Domino Portrait Generation: A Fast and Scalable Approach

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

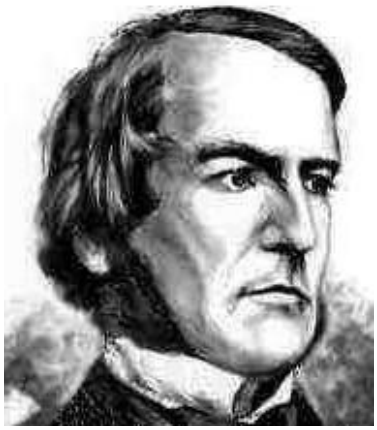
Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{h.cambazard|j.horan|e.omahony|b.osullivan}@4c.ucc.ie

Abstract. A domino portrait is an approximation of an image using a given number of sets of dominoes. This problem was first formulated in 1981 by Ken Knowlton in a patent application, which was finally granted in 1983. 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 the portraits. Empirically, we show that we obtain many orders of magnitude reduction in search time.

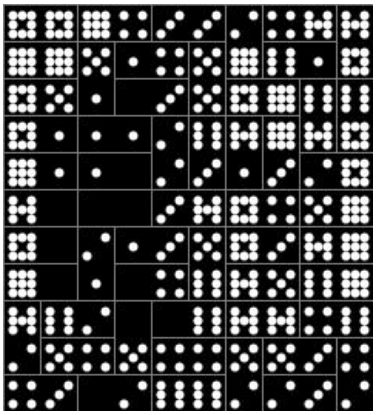
1 Introduction

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.

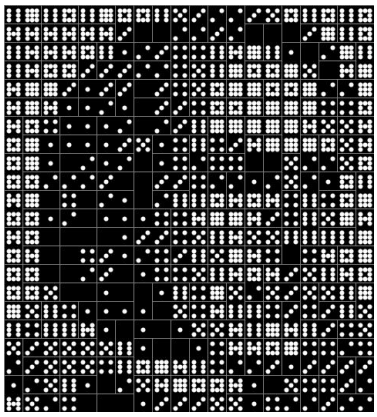
The nice property of "double nine" domino sets is that they give a wide range of shades from complete black (the blank domino) to a bright white (the double nine domino). A set of dominoes gives us a constrained palette of monochrome shades, which we can use to produce images. We say that the palette is constrained for two reasons. Firstly, each set of dominoes contains only one domino



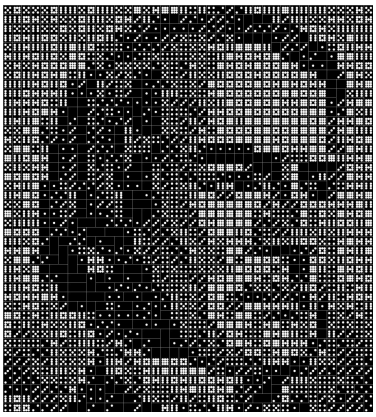
(a) George Boole.



(b) 1 set of dominoes.



(c) 4 sets of dominoes.



(d) 16 sets of dominoes.

Fig. 1. Domino portraits of George Boole.

of each type. Secondly, we are not allowed to break dominoes into two parts, but rather use the entire domino.

