Fast and Scalable Domino Portrait Generation

Hadrien Cambazard, John Horan, Eoin O'Mahony, and Barry O'Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{h.cambazard|j.horan|e.omahony|b.osullivan}@4c.ucc.ie

Abstract. A domino portrait is an approximation of an image using a given number of sets of dominoes. This problem was first stated in 1981. Domino portraits have been generated most often using integer linear programming techniques that provide optimal solutions, but these can be slow and do not scale well to larger portraits. In this paper we propose a new approach that overcomes these limitations and provides high quality portraits. Our approach combines techniques from operations research, artificial intelligence, and computer vision. Starting from a randomly generated template of blank domino shapes, a subsequent optimal placement of dominoes can be achieved in constant time when the problem is viewed as a minimum cost flow. The domino portraits one obtains are good, but not as visually attractive as optimal ones. Combining techniques from computer vision and large neighborhood search we can quickly improve our portraits to be visually indistinguishable from those found optimally. Empirically, we show that we obtain many orders of magnitude reduction in search time.

1 Introduction

In 1981 Kenneth Knowlton filed for a United States Patent entitled "Representation of Designs" [4] in which he proposed the use of dominoes to render monochrome images. Twenty five years later, at the 2006 Conference on Constraint Programming, Artificial Intelligence and Operations Research (CP-AI-OR 2006), Robert Bosch gave an invited talk on "OptArt", focusing on how optimisation could be used to create pictures, portraits, and other works of art. In that talk, Bosch not only demonstrated the beauty of computer-generated art, but also the technical challenges involved in producing it. A domino portrait is simply a rendering of an image using a given number of sets of dominoes. Generally he uses "double nine" domino sets, which contain all dominoes from the "double blank" to the "double nine", giving fifty five dominoes in all.

The nice property of "double nine" domino sets is that they give a wide range of shades from complete black (the blank domino) to a bright white (the double nine domino). A set of dominoes gives us a constrained palette of monochrome shades, which we can use to produce images. We say that the palette is constrained for two reasons. Firstly, each set of dominoes contains only one domino of each type. Secondly, we are not allowed to break dominoes into two parts, but rather use the entire domino.

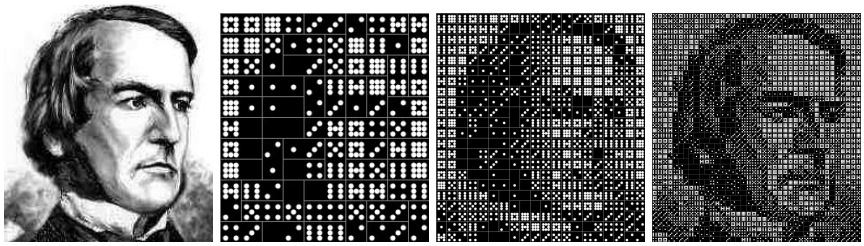


Fig. 1. A well known portrait of George Boole is presented on the left, with a sequence of domino portraits generated from this image using 1, 4 and 16 sets of dominoes as we move to the right, respectively.

Several examples of domino portraits based on a well known portrait of George Boole are presented in Figure 1. It is clear that as we increase the number of dominoes we have at our disposal, the domino portrait we obtain is a better approximation of the target input image. In Figure 2 a much larger domino portrait of Boole is presented, which is sufficiently large for the reader to see each of the individual dominoes that comprise the portrait.

A problem with current approaches to generating domino portraits is that they do not scale very well. This is mostly due to the fact that Bosch has been interested in finding optimal domino portraits; we will explain how the notion of optimality is defined later in this paper. We set out to develop a scalable approach to generating domino portraits that would not be concerned with whether the portraits found were optimal or not, but be concerned with whether the portraits were sufficiently good so as to be visually indistinguishable from the optimal ones.

In this paper we present a new approach to building approximations of a target image using a specified number of complete sets of "double nine" dominoes [2, 3]. We adopt an approach similar to Knowlton's [4] (and to Knuth's [5]), in which the image is divided up into blank domino outlines to which we assign dominos. Rather than treating this problem as a traditional assignment problem, which can be solved using the Hungarian Method, and other similar algorithms, we formulate it as a *minimum cost flow*. The advantage is that the assignment step becomes constant time, allowing us to scale to arbitrary sized portraits. However, because we predetermine the orientations of the dominoes, we are unlikely to find an optimal domino configuration. Therefore, we adopt a heuristic approach to identifying regions of the domino placement that, if redesigned, would improve the quality of the resultant portrait. This last step is performed using a *large neighborhood search*. An empirical evaluation demonstrates the utility of our approach.

