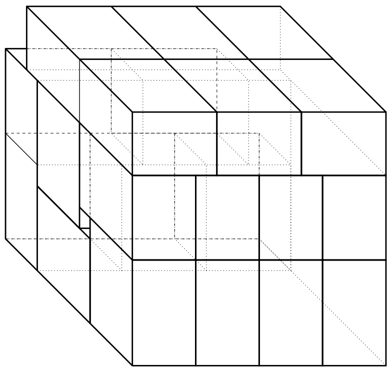
JUNIOR INDEPENDENT WORK, PART 1
SPRING 2003

ALEXEY SPIRIDONOV

This paper represents my own work in accordance with University regulations.

1. Introduction

Dean Hoffman proposed the following block-packing problem at a conference at Miami University in 1978. [1] The task was to fit 27 $a \times b \times c$, $a < b < c$, blocks into a cube with side $a + b + c$. As a simple way of preventing trivial solutions, he stipulated that $4a > a + b + c$; otherwise, one can stack blocks with four of the a sides in a row, like this:



We will also stipulate that the structure be rigid and stable[1]; a more concrete statement:

Requirement 1.1. If one puts the cube made of blocks inside a hollow $a + b + c$-cube, there will be no moving parts. (no friction, perfectly made blocks)

Thus, for every block B, there is a line in the x, y, and z directions passing through b, and some other blocks, such that their total length in that direction is $a+b+c$[2]. Hoffman's condition specifies that there be at most 3 blocks in such a line. It also cannot be 2: $a \geq \frac{a+b+c}{4} \Rightarrow c \leq 3a-b < 2a < a+b$, so $2a < 2b < 2c < a+b+c$ all result in a loose puzzle. Thus, the puzzle can be viewed as a 3×3 grid as shown in Figure 1.

[1]Actually, this is probably not necessary; however, it seems tricky to show that no loose packings exist. Moreover, a loose solution is very difficult to work with physically.

[2]We should note, for the sake of amusement, that blocks, being physical objects, are open sets — infinitesimal overlaps of faces, through which such a line might pass, are not tolerated.

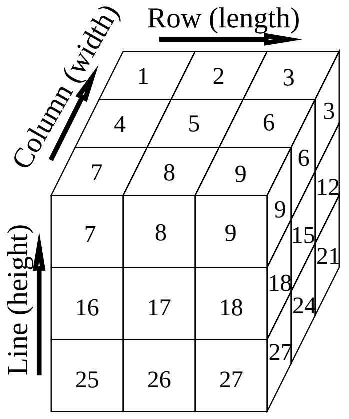


FIGURE 1. The definition of rows, columns, and lines in our usage, as well as the numbering scheme we use on a cubic grid.

Consider the cube as 3 layers of 9 blocks each. (Layers may be taken in any orientation, but $[1-9]$, $[10-18]$, $[19-27]$ can be examples.) From now on, we will call these layers *squares*. Figure 2(i) contains a view of one possible square from above; notice that the hidden dimension is trivial to deduce, we will therefore follow this notation. We will also use this numbering schemes for future squares, even though the blocks may not be explicitly labeled.

Of the 9 ways $a+b+c$ could be written as another sum of those lengths, none are valid: $3a$ is too short; $3c$ – too long; $2a+c$ is too short (by $b-a$), $2a+b$ and $2b+a$ are similarly so; $2b+c$ is too long (by $b-a$), so are $2c+a$ and $2c+b$. The only one that might possibly fit is $3b$. However if some row (wlog) is composed of $3b = a+b+c$, the cube consists of 9 rows, each of which has at least one b-dimension. Then, they use up 11 b-dimensions. However, each line and column must also have at least 1 b-dimension, for a total of 9+9+11=29 required to complete the puzzle. This is impossible, so we must have exactly the sum $a+b+c$ in every row (column, line). Figure 2(ii)-(iv) shows some invalid packings; the constraint we just formulated eliminates (ii) (although it is also loose). (iii) illustrates that, while the orientations and relative positions of the blocks may satisfy the new requirement, it may be impossible to pack them into the allotted volume. Hence, the complete constraints on a solution are:

Requirement 1.2. In every row (column, line), there are 3 blocks with distinct length (width, height), and the blocks are touching each other.