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 ap-

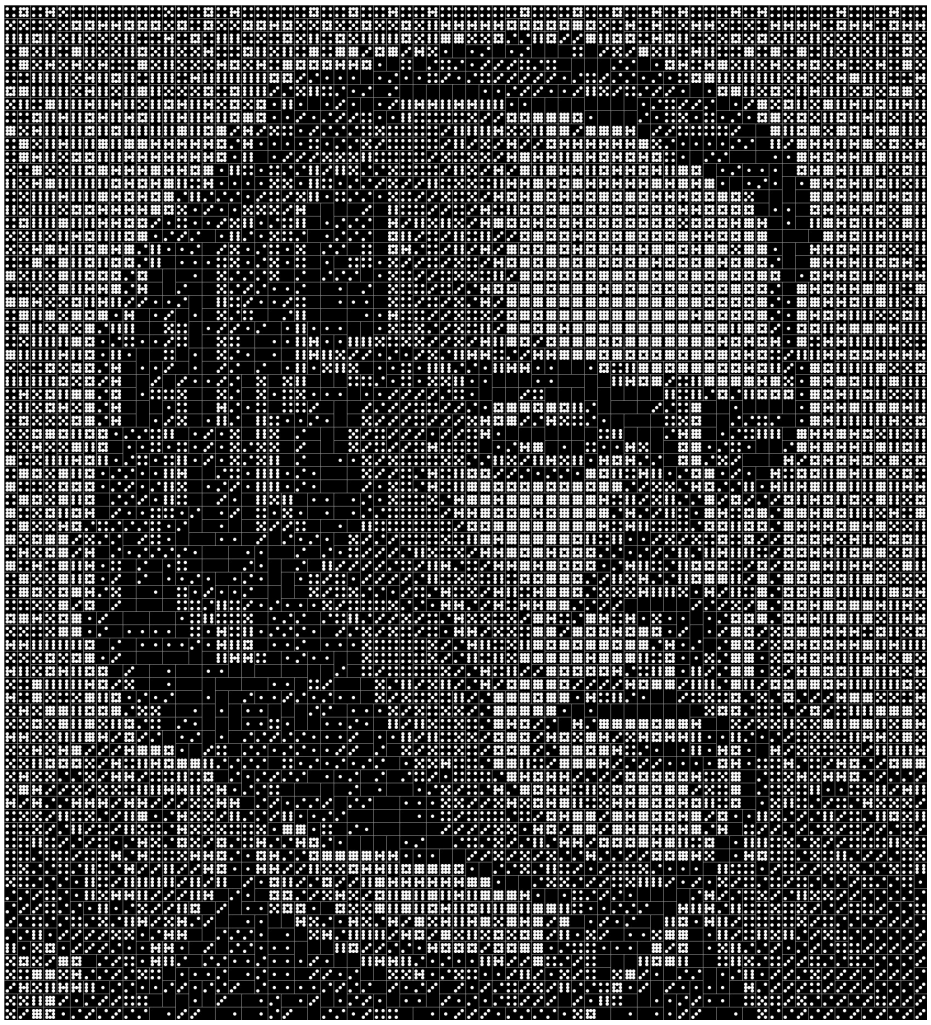


Fig. 2. A domino portrait of George Boole generated by our approach using 49 sets of "double nine" dominoes, i.e. $49 \times 55 = 2695$ individual dominoes.

proach 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 visually to be used for teaching and communication purposes.

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 dominoes. 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 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. 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).

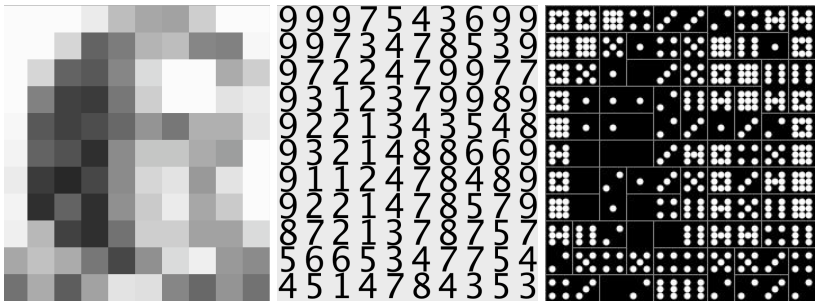
We consider rendering images using sets of "double nine" dominoes. There are 55 dominoes in a complete set: 10 dominoes with equal face values in both halves, i.e. all dominoes with face valuations $(0,0),(1,1)\ldots,(9,9)$ along with an additional 45 non-equal face dominoes with face values in $\{(v_1,v_2)|v_1 \in \{0,\ldots,8\}, v_2 \in \{v_1+1,\ldots,9\}\}$. 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^2 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 $55 \times k$ dominoes, 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 $0 \ldots 9$.

(b) The result of the scaling process.

(c) An example placement of dominoes.

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 pattern 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 dominoes 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 can generally provide a 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.