The remainder of the paper is organised as follows. Section 2 presents the domino portrait problem and explains in detail how it is defined. We then briefly summarise an existing linear model for finding optimal domino portraits in Section 3, as well as other heuristic approaches that have been studied. Section 4

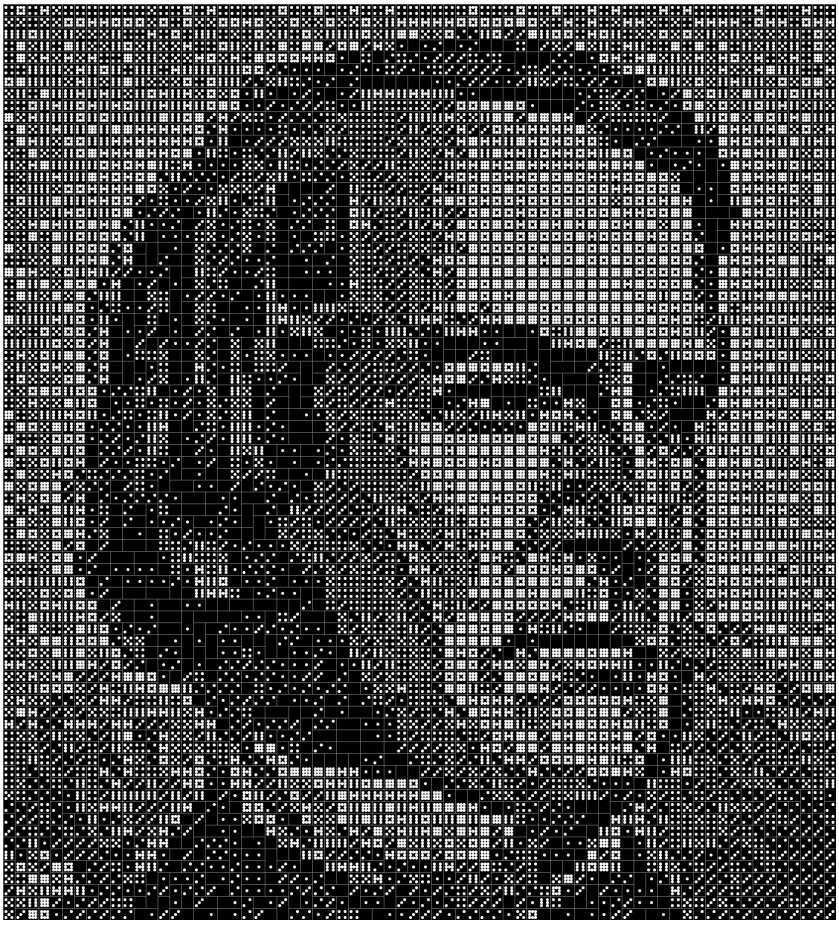


Fig. 2. A domino portrait of George Boole generated by our approach using 49 sets of "double nine" dominoes, i.e. $49 \times 55 = 2695$ individual dominoes.

describes the two-step approach we employ here, and our innovation based on a minimum cost flow formulation. In Section 5 we outline a practical improvement to our basic approach that involves locally perturbing the portrait. Section 6 presents and discusses the results. A number of concluding remarks are made in Section 7.

2 The Domino Portrait Generation Problem

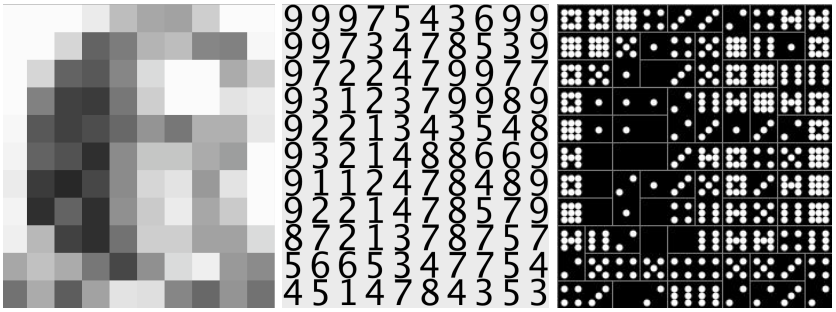
A domino portrait can be generated for any target image. The first step in the process is to convert the target image into a grayscale graphic image using, for example, the UNIX *pgm* command. Each pixel in a grayscale image is given a grayscale value between 0 (black) and 255 (white).

We consider rendering images using sets of "double nine" dominoes. There are 55 dominoes in a complete set of double nine dominoes: 10 dominoes with equal face values in both halves, i.e. all dominoes with face valuations equal to $(0,0),\ldots,(9,9)$ along with an additional 45 non-equal face dominoes with face values in $\{(v_1,v_2)|v_1 \in \{0,\ldots,8\}, v_2 \in \{v_1+1,\ldots,9\}\}$. The area covered by a single set of dominoes is 110 square units, since we have 55 dominoes each with 2 units. Therefore, given s sets of dominoes, the grayscale image is divided into $11s \times 10s$ cells and for each cell in row r_i and column c_i the mean grayscale value is computed and scaled to an integer between 0 and 9 called g_{ij}. The values in each cell defines the perfect half domino value to place in that cell.