Figure 2(iv) illustrates what happens if the condition $c > a+b$ is violated – a valid square (the same orientations as in (i)) becomes loose. The condition $4a > a+b+c$ prevents this problem, of course. However, solutions satisfying Req. 1.2 work independently of a, b, c as long as the weaker condition is satisfied[3].

So, we can treat the puzzle simply as a problem of setting the blocks at the 27 grid positions to the correct orientations. Call the set of orientations $\mathbf{O}$ =

[3]Stated without proof for lack of time, but believed to be true; consider a conjecture, if you wish.

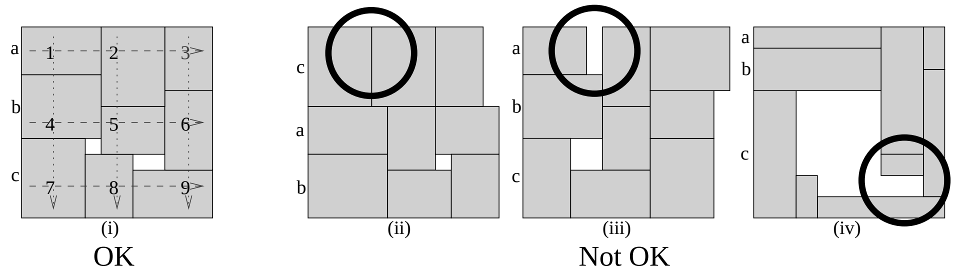


FIGURE 2. A square from a valid solution of Hoffman's cube puzzle (i) and some invalid squares (ii) - (iv). The rows and columns in (i) are labeled with dashed/dotted arrows, and a numbering scheme is presented. Circles mark the points of failure in the three invalid squares.

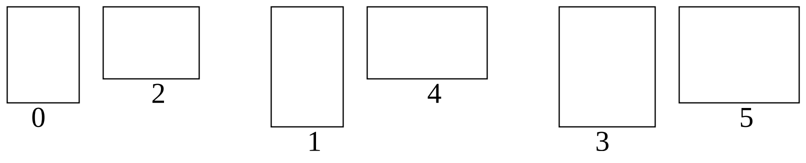


FIGURE 3. Numbering scheme for the orientations

$\{0, 1, 2, 3, 4, 5\}$; our (arbitrary) mapping of these numbers onto the orientations is shown in Figure 3.

Definition 1.1. A *solution* of the Hoffman cube puzzle is a 27-tuple of $o_i \in \mathbf{O}$ with the following properties:

(1) If o_i and o_j are in the same row (column, line), their length (width, height) are different. The formal conditions for being in the same row, column, line are trivial; e.g. $i \equiv j \pmod 9 \Leftrightarrow o_i, o_j$ in the same line.
(2) Every square of the puzzle is a valid square: the blocks in each row and column touch each other, unlike in Figure 2(iii). This is also trivial to formalize (see explanation in section 2).
(3) Surprisingly, the preceding conditions are not enough; we will discuss the significance of this shortly. One must also require that when placed into a cube structure, there are no overlaps of just the corners of blocks (2 takes care of non-corner overlaps). Some cases will be presented later.

Since the puzzle has a very simple formalization, several people have previously written computer programs to enumeratively solve it. [2] An interesting feature of these solutions is that they all omitted (3) of Def. 1.1, and neglected to manually check all solutions. As a result, they found 26 (up to symmetries), of which only 21 could be constructed. This turned out to be fortunate, since the 26 solutions have a nicer structure than the 21; we will discuss this in more depth.

The puzzle has an interesting algebraic interpretation. If one constructs a solution, one sees that there are holes in the cube (which would go away if we allowed $a = b = c$, of course). Hence, $(a + b + c)^3 \geq 27abc \Rightarrow \left(\frac{a+b+c}{3}\right)^3 \geq abc \Rightarrow \frac{a+b+c}{3} \geq \sqrt[3]{abc}$, or the arithmetic mean of 3 numbers is no less than their geometric mean.