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

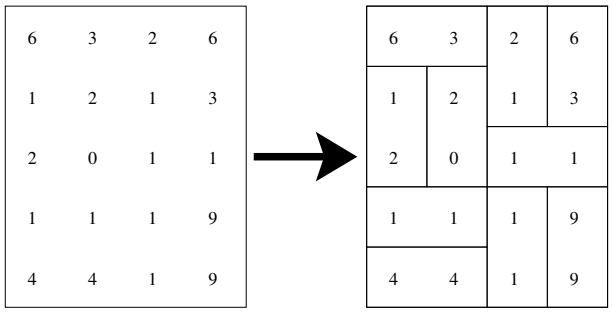


Fig. 4. An example of a pattern on the right that covers the grey matrix on the left.

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 ← the first row containing an empty cell
3:    j ← 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 a 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:

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

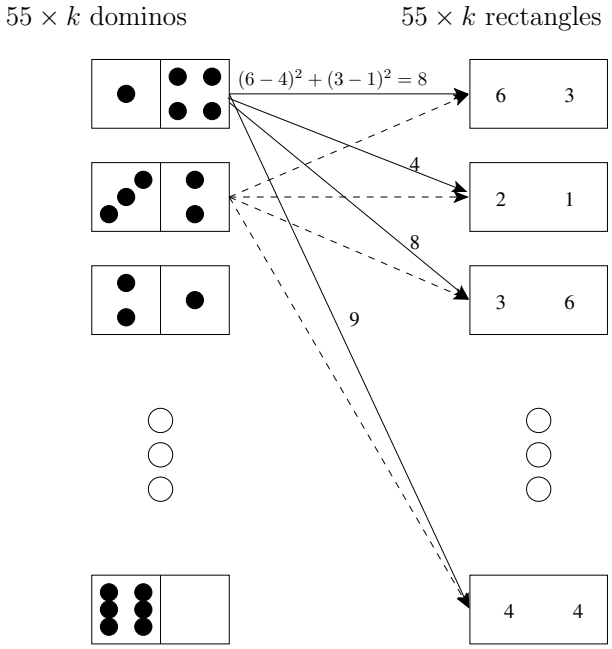


Fig. 5. An example of the assignment problem to be solved once the pattern is known.

Solving the assignment problem can be done very efficiently using the Hungarian method in $\mathcal{O}(n^3)$ time. 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. The Hungarian method is quickly unable to scale beyond 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.

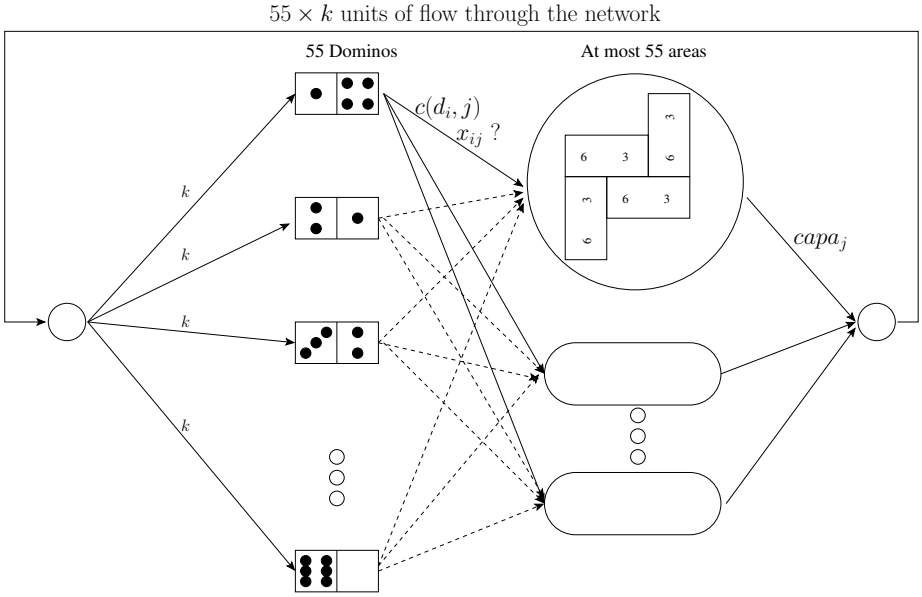


Fig. 6. The assignment problem translated as a min-cost flow problem.