Each domino with equal valued halves has two possible orientations, vertical and horizontal, whereas each non-equal valued dominoes have 4 orientations since such a domino can be flipped along its vertical and horizontal axes. For $k = s^2$ sets of dominoes we can use a canvas of size $11s \times 10s$ to be filled with the $55 \times k$ dominos, but in practice we can represent any canvas of size $110 \times k$. The following notation will be used throughout the paper:

- k is the number of sets of dominoes, and $N = 55 \times k$ is the number of individual dominoes.
- $d_i = (p_i^1, p_i^2)$ for domino number i, with $p_i^q \in \{0,\ldots,9\}$
- g_{ij} is the grey value of cell (r_i, c_j) between 0 and 9. The whole matrix of grey values is refered to as the grey matrix in the following.

The cost of positioning a half-domino p_l^q on a cell (r_i, c_j) is equal to $(p_l^q - g_{ij})^2$. Notice that it is quadratic so that the cost grows faster than the error and large errors are strongly penalised. The problem is to place the dominoes on the canvas so that the overall cost (the sum of the costs of each cell of the canvas) is minimised and every domino is used exactly once. A graphical representation of the process is presented in Figure 3.



(a) The grayscale values are scaled to $0\ldots9$.

(b) The result of the scaling process.

(c) An example placement of dominos.

Fig. 3. A summary of the process of generating a domino portrait from an image.

3 An Integer Linear Programming Model

Robert Bosch proposed an integer linear programming formulation of the domino portrait generation problem in [3]. His model is based on boolean variables that specify if a given domino is placed with a given orientation with respect to its reference square (the top left corner of each horizontally placed domino in Bosch's model) in a given cell of the canvas. Constraints then stipulate that each domino has to be used exactly once, and that each cell has to be covered by a domino. The resulting integer programs are quite large, with more than one million decision variables and five thousand constraints for $k = 49$, but Bosch reports that they are relatively easy to solve, requiring almost two hours when $k = 49$.

We used this model in our experiments as a baseline, with a very simple improvement not described by Bosch in his papers, but used by Knowlton, which involves keeping only the optimal orientation for each domino. A domino can be placed in two orientations at a given position but one often dominates the other, in terms of cost, and it is only necessary to consider the best orientation; this can be seen as a form of symmetry breaking over individual dominoes. The scalability of this model is, however, very limited and we will present a non-optimal, but much more efficient, approach to generating domino portraits in the next section, and then follow this presentation with an improvement based on large neighbourhood search.

4 A Two-Step Approximation

In his original patent, Knowlton outlined a two-stage process for generating domino portraits. The first step in his approach involved generating an initial arrangement of empty domino holders (rectangles) on the canvas, i.e. pairs of adjacent cells, which were later "filled" using dominoes. In this step he maximised the average *unbalance* of each domino holder by maximising the average difference between the two brightness values it contained. The second step involved assigning dominoes to the holders computed from the first step in order to minimise the error between the brightness provided by a domino and the brightness required in the domino holder computed from the first step. Donald Knuth subsequently recast Knowlton's method as an assignment problem [5], but because the two steps are independent, there is no guarantee the the resulting domino portrait will be close to optimality.

Here, we use another modification of Knowlton's method in which the initial pattern of empty dominoes is generated randomly, the dominoes are then placed into this pattern using an assignment problem formulation. This approach relies on the observation that the problem becomes polynomial if the *pattern* of the dominos is known, since the assignment step is itself polynomial. This suggests that restricting ourselves to searching over alternative patterns is enough to generate optimal domino portraits. In practice we will show that any random pattern provides a very good upper bound on the cost of the domino portrait. We will present the details of each step in detail.

4.1 Generating the Pattern of Empty Domino Holders

We generate a random packing of empty domino holders on the canvas using Algorithm 1. We refer to this arrangement of empty domino holders as a *pattern*. Generating the pattern can be regarded as a packing problem. An example pattern is presented in Figure 4.

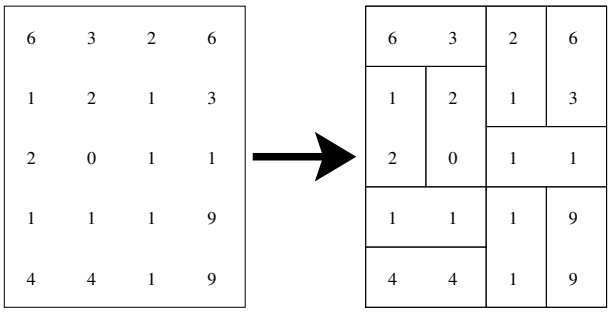


Fig. 4. An example of a pattern on the right that covers the grey matrix on the left.

Algorithm 1 proceeds by filling the grid from the bottom to the top, line by line from the left to the right (lines 2 and 3). At each step it randomly assigns a rectangle vertically or horizontally (line 4) before going into a propagation step. Once a cell is surrounded (orthogonally) by three cells already covered by a domino holder, the orientation of the rectangle covering this cell is known and can be propagated (lines 5–6). This is performed until a fixed-point is reached, or a contradiction is met. A contradiction is raised when an odd number of connected cells remains in the grid, since dominoes cover pairs of cells. Each time a contradiction is met a restart step is performed. A small sub-region of the pattern is wiped out by removing a given number of lines.

