LEGO:
Automated Model Construction

REBECCA A. H. GOWER, AGNES E. HEYDTMANN,
AND HENRIK G. PETERSEN

1 The Problem Description

The question from LEGO was, "Given any 3D body, how can it be built from LEGO bricks?"

We will assume that we have a "legoised" model, i.e., a 3D matrix containing ones in the places where unit-volume LEGO bricks should be put and zeros where there should be empty spaces. A LEGO unit-volume is a brick 8 mm long and wide and 3.2 mm high which has only one stud on the top for connection with other bricks.

So the task was to make an algorithm which takes as its input a legoised 3D model and produces a way of constructing the model from actual LEGO bricks so that the model "stands" connected.

We were restricted to the use of so-called "family" LEGO bricks and LEGO DUPLO bricks. Both kinds are rectangular prisms whose dimensions in each direction are integer multiples of the unit LEGO brick's dimensions. The lengths, widths and heights of bricks are measured quite naturally in terms of unit bricks. We were given a table of sizes of all the permitted building blocks. Due to the construction of LEGO DUPLO, these bricks can be connected to family bricks with even length and width only. Another restriction is that only family bricks of height 3 or more can be connected to the top of a DUPLO brick.

As there is more than one way of building any model of even quite small size LEGO suggested that we should seek "more stable" models. More stable models are those which are made with bigger bricks and with bricks that have more studs connected to other bricks.

Nothing can be assumed about the shape of a legoised object given as input except that it is connected. However, some characteristics of typical models and how they are commonly built were discussed.

If a model is of sufficiently large size then its interior should have a hollow space in order to save bricks. LEGO designers keep this space approximately box-shaped for simplicity and so that a supporting structure or motorised mechanism can be placed there if necessary. The thickness of the shell of such a hollow structure is between 2 and 6 unit bricks at its minimum. Faced with the task of building such a model, LEGO builders construct it roughly layer by layer. They start with a small area including part of the visible outer edge, then they work on the inner edge, that of the hollow box which is interior to the model, and finally they fill in the area between the outer and inner walls.

We found out that some "cheating" is even allowed when building a LEGO model. That is to say, one may alter the model by very small amounts if the change produces a model which still "looks good" and the change is beneficial in some other way, for example if it increases the stability.

At a later stage LEGO would like to incorporate the use of different coloured bricks, so our automated building technique should take that into consideration.

The only other requirement was that the algorithm would produce a result within a few days and thus does not need to run for enormous amounts of time to get an answer. LEGO is interested in the solution of the problem for relatively large models as are used in their parks or at exhibitions for advertising purposes. These models are typically made from approximately one million bricks. Of course, the legoised versions of these models would have a far greater number of unit LEGO bricks — the number could be expected to be at least an order of magnitude greater. In these circumstances running time is extremely important.

2 The Simplified Approach

The LEGO problem is a huge and complicated problem so in the first instance it was necessary to simplify it.

Although the use of DUPLO bricks is desirable as building with them is both cheaper and quicker, we restricted ourselves to the use of one height of brick only, the common brick of height equal to 3 minimal LEGO units. In this way we could concentrate on building layers in the model, one at a time, without concern about any brick being in more than one layer at a time. However, any algorithm which can work for bricks of height 3 could be applied without any problems to bricks of other heights if all bricks in any one layer were of uniform height. It would require some work to create an algorithm to accept bricks of mixed heights and that was beyond the scope of the study group work.

Another assumption we made was that the shape of the inner hole was given by the legoisation and we were to keep the size and shape of the legoisation regardless of the price. The reason is that to take into account what kind of "cheating" may be allowable would increase the problem size tremendously.

One layer of a model with an inner hole might look as in Figure 1.