This argument easily generalizes past the extra constraint on a, b, c — when the differences in dimensions are great, the holes are bigger (see Figure 2(iv)).

This paper presents yet another solver for the puzzle, written with the intent of minimizing the search, so that the same techniques could be applied to solve an extension of the puzzle to the four-dimensional case. We show the 26 pseudo-solutions, identifying the 21 real ones, and discuss their structure. The four-dimensional case, was, in fact, the main subject of the investigation. We discuss our formulation and results on that topic in a section 4.

2. Solving Hoffman's Cube Puzzle

The first thing to note about the problem is that the pure brute force approach will not work; a rough estimate of the time it would take to enumerate the 6^{27} cases on a modern personal computer is 2 million years. Of course, such a gross approach is unnecessary.

The general method in such search problems is to construct solutions step-by-step, discarding infeasible attempts as soon as they can be identified as such. Thus, the order, in which the solution is constructed is essential. The second principle is to avoid performing the same calculation twice. The third, but no less important, is to avoid unnecessarily complicated search strategies.

The first major constraint is that no row contain elements with the same length. Finding all possible rows is trivial and instant, even by brute force (6^4). From now on, we can work in units of rows, and thus avoid re-calculating this constraint. The next constraint is when two adjacent rows have corresponding blocks of the same width, or cannot be placed together without overlap. The former is trivial, while the later can be formulated in a concise manner. Suppose we are contemplating the placement of block B. There are two cases when this is prevented, as shown in Figure 4. Notice that there may be another row above, and column to the left: e.g., we might be trying to place block 8 below $[1-3]$, $[4-6]$, $[7]$ in Figure 2(i). In that case, $l1, l2, w1, w2$ signify the distances from the respective edges of the cube. The two formal conditions for forced overlap are: for (a) — $(w1 > w2) \wedge (l1 > l2)$, for (b) — $(w1 < w2) \wedge (l1 < l2)$. Notice that if we consider in (a) the block B', the variables $l1, l2, w1, w2$ in both cases will have the same meaning. Additionally, note that if B (B' in (a)) is the 3rd block in the row, neither of these conditions can happen. Thus, the final check for the compatibility of two rows need only be done for the first two blocks, and has this form: $((w1 > w2) \wedge (l1 > l2)) \vee ((w1 < w2) \wedge (l1 < l2))$. If a potential solution satisfies this constraint for every pair of adjacent rows (columns/lines), and also passes the distinct length (width/height) requirement (further *distinct lwh*), it is one of the 26 pseudo-solutions.

We now return to the search strategy. Having constructed all rows, we proceed to construct all squares, since that takes maximal advantage of distinct lwh in a second dimension. If it were not done in advance, the search would have to perform a subset of the construction 3 times — once for each square of the puzzle. The calculation of all squares can be broken down further: once the first row is selected, the remaining two need only be selected from those, which do not violate the lwh requirement. Hence, we can precompute those squares. Because the spatial constraint depends on all the previous rows, doing a similar precomputation is probably not possible (or very difficult). An analogous precomputation can be done when building a cube out of squares.

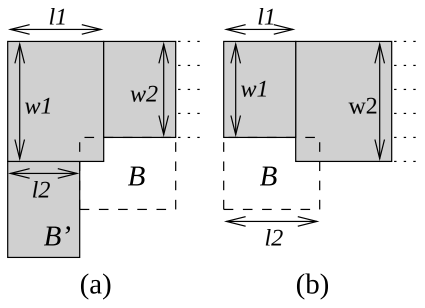


FIGURE 4. Conditions for row incompatibility.