Algorithm 1 Random pattern generator

1: **while** there exists an empty cell in the grid **do**
2: $i \leftarrow$ the first row containing an empty cell
3: $j \leftarrow$ the first column such that (i, j) is empty
4: Place a rectangle randomly at position $(i, j), (i + 1, j)$ or $(i, j), (i, j + 1)$
5: **while** there exists (i, j), an empty cell with three occupied orthogonal neighbours **and** all regions of empty connected cells are of even size **do**
6: Place a rectangle to cover (i, j) and the empty cell next to (i, j)
7: **end while**
8: **if** there is a region of an odd number of connected empty cells in the grid **then**
9: Wipe out part of the grid
10: **end if**
11: **end while**

This non-deterministic approach to random pattern generation performs very well in practice. In particular, we found this approach much faster than a complete backtracking algorithm for large number of dominoes.

4.2 Solving the Assignment Problem as a Min-Cost Flow

Once the pattern is known, placing the dominoes optimally is a polynomial problem – it is an optimal assignment problem. Figure 5 presents an example of the assignment problem. Notice that the cost $c(d_i, \langle a, b \rangle)$ of assigning a domino $d_i = (p_i^1, p_i^2)$ in a given rectangle of grey values $\langle a, b \rangle$ is defined as the best cost among the two possible orientations of the domino:

$$c(d_i, \langle a, b \rangle) = min((p_i^1 - a)^2 + (p_i^2 - b)^2, (p_i^1 - b)^2 + (p_i^2 - a)^2). \quad (1)$$

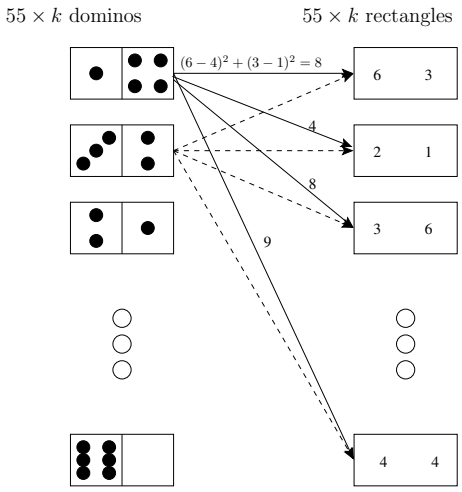


Fig. 5. An example of the assignment problem to be solved once the pattern is known.

Solving the assignment problem can be done very efficiently using the Hungarian method in $O(n^3)$. However, in our setting n denotes the number of individual dominoes, which can quickly become very large. A good portrait often requires at least 100 sets of dominoes, giving 5500 individual dominoes. Clearly, the Hungarian method would not scale to those sizes.

We propose a novel formulation of this step as a min-cost flow. Observe that in the bipartite graph in Figure 5, dominoes on the left side are repeated k times and many rectangles on the right side have identical costs. In fact as the number of points varies from 0 to 9 on each square, there is only 55 possible pairs of points (for two adjacent squares) in the portrait. We can take advantage of these symmetries using the following formulation. We define the following notation:

– An *area* is a set of all rectangles with identical pairs of costs in the pattern. Area j corresponds to a rectangle of cost $\langle j_1, j_2 \rangle$ and the number of such rectangles is denoted $capa_j$. Moreover, the total number of areas is denoted by $nbArea$ and $nbArea \leq 55$.
– x_{ij} is the number of dominoes of *kind* i assigned to area j.
– $c(d_i, j)$ is the cost of assigning a domino of kind d_i into area j. $c(d_i, j)$ is the same cost as previously so that $c(d_i, j) = c(d_i, \langle j_1, j_2 \rangle)$ as defined by Equation 1.

In the pattern given in Figure 4, we would have $nbArea = 6$ where each area would be defined by one of the six rectangles $\{\langle 6, 3\rangle, \langle 2, 1\rangle, \langle 2, 0\rangle, \langle 4, 4\rangle, \langle 1, 1\rangle, \langle 9, 9\rangle\}$. The optimal assignment can be reformulated as follows:

$$\begin{array}{l} Minimize \sum_{i,j} c_{ij} x_{ij} \\ \quad \text{subject to} \\ \quad\quad \sum_j x_{ij} = k, \forall i \leq 55 \\ \quad\quad \sum_i x_{ij} \leq capa_j, \forall j \leq nbArea \end{array} \tag{2}$$

The first constraint of this linear program ensures that exactly k dominoes of each kind are assigned. The second constraint ensures that no more than $capa_j$ dominoes are placed in the same area. In practice, there are exactly $capa_j$ dominoes to fill the area as we have $\sum_j capa_j = 55 \times k$. This problem can be better understood, and more efficiently solved, as a min-cost flow problem on the graph presented in Figure 6, where the x variables can be interpreted as the amount of flow from a domino i to an area j.