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.[1] 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 [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 $\mathcal{O}(n \times max_{j \in \{1 \ldots nbArea\}}(capa_j))$ 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 $\mathcal{O}((m \log n)(m + n \log n))$, where n is the number of nodes and m is the number of arcs. This means that for our min-cost flow formulation 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:

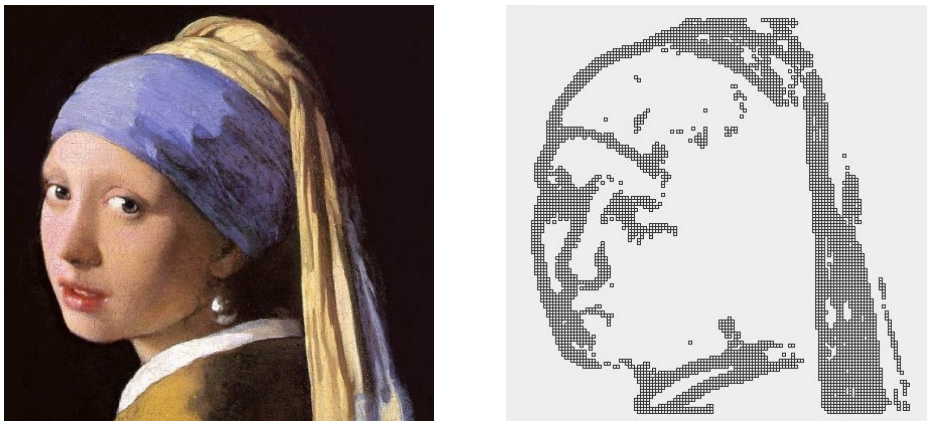
1. Identify the regions of the canvas where the grey values are unbalanced and thus, where the pattern might benefit from improvement. We denote as X the set of points (i, j) corresponding to those regions.

[1] When rendering the solution to the assignment problem, we place the dominoes with the lowest costs for each area in the centre of the image and work outwards. We found that this heuristic gave more pleasing portraits, while having no effect on the cost of the solution.

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 different place.
5. Return to Step 2 (above) as long as the average improvement (in percentage of the initial cost given by the random pattern) over the last I iterations remains above a threshold T (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 already chosen points. This is to maximize the impact of the improvements during the initial runs 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 suited to portrait generation 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 ($M = \{15, 20, 25\}$ is the setting used in our experiments). The enumeration is performed using the propagation described in Algorithm 1 in a complete backtracking search. Finally, the problem of finding the optimal flow due to small changes in $capa_j$ 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. We used his approach as a baseline in our experiments. We used Vermeer's "A Girl with a Pearl Earring" (Figure 7(a)) as our main benchmark image since Bosch used it to report many of his results. However, we also considered ten other images to demonstrate the utility of our methods.

The times quoted in our experiments are the totals taken to generate and solve the respective models, but they do not include the time taken to convert the solution into a viewable image. Our experiments were run in a single thread on a Dual Quad Core Xeon CPU @ 2.66GHz with 12MB of L2 cache per processor and 16GB of RAM overall, running Linux 2.6.25 x64.

6.1 The Linear Programming Model

We show results obtained using Bosch's ILP model discussed in Section 3 which was solved using CPLEX (version 10.0.0). The ability of CPLEX to solve such large integer formulations is quite impressive, so we were also able to compare the quality of the linear relaxation (LP) and the performance of the ILP. Table 1 shows the results for different values of k on the "A Girl with a Pearl Earring".

Table 1. Comparing the ILP and LP on the "Girl with a Pearl Earring".