We implemented a solver using these ideas (see source of `solve3D.c`, refer to Appendix A). Since a factor of 48 is inconsequential in this case, it does not take care to limit its search to non-symmetric variants. For the sake of simplicity, it also neglects to check square-compatibility in the reverse order (as in, going from $[19-27]$ to $[10-18]$ to $[1-9]$; this leads it to produce some non-pseudo-solutions; however, they lack their full set of symmetries. For these reasons, there is a second program (`symm.c`), which discards the latter, and also removes all but one of each full sets of symmetric solutions. It reduces the output of `solve3D.c` (1332 solutions) to 26. Finally, there is a program (`drawsoln.c`) which produces the representation of the cubes in Figure 6, and detects corner overlaps to mark real solutions distinctly from pseudo-solutions. The three programs take about 80 milliseconds to run on a Pentium 4 1.7 GHz, and do not use any substantial amount of operating memory (RAM); that is quite an improvement as compared to 2 million years.

3. The 26 Solutions

Definition 3.1. The *dual* $S' = (e'_1, e'_2, \ldots, e'_{27})$ of some solution $S = (e_1, e_2, \ldots, e_{27})$ has e'_i such that the orientations of the a and c dimensions are swapped. In our numbering scheme, $e'_i = 5 - e_i$.

The dual is an interesting operation because not only does it preserve distinct lwh, but also keeps all squares valid (not difficult to argue, even easier to validate enumeratively, which we have done). So, the dual of a pseudo-solution is again a pseudo-solution. Hence, we made `drawsoln.c` group solutions by duality, as can be seen in Figure 6. Duality, and validity break the solutions down into 4 categories: 1 valid and self-dual, 16 valid ones with distinct valid duals, 4 valid with distinct invalid duals (and the reverse), 1 invalid and self-dual.

Another notable feature is that the configuration marked in light gray occurs only in invalid pseudo-solutions. The configuration coerces one of two squares (consider solution 3 – the configuration coerces block 1, after then 4, then 6, and leaves one choice in the final row, compare solutions 3 and 10). This is of help in identifying or remembering the invalid pseudo-solutions. However, square 25 does not contain this square in any orientation. Another interesting regularity is that every valid square must contain 3 instances of orientation 0 and 2, 3 of 3 and 5, and 3 of 4 and 1. This is a consequence of a counting argument like the one given in section 1 to show why 3 b-lengths could not be in one row. In the same way, one can easily

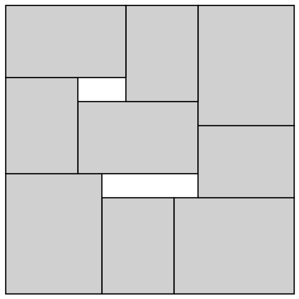


FIGURE 5. An unstable square

argue (once we have limited ourselves to a 3×3 grid) that unstable squares (as in Figure 5) cannot be in a solution.

Blocks 2,3,5,6 in the top square, solution 1, form a configuration that is ubiquitous in valid solutions; solutions 2, 5, 11 are not counterexamples – they have it in another orientation. Awareness of this fact (and the possible positions of this configuration) should make finding a solution by hand considerably easier. There are probably many other interesting regularities, but covering them all is beyond the scope of this paper.

4. Approaching the Four-Dimensional Case

The four-dimensional problem is just a direct extension of Hoffman's puzzle to a $4 \times 4 \times 4 \times 4$ grid with $a \times b \times c \times d$ blocks, $a < b < c < d$. For simplicity, rather than extend his condition, we assume the sizes differ by a very small amount, say 99, 100, 101, 102. The same formalization, and similar numbering schemes can be used in this case. Here, the brute force approach is considerably more hopeless: 24^{256} computations would take far longer than the age of the universe to complete. That does not deter us, however.

The strategy described for building squares translates to the 4-dimensional almost directly. For the purposes of constructing them, we may disregard the 3rd and 4th dimensions, and thus reduce the number of orientations to 12. There are a total of $4!3^4 = 1944$ rows with this restriction (if we added the other dimensions, the number would grow by a factor of 2^4) In the 4D case, we may beneficially precompute compatible rows twice, rather than just once. All the squares take a few minutes to compute, resulting in a total of 77436138. Adding the other dimensions increases it by a factor of 2^{16}. The cube search strategy outlined in section 2 is at least cubic in the number of squares: we must try $2^{16-3}77436138$ options for the first square (discard a factor of 8 for a square's symmetries, something we cannot do in other layers). The odds that a choice of the 3rd and 4th coordinates in the first will coerce it in the 2nd are $\frac{1}{6}$, so can expect roughly $2^{13\frac{1}{3}}77436138$ choices for the second layer. In the third layer, only $\frac{1}{6}$ of the choices for the third dimension are non-coerced; in addition, $\frac{1}{6}$ of the coerced blocks will fail distinct lwh. So, 77436138 appears a fair conservative estimate for the number of choices in the third layer. The fourth layer is very severely constrained both by distinct lwh considerations, and by spatial requirements; enumerative testing would probably be replaced by a fast directed search. This argument disregards the portion of choices