There are two key observations to be made about this formulation. Firstly, we only need to know the area where a domino is assigned and not specifically where it is placed in this area. Secondly, we only need to know how many dominoes of each kind are assigned in each area and not where each specific domino is assigned. The min-cost flow formulation takes these symmetries into account and provides a much more efficient way of solving the previous assignment problem. Notice that the size of the graph (number of nodes and edges) supporting the flow is independent of k; only the flow and capacities are increasing, making the approach robust to increases in k.

Once reduced to a min-cost flow formulation the problem can be solved in a variety of ways. It is easy, for example, to formulate it as a linear program (see Model 2). Fortunately this linear program has the quality of integrality, thus only the linear relaxation needs to be solved. Alternatively, there exist many algorithms to solve min-cost flow, e.g. the Successive Shortest Path (SSP) [1] algorithm which sends the largest possible flow along the shortest path from source to sink, found by Dijkstra's algorithm, at each iteration. The complexity of this algorithm with a small optimisation is $O(n \times max_{j \in \{1 \ldots nbArea\}}(capa_j))$ where n is the number of nodes.

An alternate algorithm, the Enhanced Capacity Scaling algorithm [1], is a strongly polynomial improvement on the Successive Shortest Path algorithm. It has a complexity of $O((m \log n)(m + n \log n))$, where n is the number of nodes and m is the number of arcs. This means that for our min-cost flow formulation

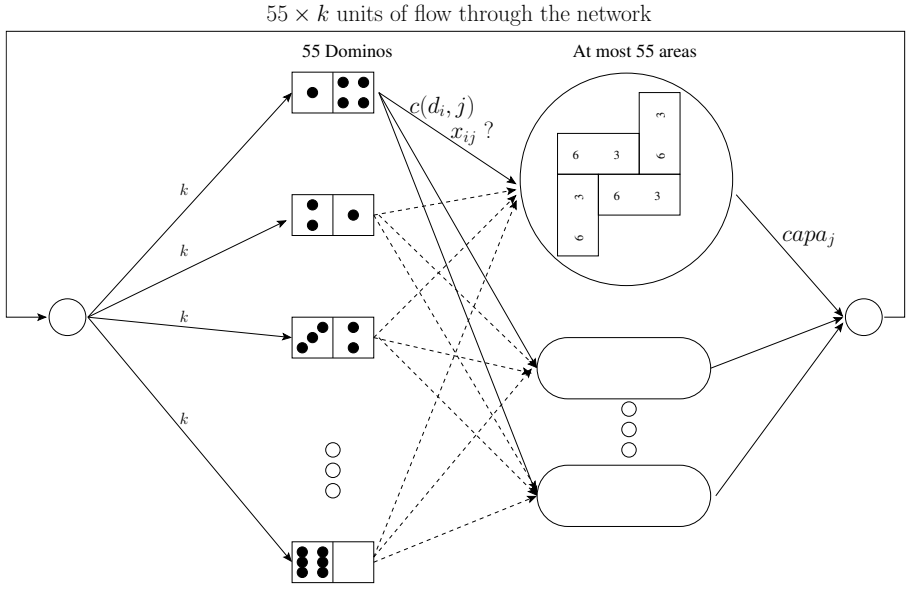


Fig. 6. The assignment problem translated as a min-cost flow problem.

the algorithm runs in *constant time* as the number of nodes and the number of arcs are constant and not dependent on the number of sets of dominoes used to generate the portrait.

5 Improving the Pattern using Local Search

Using the min-cost flow formulation we can solve the assignment step of the domino portrait generation problem in constant time. The only obstacle to generating optimal domino portraits is the choice of pattern to provide to the flow step. Notice that the pattern only matters where the grey values are unbalanced; the pattern in uniform areas has almost no effect on the final cost. In terms of the flow formulation, it means that a change of the pattern that would not affect the size of the areas of the flow graph, the $capa_j$ values, has no effect on the optimal assignment. Therefore, we consider perturbing the pattern slightly in a local search approach to affect the $capa_j$ values in order to improve the flow.

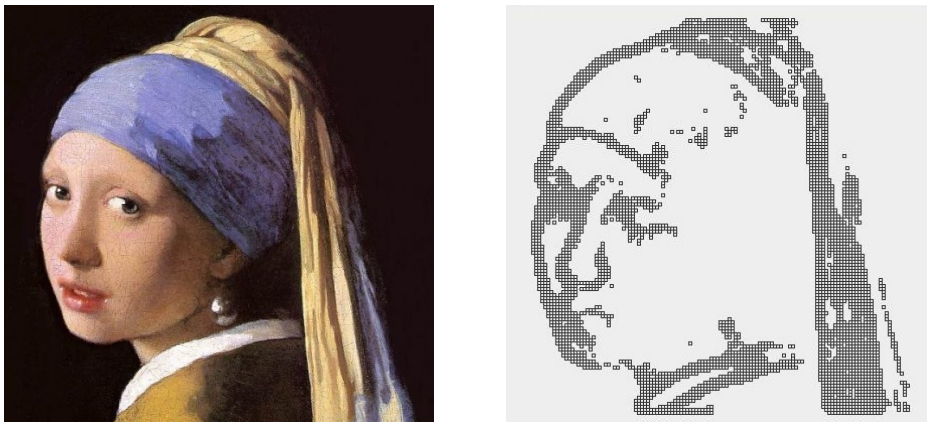
The algorithm we implemented can be described as a Large Neighborhood Search [8] over patterns. It proceeds as follows:

1. Identify the regions of the canvas where the grey values are unbalanced and thus, where the pattern might benefit from improvement. We denote as X the set of points (i, j) corresponding to those regions.
2. Select a point $x \in X$ and remove it from X. If X is empty then select a point randomly.