	LP		**ILP**		
k	Cost	Time (s)	Cost	Time (s)	Nodes
1	1,192	0.33	1,192	0.44	0
4	4,844	1.75	4,844	3.46	0
9	11,254	6.86	11,255	76.43	70
25	33,673	47.78	33,673	81.22	0
49	69,585	148.01	69,585	328.47	0
64	98,906	273.54	98,908	263,152.71	12838
121	171,960	815.43	171,961	11,413.08	121
225	374,393	8,616.12	374,393	22,390.44	0
256	463,814	16,310.84	463,814	25,693.15	0

Clearly, the LP provides the optimum value in all cases but three, where a very small gap remains. The $k = 64$ instance is particularly difficult for CPLEX, which needs around 3 days to solve it optimally. However, when the LP cost is equal to the optimal value, the solution found is still not integer. In these situations, CPLEX applies a heuristic to compute the first upper bound (before branching) and obtains the integer optimal solution without branching (thus the values of 0 for the nodes). The formulation is, therefore, clearly not unimodular. We considered whether the small gaps observed in the three exceptional cases could be due to rounding mistakes, and ran further experiments on 10 other images using two values of k (9 and 25). On the new 20 instances, all of them, except the four presented in Table 2, have an LP cost equal to the optimal value and were solved in 0 nodes by CPLEX. The four exceptions show two new cases: picture 4 ($k = 9$) has an LP cost less than the optimal value but the problem is solved without branching, and picture 7 ($k = 25$) has an LP cost equal to the optimal value but branching is needed to find an integer solution.

Table 2. Comparing the LP and ILP on various images.

		LP		**ILP**		
picture	**k**	Cost	Time (s)	Cost	Time (s)	Nodes
1	9	2,018	2.83	2,019	18.64	5
4	9	12,215.33	10.26	12,216	17.31	0
3	25	7,721	12.94	7,722	823.21	510
7	25	42,757	64.70	42,757	471.15	10

The LP was also solved using the Barrier algorithm which gave the exact same answer than the Simplex thus making more unlikely the scenario of rounding mistakes. Despite all our efforts we have not been able to show that the problem is NP-hard, nor that it is polynomial.

6.2 Evaluating the Approximate Methods

Comparing the Assignment and Min-Cost Flow Formulations. We sought to compare the performance of the min-cost flow and Hungarian method to demonstrate the scalability of the flow algorithm (Table 3). The flow algorithm used is Successive Shortest Path (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.

The min-cost flow behaves in a constant-time manner in this setting (although we are using the SSP algorithm). The small variation in time for different settings of k is due to the time spent generating the problem.

Evaluating the Quality of a Random Pattern. We show on Table 4 the average quality of a random pattern for different number of sets of dominoes on

Table 3. Comparing the Hungarian and Min-Cost Flow approaches to solving the assignment phase of domino portrait generation.

#Sets of Dominoes	Time (in seconds) Min-Cost Flow	Hungarian
9	0.06	0.22
25	0.07	4.50
49	0.06	36.25
121	0.07	536.00
2500	0.12	-
10000	0.21	-

"A Girl with a Pearl Earring". We present both the average and best costs for the random pattern approach over 1000 runs. 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 improves; we can find portraits that are 2.65% worse than the optimal cost found using ILP when using 256 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 the ILP takes several hours for larger numbers of dominoes. Table 5 presents the same data on 10 different images[2] for 225 sets of dominoes. On the 11 pictures evaluated the random pattern generally provides a good bound on the quality of the domino portrait with an average gap of 11.8 % to the optimum for 225 dominoes.

Table 4. Comparing the quality and speed of ILP and random patterns on "A Girl with a Pearl Earring".

	ILP		**Two phase (1000 runs)**			**Gap (%)**	
k	Cost	Time (s)	Avg Cost	Best Cost	Time (s)	Avg	Best
1	1,192	0.44	1,263	1,212	0.01	5.96	1.68
4	4,844	3.46	5,231	5,105	0.02	7.99	5.39
9	11,255	76.43	12,185	11,931	0.04	8.26	6.01
25	33,673	81.22	36,270	35,868	0.06	7.71	6.52
49	69,585	328.47	74,070	73,523	0.06	6.45	5.66
121	171,961	11,413.08	181,793	180,990	0.07	5.72	5.25
225	374,393	22,390.44	386,874	386,022	0.08	3.33	3.11
256	463,814	25,693.15	476,121	475,429	0.08	2.65	2.50

Evaluating the parameters of the LNS Approaches. We first evaluated the impact of the two main parameters of the local search approach, namely the

[2] The grid of grey values corresponding to the pictures (and the pictures themselves) used as benchmark are available at this address : http://www.4c.ucc.ie/datasets/dominoes

Table 5. Comparing the quality and speed of ILP against random patterns for 10 different images for 225 dominoes.