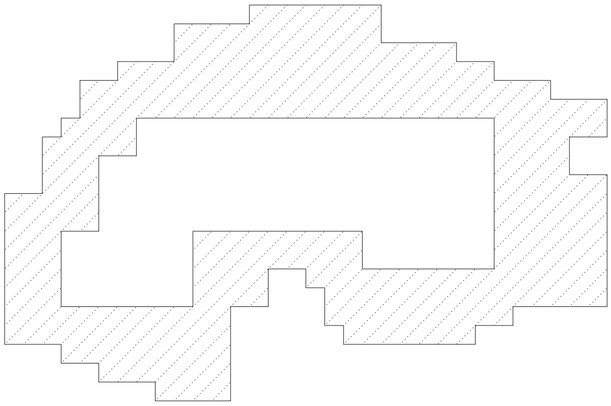

Figure 1: Example of a layer.

3 Formulation as a Combinatorial Optimisation Problem

Despite these considerable simplifications, this problem is still best tackled by a computer and any algorithm which is to be implemented by a computer must have some sort of cost function which the computer can calculate to decide if a brick placement is good or not. The concept of a “good” brick placement is something that the LEGO builders and children who play with LEGO learn from experience. It is possible for computers to mimic the learning of humans through the use of neural networks and this is something that LEGO could investigate [1] although we fear the problem might be too complicated. Instead of attempting to teach the computer a cost function this way, we chose to design one based on heuristics. We shall present this cost function (or penalty function) in Section 4.

Assuming that “the whole is *not* more than the sum of its parts”, a statement that is not quite obvious in this context, we may compute a cost function for the overall structure by adding up cost functions for all the individual bricks. We can now formulate the LEGO problem as a mathematical problem, namely: minimise this overall cost function. This is a so-called combinatorial optimisation problem, that is to minimise a function which is a map from a finite discrete set (the set of all allowable brick configurations) into the real numbers.

There are several advantages to having an overall cost function which is a sum of single-brick cost functions: The single-brick cost functions are easier to understand, and the localised nature of the single-brick cost function makes the optimisation problem very suitable for search methods based on a sequence of small changes.

4 The Penalty Function

When constructing the penalty function we had various issues in mind. The penalty function should be easy to understand conceptually and easy to modify based on experimental results. It should give a high value to structures which are undesirable and a low value to structures which have good strength. The strength of the structure depends on its connectivity so that leads to the question, "What does high connectivity mean in terms of individual bricks?"

We were provided with some examples of ideal walls of thickness 3, 4 and 6. In all cases the side view of the wall is the same and it is shown below in Figure 2.

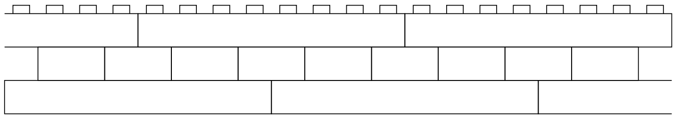

Figure 2: The side view of a wall of good structure.

The top views and end views for the walls of different thicknesses vary but they are similar to each other in many ways too. Compare Figures 3, 4 and 5.

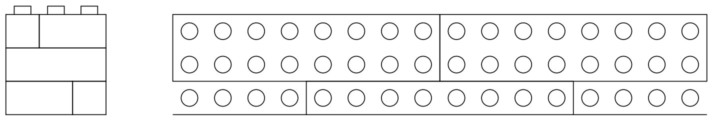

Figure 3: The end view and top view of a wall of width 3.

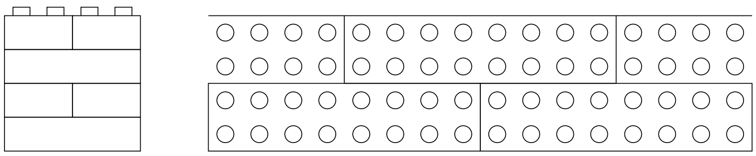

Figure 4: The end view and top view of a wall of width 4.

We were also told that a herring-bone arrangement of bricks is good for filling in thicker areas within a model. When using this structure in consecutive layers, LEGO builders would use the same arrangement of bricks but shifted by the amount of one unit brick in both directions of the plane. This structure is illustrated in Figure 6.

