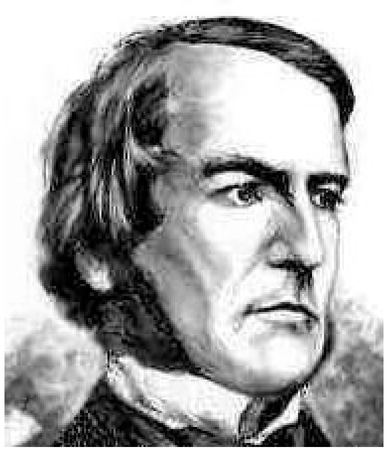
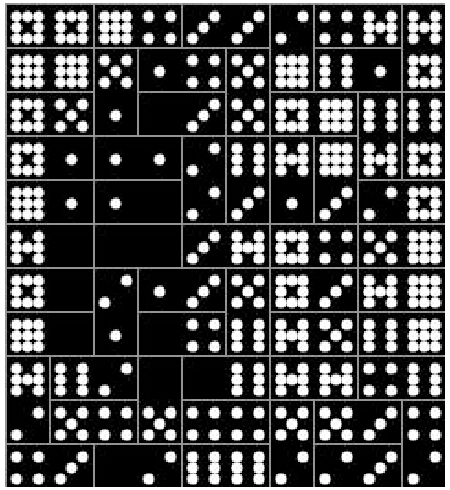
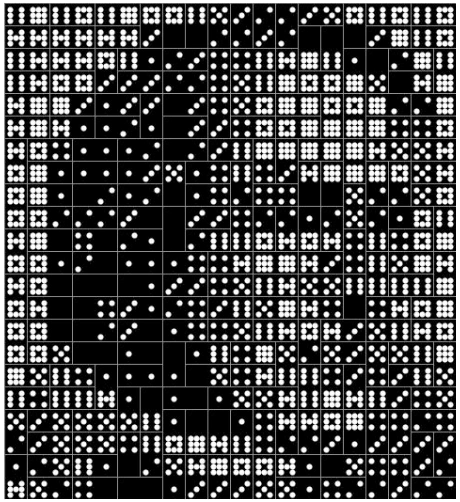
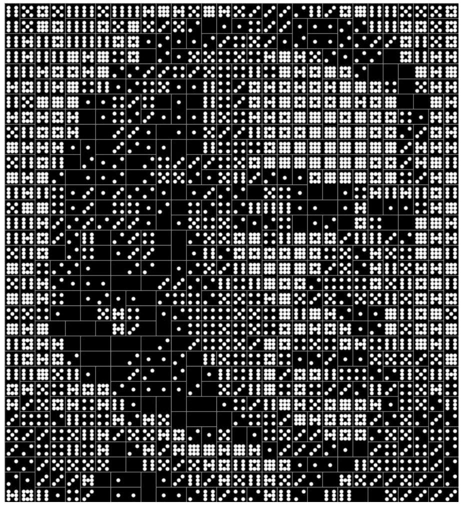
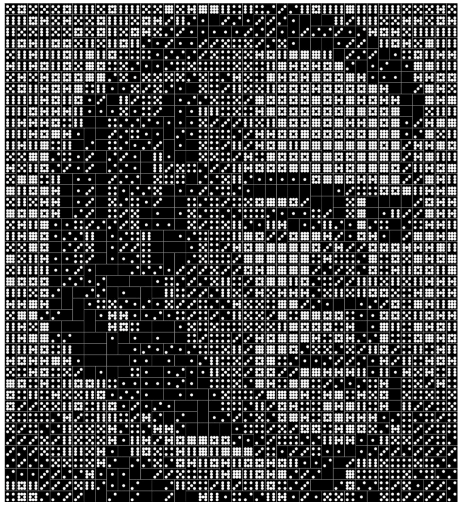


