if (psm < self.numPositions-1) and ((pmd+1) in self.higher[psm]) and ((plg+1) in self.higher[psm]):
                                        # add this transition (and reverse) to adjacency list
                                        self.confAdj[conf].append(conf+1)
                                        self.confAdj[conf+1].append(conf)
                              # medium ball transition -- possible/valid?
                              if (pmd < self.numPositions-1) and ((psm+1) in self.lower[pmd]) and ((plg+1) in self.higher[pmd]):
                                        # add this transition (and reverse) to adjacency list
                                        self.confAdj[conf].append(conf+self.numPositions)
                                        self.confAdj[conf+self.numPositions].append(conf)
                              # largest ball transition -- possible/valid?
                              if (plg < self.numPositions-1) and ((psm+1) in self.lower[plg]) and ((pmd+1) in self.lower[plg]):
                                        # add this transition (and reverse) to adjacency list
                                        self.confAdj[conf].append(conf+self.numPositions**2)
                                        self.confAdj[conf+self.numPositions**2].append(conf)
          def printConf(self, conf):
                    """returns a configuration as user readable tuple, with value 1-5"""
                    (psm, pmd, plg)= (conf % self.numPositions, conf / self.numPositions % self.numPositions, \
                                        conf / self.numPositions**2)
                    print "(",plg+1,",",pmd+1,",",psm+1,")"

class BFSSolver:
          """BFS solver to solve for the shortest solution and the above type of graph (ConfAdj)"""
          def __init__(self,adjList):
                    self.adjList=adjList
                    # predecessors -- shortest paths found by BFS so far
                    self.pred= [-1]*(adjList.numConfigurations)
                    self.queue= [0]
          def run(self):
                    """do a BFS (breadth-first-search): as long, as there are configurations in the queue, pick the first and work"""
                    while len(self.queue) > 0:
                              conf=self.queue.pop(0)
                              # check all neighbor nodes of this one (conf) and queue them if not yet visited
                              for nbr in self.adjList.confAdj[conf]:
                                        # not yet visisted
                                        if self.pred[nbr] == -1:
                                                  # conf will now be predecessor
                                                  self.pred[nbr] = conf
                                                  self.queue.append(nbr)
                                                  # special case: if we have found last one, we're done
                                                  if nbr == self.adjList.numConfigurations-1:
                                                            return
          def createPath(self):
                    """return the path found in run method"""
                    # start with maximum index (goal element) and go back
                    self.path= [self.adjList.numConfigurations-1];
                    while self.path[0] > 0:
                              self.path.insert(0, self.pred[self.path[0]])
                    return self.path
          def printPath(self):
                    self.createPath()
```

```
                    # do the actual printing
                    for conf in self.path:
                            self.adjList.printConf(conf)
                    print "Solution has ",len(self.path)," steps"

# main program

print "Solver for Jean-Claude Constantin's puzzle 'Die Welle' by Goetz Schwandtner"
print "Program (c) by Goetz Schwandtner 03/2011"
print "Website: http://www.schwandtner.info"

# initialize adjacency list
adjList= ConfAdj()
# print adjList.confAdj

# initialize solver
solver= BFSSolver(adjList)
solver.run()
solver.printPath()
```