After examining these examples the final decision reached was that there was no single measure of connectivity and structural strength. Therefore we have a

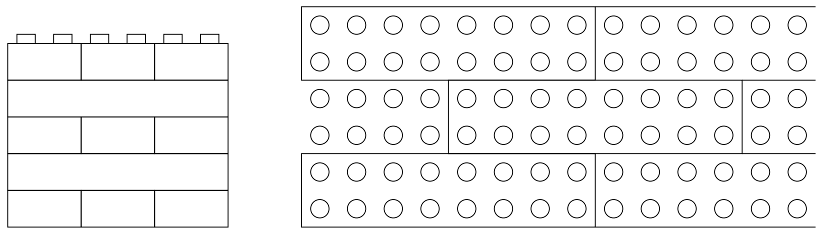

Figure 5: The end view and top view of a wall of width 6.

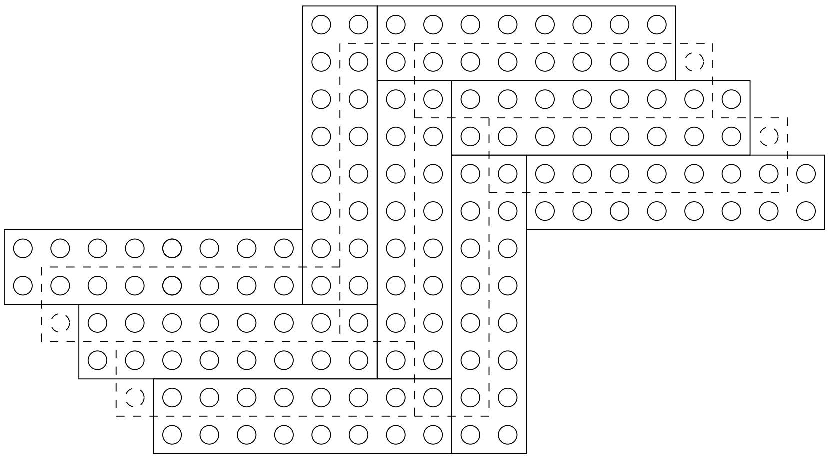

Figure 6: The top view of two layers of a herring-bone pattern. The lower layer is indicated by dashed lines.

penalty function with many terms to represent the many features which contribute to the strength of a LEGO construction.

The important factors relating to the connectivity of bricks are:

1. a high percentage of each brick's area covered, above and below, by other bricks;
2. the use of big bricks;
3. alternating directionality of bricks in consecutive layers;
4. a high percentage of each brick's vertical boundary covered by bricks in the layers above and below;
5. the joining boundary of two bricks in one layer which makes the vertical line in a "T" pattern of boundaries is adjacent to the middle of the neighbouring brick in the same layer which resides above the horizontal line in the "T" — see Figure 7 a);
6. the joining boundary of two bricks in a layer is covered above and below by a brick centred with respect to this boundary — compare with Figure 7 b).

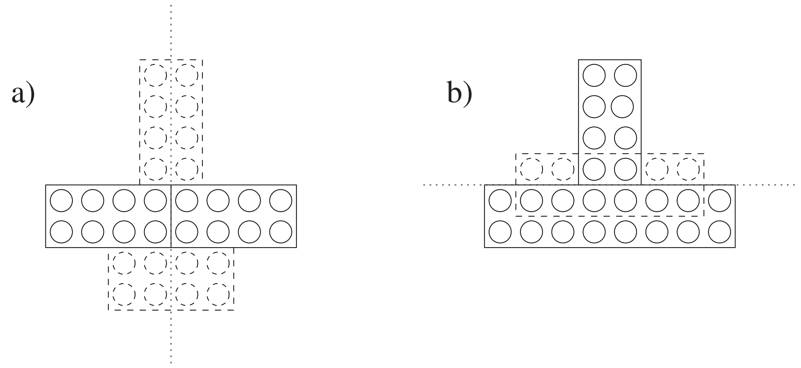

Figure 7: Horizontally a) and vertically b) centred alignment — dashed LEGO bricks indicate possible "good" placements with respect to the boundary indicated by the dotted line.

Although the first factor is important there is not so much one can do about it as that is, to a large extent, determined by the shape of the model which is fixed — at least, we assume it to be so.