	ILP		Two phase (1000 runs)			Gap (%)	
Image	Cost	Time (s)	Avg Cost	Best Cost	Time (s)	Avg	Best
1	134,974	3,134	157,691	156,026	0.05	16.83	15.6
2	131,025	2,465	147,922	146,825	0.04	12.9	12.06
3	71,440	4,486	93,492	92,168	0.05	30.87	29.01
4	166,709	3,455	185,390	184,193	0.06	11.21	10.49
5	152,538	17,152	169,161	168,102	0.04	10.9	10.2
6	124,073	3,019	137,284	136,347	0.03	10.65	9.89
7	313,529	6,553	321,985	321,309	0.04	2.7	2.48
8	141,171	16,443	150,111	149,474	0.02	6.33	5.88
9	68,740	15,602	82,319	81,360	0.03	19.75	18.36
10	238,139	8,567	248,075	247,297	0.03	4.17	3.85

number dominoes removed to define the size of the neighborhood (M), and the number of iterations (I). It is on the parameter I that we compute the average improvement of the objective function, as compared with the initial starting cost of a random pattern, to stop the algorithm if this value drops below a threshold (T). We set T to a very small value (6.10^{-5}) so that most of time, the algorithm stops after I non-improving iterations. T is not set to 0 because we met a few cases of very slow convergence. We ran this experiment on Picture 3 where the random pattern gives the worse performance. Table 6 reports the results. As expected, as M increases, so does the running time and image quality. The parameter I, for a given M also improves the quality of the resultant image.

Table 6. Evaluation of the impact of the parameters of the LNS with algorithm on a portrait of Astor Piazolla (picture number 3).

		LNS patterns (100 runs)			Gap (%)	
M	**I**	Avg Cost	Best Cost	Time (s)	Avg	Best
10	10	92,713	90,860	0.44	29.78	27.18
10	20	90,283	83,172	1.44	26.38	16.42
10	30	86,601	80,100	2.92	21.22	12.12
15	10	89,913	81,618	7.39	25.86	14.25
15	20	80,940	77,298	27.67	13.3	8.2
15	30	78,806	77,268	34.37	10.31	8.16
20	10	80,865	76,392	202.58	13.19	6.93
20	20	77,257	76,392	264.42	8.14	6.93
20	30	77,129	76,382	274.08	7.96	6.92
25	10	78,023	75,440	2,344.88	9.21	5.6
25	20	76,022	75,430	2,693.78	6.41	5.59

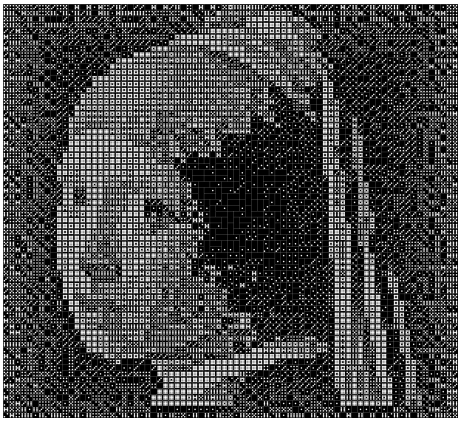
Comparing the ILP and the LNS Approach. In Table 7 we show the results of the ILP formulation and the flow-based approach using local search over patterns which provide very good portraits within a few percent of the optimal value with orders-of-magnitude of speed-up in search time. The resulting images for "A Girl with a Pearl Earring" are shown in 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 256 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%).

Table 7. Comparing the quality and speed of ILP against the flow-based approach (M = 15, I = 30) using local search to improve the pattern on "A Girl with a Pearl Earring".