3. Remove M dominoes around x; x can be seen as the centre of the new empty region.
4. Enumerate all possible patterns that can fill the empty region. For each of those patterns incrementally update the $capa_j$ values and compute the corresponding new min-cost flow denoting the cost of the overall resulting pattern. Note that this is a global optimization step as the dominoes that were previously assigned in the region might now be in a completely different place.
5. Return to Step 2 (above) as long as the average improvement over the last 20 iterations remains above a threshold (set very low in practice).

The points in the set X are weighted in such a way that any points of interest adjacent to one already chosen for improvement are less likely to be selected than those that are independent of chosen points. This is to maximize the impact of the improvements during the initial executions and to ensure an overall faster convergence.

The first point is performed using an algorithm from computer vision that performs *corner detection,* or *interest point detection*, to extract certain kinds of features to infer the contents of an image. We used the FAST (Features from Accelerated Segment Test) algorithm from [6, 7]. This approach seems very well suited for portraits as it highlights the important characteristics of the face (eyes, mouth, hair etc...) which matter in the final domino portrait. Figure 7(b) shows the result of FAST on the "Girl with a Pearl Earring".



(a) Vermeer's "A Girl with a Pearl Earring".

(b) The X region detected by the FAST algorithm for $k = 225$.

Fig. 7. Selecting the interesting region to focus on in the local search step.

The neighborhood explored is defined by all the possible patterns for a small region of $2 \times M$ squares of the grid ($M = 15$ is the setting used in our ex-

periments). The enumeration is performed using the propagation described in Algorithm 1 in a complete backtracking search. Finally, the problem of finding the optimal flow regarding small changes of $capa_j$ is a sensitivity analysis problem on the min-cost flow and can be performed incrementally [1]. The optimal flow is maintained while performing a local search on the $capa_j$ values reflecting the changes in the pattern. This is possible due to the efficiency of the flow model and its incremental behaviour.

6 Experiments

Robert Bosch proposed an integer linear programming (ILP) approach to solving this problem, which we discussed earlier in the paper, and we used his approach as a baseline in our experiments. We used Vermeer's "A Girl with a Pearl Earring" (Figure 7(a)). All times quoted are the times it takes to generate and solve the respective models; they do not include the time taken to convert the solution into a viewable image. Experiments were run on a 2.8GHz Intel Xeon processor running Linux Fedora Core 2 with 4Gb of RAM.

Firstly, we sought to compare the performance of the min-cost flow and Hungarian method to demonstrate the scalability of the flow algorithm (Table 1). The flow algorithm used is SSP, mentioned previously, which was efficient enough for our purposes and easy to implement. Clearly, the Hungarian method does not scale, while the min-cost flow does very well. While the min-cost flow is constant-time in this setting (although not necessarily so when using the SSP algorithm), there is a small variation for different numbers of sets of dominoes due to the time spent generating the problem.

Table 1. Comparing the Hungarian and Min-Cost Flow approaches to solving the assignment phase of domino portrait generation.

#Sets of	Time (in seconds)	
Dominoes	Min-Cost Flow	Hungarian
9	0.23	0.47
25	0.15	6.87
49	0.15	50.17
121	0.17	734.69
2,500	0.31	-
10,000	0.63	-

Secondly, we show the average quality of a random pattern (over 100 runs) for different number of sets of dominoes to support our claim that any random pattern provides a good bound on the quality of the domino portrait (see Table 2). The ILP model is solved with CPLEX using Bosch's model discussed in Section 3. In Table 2, the cost is the total cost of the optimal solution; we present both the average and best costs for the random pattern approach. It

is interesting to note that as the number of sets of dominoes is increased, the quality of the portrait generated from a random pattern is improving; we can find portraits that are 2.69% worse than the optimal cost found using ILP when using 225 sets of dominoes. A very important difference between methods here, of course, is that the random pattern-based portrait is generated in a fraction of a second, while ILP takes several hours for larger numbers of dominoes.