# Fast generation of domino portraits

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

Cork Constraint Computation Center

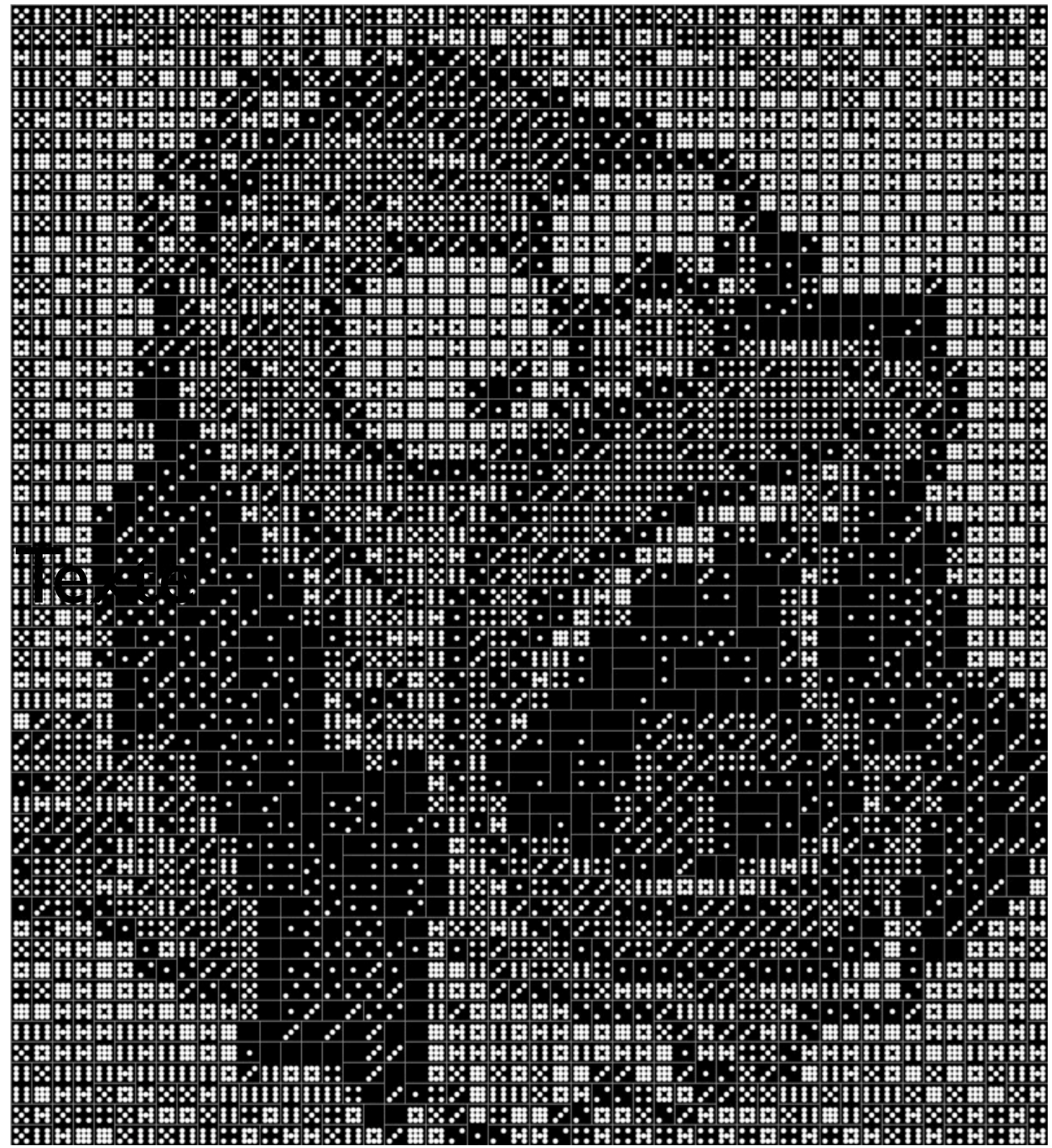
SFI grant number 05/IN/I886

# Outline

- The domino portrait problem
- The Integer linear programming approach of Robert Bosch
- A two-step approach
  - flow-based formulation of the problem
- Applications