In designing our penalty function we assumed that the creation of a model from the legoised information would be done one layer at a time and that once a layer was completed it would be left fixed. Although this may seem restrictive it is the practice of the professional LEGO builders and because there are so many possible ways to turn unit bricks into larger bricks we hope that this will still produce a good solution. In any case, it will help produce a solution in reasonable time.

We recommend the use of a single-brick penalty function such as the following:

$$P = C_1 P_1 + C_2 P_2 + C_3 P_3 + \cdots$$

where the C_i s are weighting constants and the P_i s are as described below.

P_1 relates to perpendicularity: This function is to penalise bricks which do not fit with factor number 3 above. The purpose of this term is to encourage a solution in which bricks on consecutive layers should be perpendicular to each other if they overlap each other when seen from above.

The Penalty P_1: Suppose a brick in layer n is placed so that it is connected to a number of bricks in layer $n-1$. For each brick in layer $n-1$ connecting to this brick in layer n, but not perpendicular to it there is a penalty of 1. (Square bricks are defined to be perpendicular to no other brick.)

The decision regarding square bricks is both natural and necessary. If they were defined to be perpendicular to everything then one way to get zero penalty from this term would be to build the entire model (so far as is possible) from square bricks but this is obviously undesirable.

As this stands it gives quite a harsh penalty to the herring-bone structure when it is used in consecutive layers as shown in Figure 6. There would be a perpendicularity penalty of 2 per brick. An alternative would be to make the penalty equal to the area of overlap with parallel bricks divided by the total area of the brick. This would reduce the penalty from this term a little. However, it could be reduced to as little as 1/8 per brick if the layers were arranged on top of each other differently, namely as in Figure 8.

P_2 relates to vertical boundaries: This function is to penalise the placement of a brick which does not fit with factor number 4 above. Boundaries between bricks on one layer should not be visible from above through the next layer.

The Penalty P_2: There is a penalty of 1 for each unit length of boundary not hidden by a brick from the layer above it. Outer boundaries, that is those on the edge of the model, are not counted.

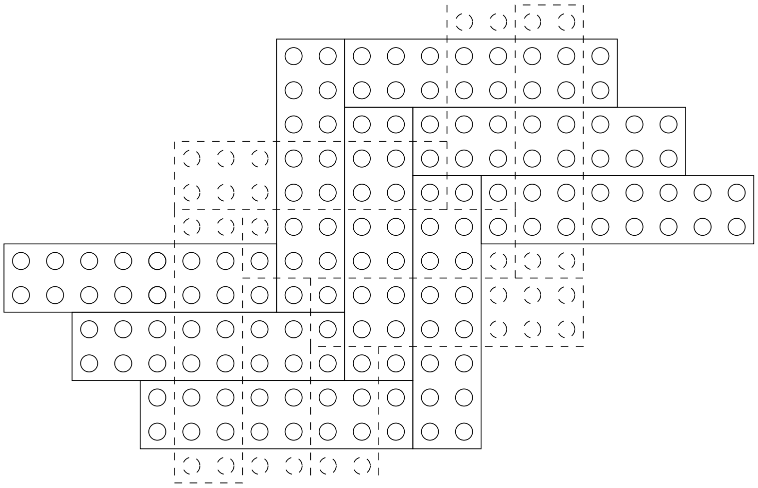

Figure 8: The top view of an alternative construction of two layers of a herring-bone pattern. The lower layer is indicated by dashed lines.

The condition about outer boundaries is necessary if the penalty function is to be used in a local search algorithm of some kind. It would make no difference in comparisons between different ways of building entire layers, or indeed, entire models. However, if this condition is not put in then a local search will try to avoid placing bricks on the edges of the model as it will give a penalty. Obviously this is undesirable as we would ideally like to arrange the bricks of a model from the outside edges inwards as LEGO builders currently do.

P_3 relates to horizontal alignment: This is to penalise bricks placed in a way that goes against factor number 5 listed above. If a brick is placed so that either its short or its long side comes up against a "T" shaped boundary formed by two other bricks in the same layer, then it should be penalised if it is not placed so that the middle of the edge is against the boundary.