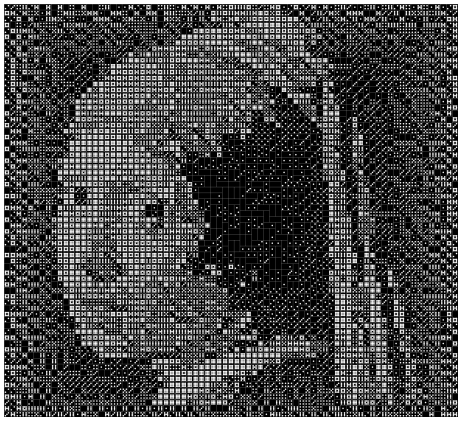
	ILP		**LNS M = 15 I = 30**			**Gap (%)**	
k	Opt Cost	Time (s)	Avg Cost	Best Cost	Time (s)	Avg	Best
1	1,192	0.44	1,207	1,206	4.49	1.26	1.17
4	4,844	3.46	4,906	4,879	7.51	1.28	0.72
9	11,255	76.43	11,529	11,359	6.64	2.43	0.92
25	33,673	81.22	34,498	34,264	7.8	2.45	1.76
49	69,585	328.47	70,952	70,587	9.22	1.96	1.44
121	171,961	11,413.08	175,615	175,098	14.19	2.12	1.82
225	374,393	22,390.44	380,313	378,858	16.84	1.58	1.19
256	463,814	25,693.15	470,799	468,285	12.91	1.51	0.96

We performed a final experiments on the 10 different images by fixing the number of sets to 225 and running the LNS with two sets of parameters (M = 15, I = 30 and M = 20, I = 20) which offer different tradeoff in time and quality. The resulting algorithm can produce portraits in average within 5 and 7 % of the optimal one in around 10 seconds or within 4 and 5 % in around 2 minutes. We generated many portraits in various circumstances using this software (science discovery event, lab visitors, etc.) using the fast setting M = 15, I = 30.

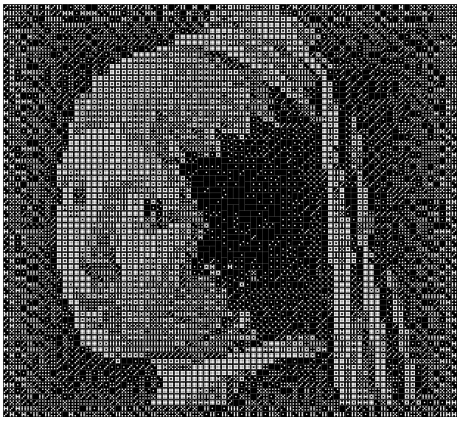
We could not solve the ILP model for portraits requiring more than 256 sets of dominoes because of memory problems. However, even portraits requiring $10,000$ sets of dominoes ($55,000$ dominoes) are not a challenge for our approach. In fact, the larger the number of domino sets we use, the less we need 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.



(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.

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 showed 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 quickly. It is moreover very robust to the increase of the size of the problem. However, the computational complexity class of the domino portrait generation problem remains open.