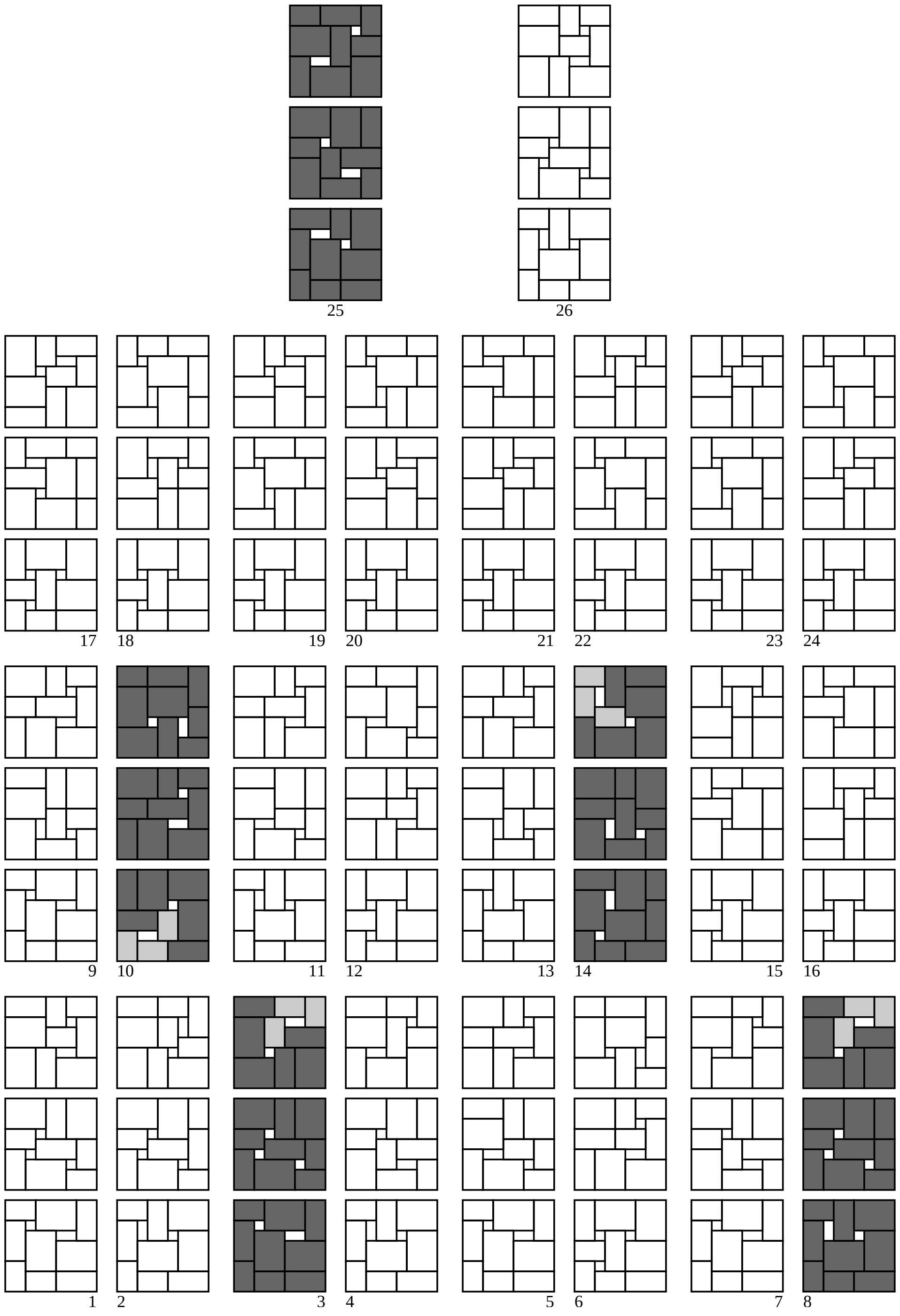


FIGURE 6. Solutions to Hoffman's cube puzzle. Ones invalidated by corner overlap are shaded. Solutions are paired by duality: 1 and 2 are dual, 3 and 4, etc. 25 and 26 are self-dual.

that is discarded by spatial constraints; however, these cannot be precomputed, so the reduction is comparable to a constant factor c (we may throw out all but a fraction of choices for layer 2, but we still have to test all the layer 3 choices as computed above). So, the number of computations we have to perform is on the order of $77436138^3 2^2 6c^2$; As such, it is probably even beyond the reach of current distributed computing projects. One should beware, however, that these estimates disregard the possibility of the problem having a non-uniform structure in some respects; this can affect the search considerably. The same disclaimer applies to our next attempt.

An improvement is possible that uses the spatial constraints to prune the search earlier, and organizes the data for effective searching. While generating the allowed squares, it is a simple and effective matter to categorize them by several of their rows (adding no more than a few minutes to the task). This allows us to instantly retrieve squares containing a specified row or rows in specified positions. We must pick an initial (orientation-insensitive) square, but we can refrain from defining all the heights of the blocks. Instead, choose just two adjacent sides, producing $2^{7-3}77436138$ options. Now, we can select a square based on one of the two rows we filled in. We expect there to be $77436138/1944$ such squares on average. (each row has some associated squares, so there is no value in postponing the choice of the 3rd and 4th coordinates for the second row) The result is in Figure 7(a). Next, we choose a third square adjacent to the two already chosen. For that, we need to define the last coordinates for 3 more blocks, increasing the computation by a factor of $2^3 77436138/1944^2$. We can now define 6 