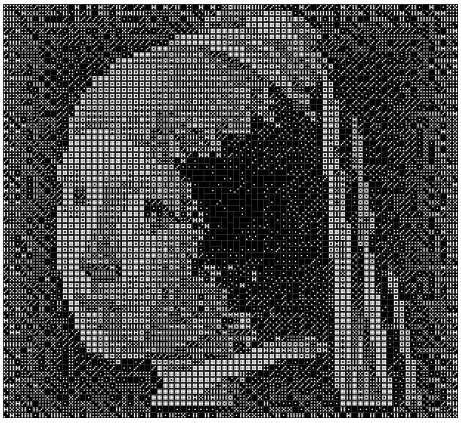
Table 2. Comparing the quality and speed of ILP and random patterns.

	ILP		**Two phase (100 runs)**			**Gap (%)**	
k	Cost	Time (s)	Avg Cost	Best Cost	Time (s)	Avg	Best
1	1,192	1.04	1,260	1,222	0.02	5.96	2.52
4	4,844	13.8	5,228	5,139	0.04	7.99	6.09
9	11,255	65.9	12,183	12,013	0.07	8.26	6.73
25	33,673	325.62	36,265	35,998	0.12	7.71	6.90
49	69,585	7,030.29	74,075	73,639	0.13	6.45	5.83
121	171,961	9,797.55	181,768	180,991	0.16	5.72	5.25
225	376,176	44,895.86	386,870	386,326	0.17	2.84	2.69

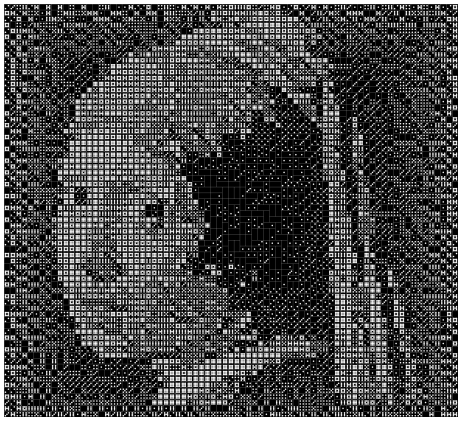
Table 3. Comparing the quality and speed of ILP against the flow-based approach using local search to improve the pattern.

	ILP		**LNS patterns**		
k	Opt Cost	Time (s)	Cost	Time (s)	Gap (%)
1	1,192	1.04	1,207	8.32	1.26
4	4,844	13.80	4,903	14.00	1.22
9	11,255	65.90	11,512	14.54	2.28
25	33,673	325.62	34,498	15.72	2.45
49	69,585	7,030.29	70,977	17.66	2.00
121	171,961	9,797.55	175,669	27.34	2.16
225	376,176	44,895.86	380,408	32.71	1.13

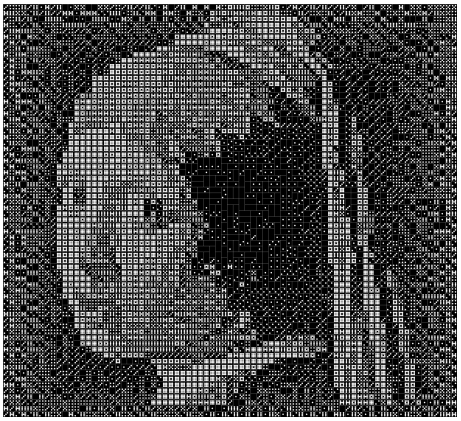
Thirdly, in Table 3 we show the results of the ILP formulation and the flow-based approach using local search over patterns which provide very good portraits within a few percent of the optimal value with orders-of-magnitude of speed-up in search time. The resulting images are indistinguishable visually from the optimum for "A Girl with a Pearl Earring" as shown on Figure 8. We show three portraits using 49 sets of dominoes corresponding to the optimal value obtained by the ILP, random pattern and local search approaches. For interesting sizes (between 9 and 225 sets of dominoes), the local search approach outperforms the ILP model in time without losing any relevant quality in the picture (gap no more than 2.45% and visually irrelevant).



(a) Optimal (ILP)



(b) Random Pattern + Min Cost Flow



(c) Large Neightbourhood Search

Fig. 8. Comparing the output of the ILP versus our full approach combining min-cost flow and local search.

We could not solve the ILP model for portraits requiring more than 225 sets of dominoes because of memory problems. However, even portraits requiring $10,000$ sets of dominoes ($55,000$ dominoes) are not a challenge for our approach. In fact, the larger the number of domino sets we use, the less we need to optimize the pattern using local search. In Figure 9 we show a very complex portrait of Alan Turing generated using 361 sets of dominoes, and report the times required by the min-cost flow and local search phases in its caption. We also report that using $10,000$ sets of dominoes we can generate portraits even faster because we have a much shorter local search step.