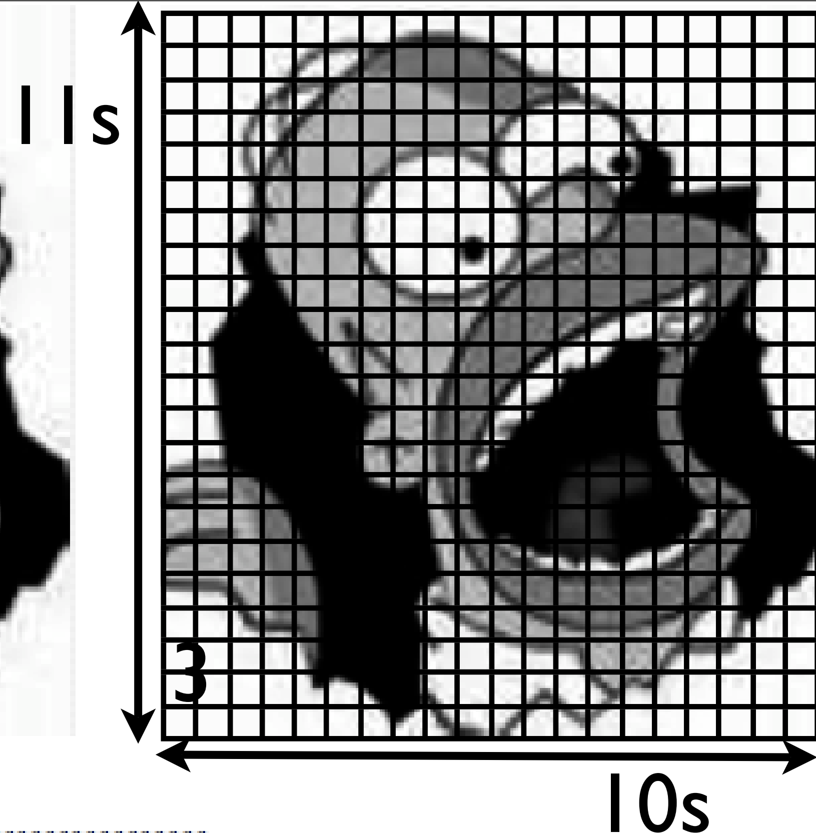
# Domino portraits

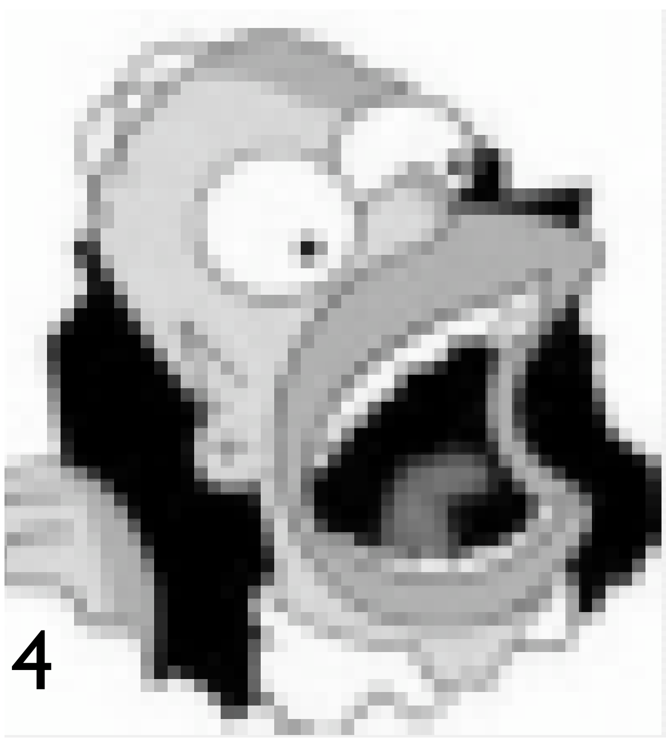


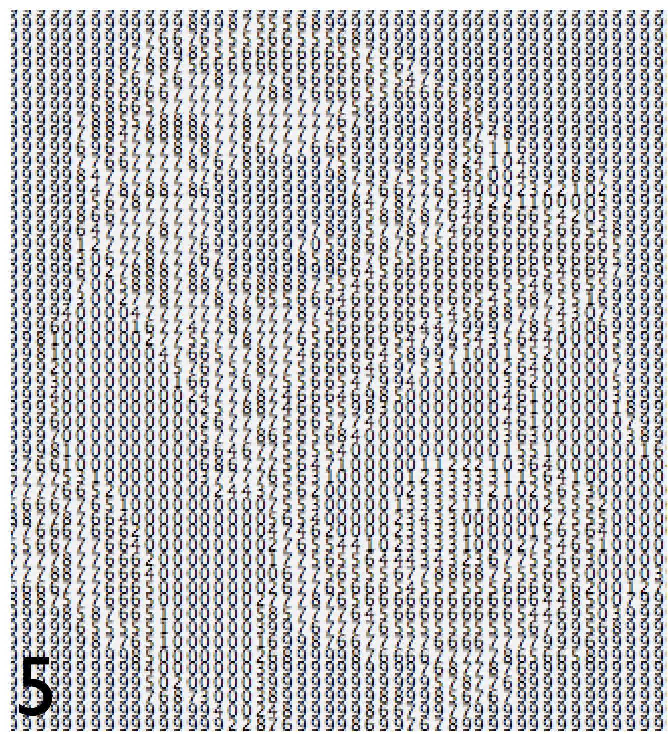
k sets of double nine dominos

[Knowlton, Representation of design 1983]
[Knuth, The Stanford GraphBase,1993]
[E. Berlekamp and T. Rogers, The mathemagician and pied puzzler, 1999]
[Bosh, constructing domino portraits, 2004]