more 3rd coordinates, as in Figure 7(b). That adds a factor of $2^6 77436138/1944^2$. To add the next face, we need to fully define 8 more blocks, and search all squares constrained by 3 rows. Most of these squares do not exist, so the search would be considerably pruned at this stage; on the assumption of uniformity, this stage adds a factor of $2^8 77436138/1944^3$. We might fill in the remaining face in this manner, and use the now-strong spatial and distinct lwh constraints to complete the cube directly, or repeat the procedure several more times. We might also at some stage go back to layer-by-layer completion. Disregarding non-uniformity, and the inevitable pruning of the search at the later stages, the computational cost of getting to a cube with the edges defined completely, faces partially defined (2 steps after Figure 7(c)) is $\frac{77436138^6 2^{29}}{1944^{12}} \approx 2^{55}$. This is, or will shortly be accessible to distributed computing projects. With further refinements (also computing half-squares – 531726 according to untested computations, and then computing half-cubes seems promising), it may be possible to run this on a single PC. We have implemented the search up to the stage shown in Figure 7. The running time appears to be consistent with the theoretical estimates. Due to time considerations, further implementation has not been pursued; our development program is called `cubesearch4D.c` and is documented in Appendix A.

Depending on the number of cubes found, all the solutions may or may not be feasible to enumerate. However, given a cube, the task of completing it to a hypercube is very constrained – there are no undefined dimensions, so the approach suggested above for building cubes from squares should be even more powerful. So, as long as the cubes are few enough to store and access effectively, we expect that completing a cube to a hypercube will be fast. This is true, since we can presently only store about 3 orders of magnitude more than the number of squares – 1000