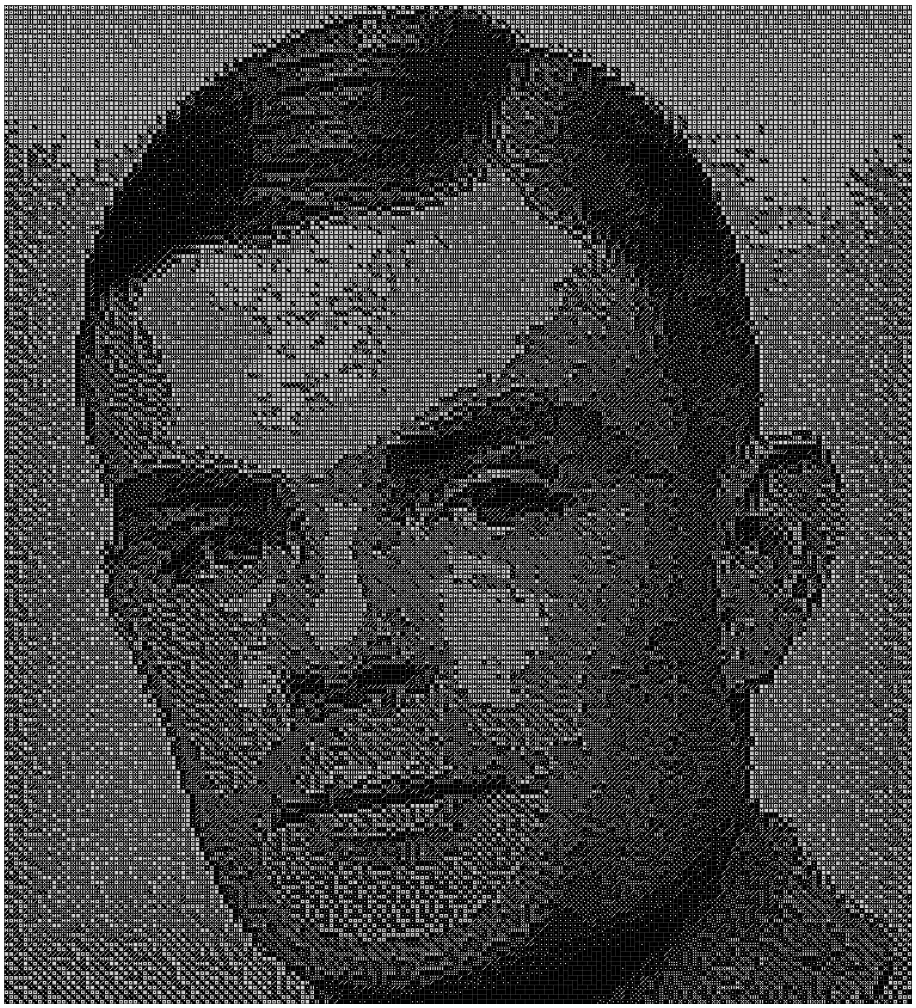


Fig. 9. A domino portrait of Alan Turing generated by our approach using 361 sets of "double nine" dominoes, i.e. 19,855 individual dominoes. The min-cost flow phase for this portrait required 0.194 seconds, and the local search phase required 33.965 seconds. We also generated a much larger portrait using 10,000 sets of dominoes (55,000 individual dominoes), which required 0.539 seconds and 26.285 seconds for the min-cost flow and local search phases, respectively. This portrait is not included in the paper since it would look almost like a standard grayscale image.

7 Conclusion

We have proposed a new solving technique for the domino portrait problem which is based on an original and efficient reformulation of part of the problem as a min-cost flow problem combined with local search. We show that we can obtain several orders-of-magnitude of speed-up to get high quality portraits within a few percent of the optimal value. This approach does not provide optimal solutions but produces high quality solutions within a couple of seconds. It is moreover very robust to the increase of the size of the problem.

Interesting ideas have been explored that might be useful in the context of packing problems with a positioning cost. The packing problem here is easy, as it is only made of rectangles of the same size, but the overall approach might be interesting in more complex and real-life applications where the objects are of different shapes.

Our application involves well known OR algorithms (Hungarian, Min-cost flow and sensitivity analysis of the flow), search techniques (large neighbourhood search, depth first search with constraint propagation) as well as an algorithm from the computer vision area (FAST) and is, therefore, well suited for teaching Operations Research. It has been used with great success at the Discovery Exhibition 2007 in Cork[1], a science outreach event for pupils aged between 10 and 16.

Acknowledgements

This work was supported by Science Foundation Ireland (Grant No. 05/IN/I886). We thank Robert Bosch for providing the inspiration for tackling this problem and for providing some useful feedback.

References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows - Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs NJ, 1993.
2. Elwyn Berlekamp and Tom Rogers. The mathemagician and pied puzzler: A collection in tribute to Martin Gardner. *AK Peters*, 1999.
3. Robert Bosch. Constructing domino portraits. *Tribute to a Mathemagician*, pages 251–256, 2004.
4. Kenneth C. Knowlton. Representation of designs. *U.S. Patent # 4,398,890 (August 16th, 1983)*.
5. Donald E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Addison-Wesley, 1993.
6. Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *ICCV*, pages 1508–1515, 2005.
7. Edward Rosten, Gerhard Reitmayr, and Tom Drummond. Real-time video annotations for augmented reality. In *ISVC*, pages 294–302, 2005.
8. Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP*, pages 417–431, 1998.

[1] http://www.corkcity.ie/discovery/