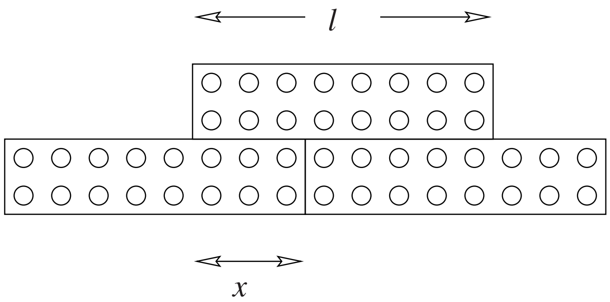

Figure 9: The top view of a situation in which the term P_3 is calculated.

The Penalty P_3: Suppose the length of the edge of the brick which is against the boundary is l and that the boundary is x units away from the end of the brick then the penalty is

$$\frac{|x - l/2|}{l/2}.$$

The division by $l/2$ is to make sure that longer bricks are not penalised by their capacity to have their half-way mark further from the boundary between the other two bricks.

The penalty function with these three terms gives zero values for the ideal walls given to us. So, although this penalty function may treat other wall constructions as being equally good as these, it will at least consider these to be optimal.

For the herring-bone structure (of either Figure 6 or 8) the value from this penalty function is higher than for a structure in which each layer looks like the top view in Figure 5 and in which the directionality of the bricks alternates from layer to layer. If care is taken to avoid a vertical boundary penalty we get a penalty of zero for this brick-wall like structure.

Note that term P_2 tends to favour the use of larger bricks as they have less boundary per unit of area. The smaller the sum of the boundary lengths in any layer the less likely it is to incur a penalty of this type. If LEGO need the penalty function to favour the use of large bricks explicitly a fourth penalty term could be added. Say, P_4 with value $1/A$ where A is the area of the brick.

One way of making sure the use of large bricks in not discouraged, which could be used in a local search, would be to take account of the area covered by each optional arrangement of bricks in any step of the local search. Then the total penalty for the bricks in each option can be divided by their area.

We have thus constructed a penalty function which we believe is a good starting point. It must be expected that (based on experiments) the user will find it necessary to alter the penalty function by modifying the weighting constants or by adding new terms to take into account other considerations such as factors number 2 and number 6. Possible modifications to take account of factor number 2 have been discussed above. Factor number 6 could be incorporated in a term in the style of P_3.

The advantage of our penalty function is that it is relatively easy to determine the necessity of such alterations and they would be relatively easy to make.

5 Using the Penalty Function — Implementation Strategies

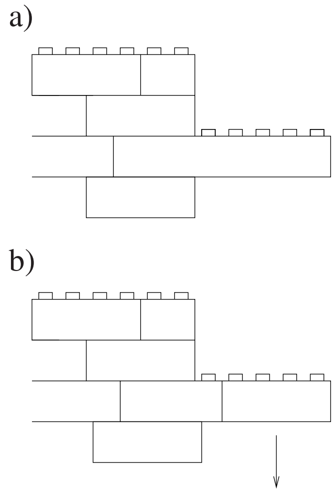

Figure 10: How a protruding part in a model should be protected by a) instead of b).

There are many ways one could make use of the above penalty function. However, in all cases there is a need to be careful at the edges of the model. For example, it is possible that in some part of a model one layer sticks out a little beyond either the layer above or below. In this case it is important that the construction of the protruding part is designed in such a way that it is part of a brick or set of bricks which are firmly attached to the model. (See Figure 10 a).) We want to exclude the possibility that the protruding part is to be made of a brick or bricks with no connection to the model as illustrated in Figure 10 b). Although such a design would incur a penalty with our function it is not forbidden. Therefore it is necessary to fix these "dangerous" edges first.

For a little extra strength and as a precaution against constructions in which bricks are only connected from either the top or the bottom we define the dangerous edges more generally as follows. The dangerous unit-volumes in layer n are those which are not overlapped (when viewed from above) by the intersection of layers $n-1$ and $n+1$. Or mathematically: Project the areas covered by layers $n-1$, n and $n+1$ into one plane and call them A, B and C respectively. Take $B \setminus (A \cap C)$ as the "dangerous", possibly disconnected areas — see Figure 11. These unit volumes should be recognised and allocated to bricks which will hold firmly in place before any other work is done on that layer and should then be considered fixed while the rest of the layer is completed.