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

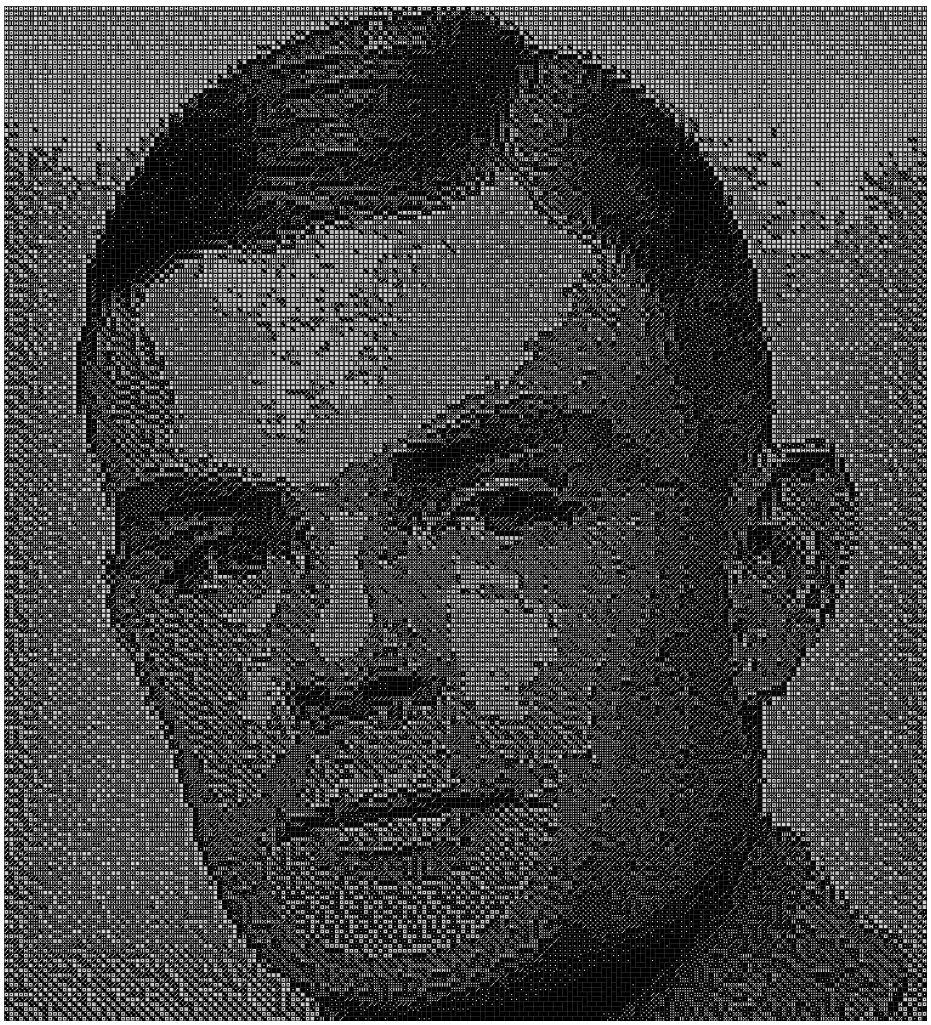


Fig. 9. A domino portrait of Alan Turing generated by our approach using 361 sets of "double nine" dominoes, i.e. 19,855 individual dominoes. The min-cost flow phase for this portrait required 0.175 seconds, and the local search phase required 16.07 seconds. We also generated a much larger portrait using 10,000 sets of dominoes (55,000 individual dominoes), which required 0.335 seconds and 9.265 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.

Table 8. Comparing the ILP and LNS (two sets of parameters) on 10 images for 225 sets of dominoes on 100 runs.

	ILP		**LNS M=15 I=30**			**LNS M=20, I=20**		
				Gap (%)			**Gap (%)**	
	Cost	Time (s)	Time(s)	Avg	Best	Time(s)	Avg	Best
1	134,974	3,134	12.54	9.15	6.05	107.64	6.9	5.64
2	131,025	2,465	11.42	8.52	3.52	143.06	4.5	3.07
3	71,440	4,486	32.22	10.31	8.16	254.76	8.14	6.93
4	166,709	3,455	5.42	9.7	2.72	141.95	4.24	1.97
5	152,538	17,152	2.67	10.19	9.22	65.99	8.82	4.17
6	124,073	3,019	11.66	4.96	3.65	97.77	3.74	3
7	313,529	6,533	1.2	2.53	2.37	8.03	2.51	2.36
8	141,171	16,443	6.73	3.84	2.87	59.79	3.16	2.7
9	68,740	15,602	19.93	7.07	5.9	148.65	5.91	5.09
10	238,139	8,567	2.14	3.55	2.82	24.79	3.1	1.61
average		8085.6	10.59	6.98	4.73	105.24	5.10	3.65
median		5509.5	9.08	7.80	3.59	101.51	4.37	3.04

it comprises rectangles of the same size, but the 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, thus, well suited for teaching Operations Research. It has been also successfully used at the Discovery Exhibition in 2007 in Cork[3], 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 Thierry Benoist, Robert Bosch, Frederick Gardi, Emmanuel Hebrard and Alexandre Papadopoulos for their helpful discussions on the computational complexity of this problem.

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)*, 1983.

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

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.