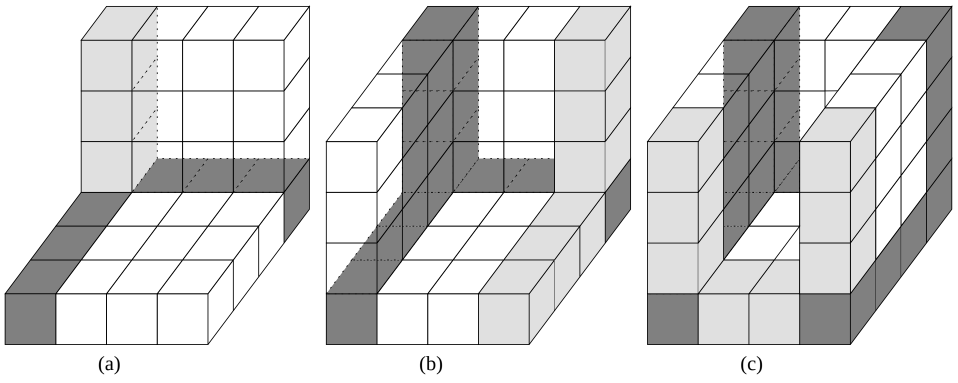


FIGURE 7. Construction of cubes for the four-dimensional puzzle. Fully defined blocks are marked with dark gray, partially defined ones are white or transparent. Light gray are the next to become fully defined.

cubes per square, on average. Hence, the search would be no bigger than 10^{12} (if we just test all the cubes starting with each of the 4 squares in the given cube). 10^{12} simple calculations would take between a fraction of an hour and a day on a modern PC. We have some reason to believe that the number of cubes is not excessive, based on the growth patterns observed in a similar, but easier problem (results not shown).

5. Conclusion

We repeated the work to solve Hoffman's cube puzzle, producing a very fast program for doing so. We have also made some observations about the solution set that, as far as we know, are new. We developed some strategies and running-time estimates for the four-dimensional variant of Hoffman's puzzle. Although we have been unable to solve it, our calculations suggest that even a complete enumeration of the answers is not hopeless.

Appendix A. Source code

The archive with all the code developed in this project is available at `http://www-math.mit.edu/~lesha/papers/hoffman-puzzle.tgz`. It was developed on an x86 GNU/Linux system, but should work with few or no adjustments on most modern systems.

It contains the source files mentioned in the text, implementing the approaches discussed. It also includes a `Makefile`, which is provided to simplify the compilation of this code. If you have the "make" utility, typing `make` in the directory created upon extracting the archive will compile all programs. The resulting executables will be as in Table 1.

The programs in the first part of the table pertain to the original puzzle. To generate the figure with solutions used in this paper, run `n3 | sym | ds > solutions.ps`. (this will not have the light gray highlights used to show the telltale invalid configuration)

Executable program	Source file	Description
`s3`	`solve3D.c`	Basic Hoffman puzzle solver
`sym`	`symm.c`	Filters out symmetric copies, discards partial symmetry sets
`ds`	`drawsoln.c`	Produces a PostScript drawing of the solutions, grouping by duality, highlighting invalid ones.
`gr`	`genrows4D.c`	Generates the 1944 4D rows
`gs`	`gensquares4D.c`	Generates the 4D squares
`cnvsrt`	`cnvsrt.c`	Sorts the stored 4D squares
`cs`	`cubesearch4D.c`	Performs a part of the 4D cube search using the data produced by the previous programs.

Table 1. Executables produced from the various source files by the provided `Makefile`.

The programs in the second part of the table are not really fit for public consumption; one must adjust the paths in all the programs (including `convall`) to fit your system. You will need about 600-700 megabytes of free disk space to run them. The procedure then is as follows. One runs `gr`, then `gs`, then `convall`, which executes `cnvsrt` for all the files storing squares. Only now are you ready to run `cs`. It picks a random square, and counts the number of configurations as in Figure 7(b). It is also rather inefficient. If anyone were to continue this work, we would suggest implementing a lot of the code from scratch, while occasionally referring to the supplied code.

References

[1] J. Rausch, *Hoffman's Packing Puzzle*, Puzzle World, Online. Available May 2003 at `http://www.johnrausch.com/PuzzleWorld/puz/hoffmans_packing_puzzle.htm`

[2] R. K. Guy and J. H. Conway, *Winning Ways for Your Mathematical Plays*, Academic Press, 1982