During processing of the dangerous edges, one could use the penalty function as it stands with a high weight on vertical boundaries. This should give satisfactory connections with the layer below. Furthermore, to satisfy the wish to connect properly with the layer above also one may include a vertical boundary term relating to the above (as yet nonexistent) layer given as one big imaginary brick having the size and shape of that entire layer.

Overall, we recommend the following procedure.

1. Fix the dangerous edges by a local search or simulated annealing.

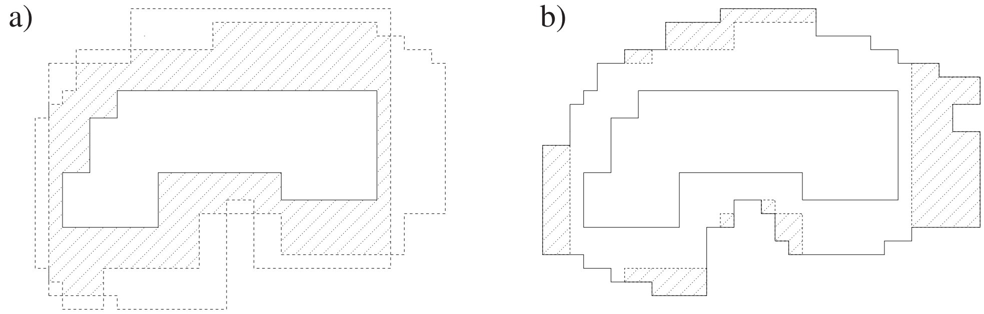

Figure 11: In a) we illustrate the outer boundaries of layer $n-1$ and layer $n+1$ with dashed lines and the shaded area is the intersection of these two layers. In b) the solid line illustrates the outer boundary of layer n and the shaded areas are the "dangerous" edges in the n-th layer. For simplicity of the inside hole, the inner boundaries of all three layers coincide.

2. Divide the remainder of the layer into regions and apply the appropriate mechanism to each region;

   (a) use of a library;

   (b) pattern filling and translating;

   (c) use of local search or simulated annealing.

By the use of a library we mean that there could be a library of methods to build certain types of features in what is known to be a good way. For example, the library could contain information about building walls of various thicknesses. The automated building programme would either have to detect those features or, in an interactive setting, a LEGO employee could mark the area of a model that is to be built by a certain method.

Pattern filling would be suitable for use in layers where there were large areas to be filled which were not suited to one of the library constructions. The filling of such an area could be done by use of something like a herring-bone structure or a brick-wall like structure or some other kind of tiling. If subsequent layers shared similar large regions to be filled with bulk pattern then each layer should be designed so that it fits in a way which is known to be good with the bulk part of the layer below. Usually this will involve use of the same pattern but translated in some way relative to the pattern in the lower layer. This is what is meant by "translating" in 2(b).

The idea behind a local search is to place either one brick or a few bricks at a time by searching through some of the possibilities for a small region. One way in which this could be done is to decide on the next small area which should be covered and then look at different options for doing this. In examining and evaluating the different options one can look ahead a few steps at a time to avoid getting into an undesirable situation later on. Having chosen the best option, according to which gave the lowest penalty, either just one brick or a few bricks could be placed.

The simulated annealing method is probably the most promising method for dealing with the regions described in parts 1 and 2(c) of the procedure. The idea is to divide these regions into medium sized subregions of size $n = 200 - 500$ studs. Each subregion is then treated separately using simulated annealing. The idea for using simulated annealing here is to take some arbitrary initial guess for brick placements on these substructures and then iterate on that by removing some small number of bricks and replacing them by a few new bricks. It is important for the success of using simulated annealing that the *change* in the cost function for each iteration can be computed rapidly.

The simulated annealing method will then lead to near optimal results for each of these subregions. The quality of the result depends on various method parameters used in simulated annealing. A description of the method and how to choose these parameters is beyond the scope of this report, excellent references treating all these problems are [2] and [3].

Apart from the values used for the method parameters, the quality of the result will depend on the size of the subregions. For very small subregions there are few degrees of freedom and a result similar to local search methods must be expected. Increasing the size of the subregions leads to better results. However, the number of necessary iterations goes quadratically with "the problem size", which is linearly related to n. Thus, for an overall structure of M studs treated entirely with simulated annealing, we get an overall computer time for all the M/n subregions of

$$T = \alpha \times (\text{number of iterations}) = \alpha\beta \left(\frac{M}{n}\right) n^2 = \alpha M n.$$

The constant α is the overhead for computing the change in the object function for one iteration. This constant depends strongly on how efficiently the software implementation is. Advanced geometrically based data structures should be used to keep track of which bricks in the layer below may overlap with any given brick, and which bricks share a boundary with a given brick. If the number of necessary (floating point or boolean) operations for an update is 1000 (a rather conservative guess), the value for α will be around $\alpha \simeq 3 \times 10^{-7}$ minutes on a Pentium Pro. The constant β gives the linear relation between the number of iterations and the problem size. This constant depends on the method parameters (that is the required quality). It is difficult to give a guess at the value of β, but for other applications such as Travelling Salesman Problems, the constant β has typical values between 10 and 100. We then get an overall computer time of $T \simeq 7000$ minutes or 5 days when $M = 10^7$, $n = 200$ and $\beta = 10$. The bottom line of this overhead discussion is that the quality of the result depends on the patience of the user and the computer power. There may be nothing to do about the users patience, but with the current rate of increase in computer power there should be a hope for high quality results within a reasonably short time. Furthermore, subregions on the same layer which do not share a boundary may be treated simultaneously. The method is thus well suited for a multiprocessor computer with a possible high speedup gained.

6 Conclusions

Although the problem is far from completely solved we have produced some ideas which can help LEGO begin the next stage of their automatic construction of models. Namely, we have produced a penalty function which can be adapted to their needs and we have discussed implementation strategies for an algorithm to use this function.

Obviously the next stage would be to test the ideas presented here by writing a computer program to implement an algorithm along these lines. Then some experimentation could be performed to determine suitable weighting constants in the penalty function.

It may be that further refinements of the penalty function or implementation strategy are required. For example, it may be necessary to add further terms to the penalty function. Other open questions are, "How should the first layer be built?" and, "Which layer should be chosen as the first?" A natural approach might be to start with the layer which has the biggest solid area.

Finally, when testing and refining has produced a satisfactory result it will be time to generalise this work and produce an algorithm capable of using bricks of different heights. The first step could be to use bricks of different heights in separate layers and when that is successful to move on to bricks of mixed heights crossing layer boundaries.

Another feature which could be incorporated into a more advanced algorithm would be to look beyond the connections with the layers immediately above and below a particular layer. At the moment there is nothing in the algorithm to favour the design illustrated in Figure 2 over an arrangement in which the vertical boundaries in every second layer are aligned. Yet the arrangement illustrated in this paper is preferable.

Acknowledgements

Olga Timcenko for many helpful comments and patiently answering so many questions during the week.

Bibliography

[1] Bishop, C. M., *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.

[2] Aarts, E. H. L. & van Laarhoven P. J. M., *Simulated Annealing: Theory and Applications.* D. Reidel Publishing Company, Dordrecht-Boston, Mass., 1987.

[3] Aarts, E. H. L., *Simulated Annealing and Boltzmann machines.* John Wiley and sons, Chichester, 1989.

The Group Working on the Problem

Anton Antonov, [anton@unidhp1.uni-c.dk]

Rebecca A. H. Gower, [Rebecca.Gower@dfee.gov.uk]

Agnes E. Heydtmann, [Agnes.Heydtmann@mat.dtu.dk]

Thomas Jakobsen, [T.Jakobsen@mat.dtu.dk]

Jørn Møller Jensen, [J.M.Jensen@mat.dtu.dk]

Ahn Louise Larsen, [A.L.Larsen@mat.dtu.dk]

Henrik G. Petersen, [hgp@mip.ou.dk]

Carsten Thomassen, [C.Thomassen@mat.dtu.dk]

Olga Timcenko, [olga@digi.lego.com]