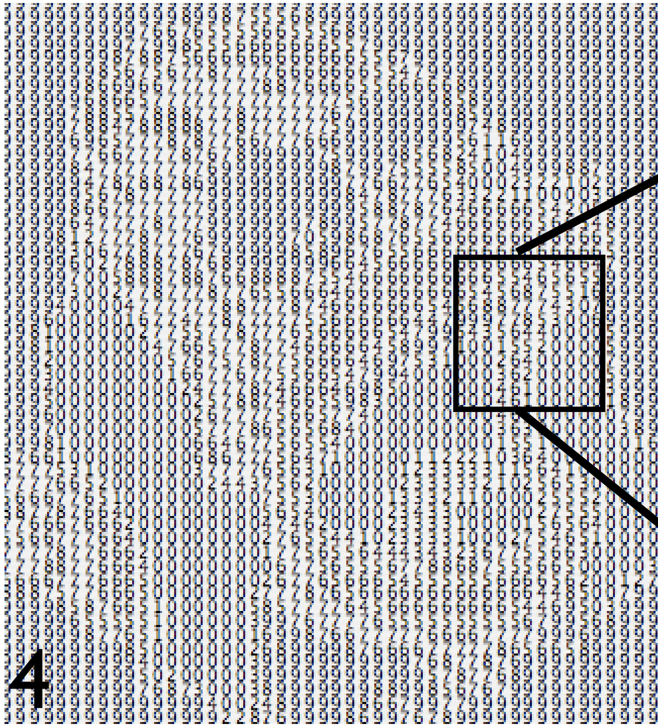



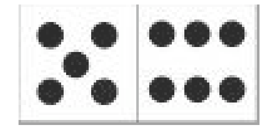

$k = s^2$

55 x k
dominos

110 x k
cells




| | | | |
|---|---|---|---|
| 6 | 3 | 2 | 6 |
| 1 | 2 | 1 | 3 |
| 2 | 0 | 1 | 1 |
| 1 | 1 | 1 | 9 |
| 4 | 4 | 1 | 9 |



(5 - 1)^2 +
(6 - 9)^2 = 25

Domino k is denoted $d_k = [p_k^1, p_k^2]$

The grayscale value of cell (i,j) is denoted $g_{i,j}$

Domino $d_k$ is placed on cells $(i_k^1, j_k^1)$ $(i_k^2, j_k^2)$

Pb : $Minimize \sum_{d_k \in D} (p_k^1 - g_{i_k^1, j_k^1})^2 + (p_k^2 - g_{i_k^2, j_k^2})^2$

Use each domino exactly once

# The integer linear programming model

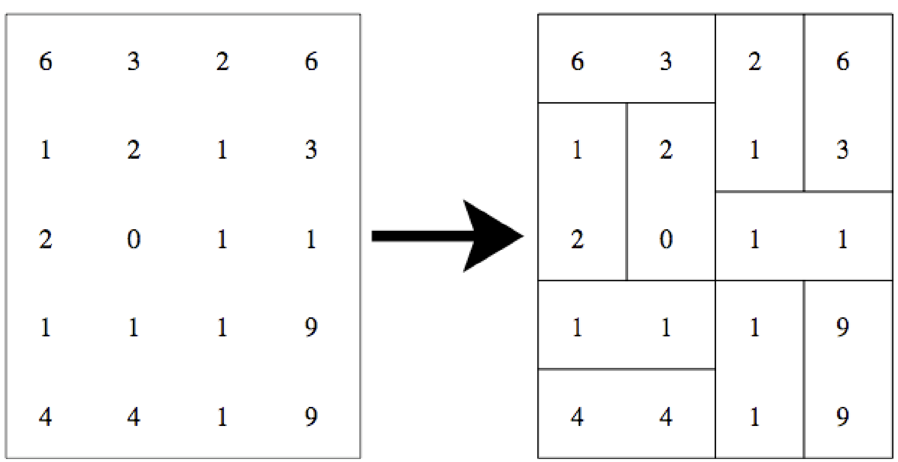
[R. Bosch. Constructing domino portraits. Tribute to a Mathemagician, 2004]

- The model :
  - Boolean variables to specify if a **given domino** is placed with a **given orientation** with its reference square in a **given cell** of the grid.
  - Each domino has to be used once
  - Each cell is covered by a unique domino
- Very large models
  - k=49 gives 1.063.300 variables and around 2 hours of computation

# A two-step approach

## 1. **Pattern generation**:

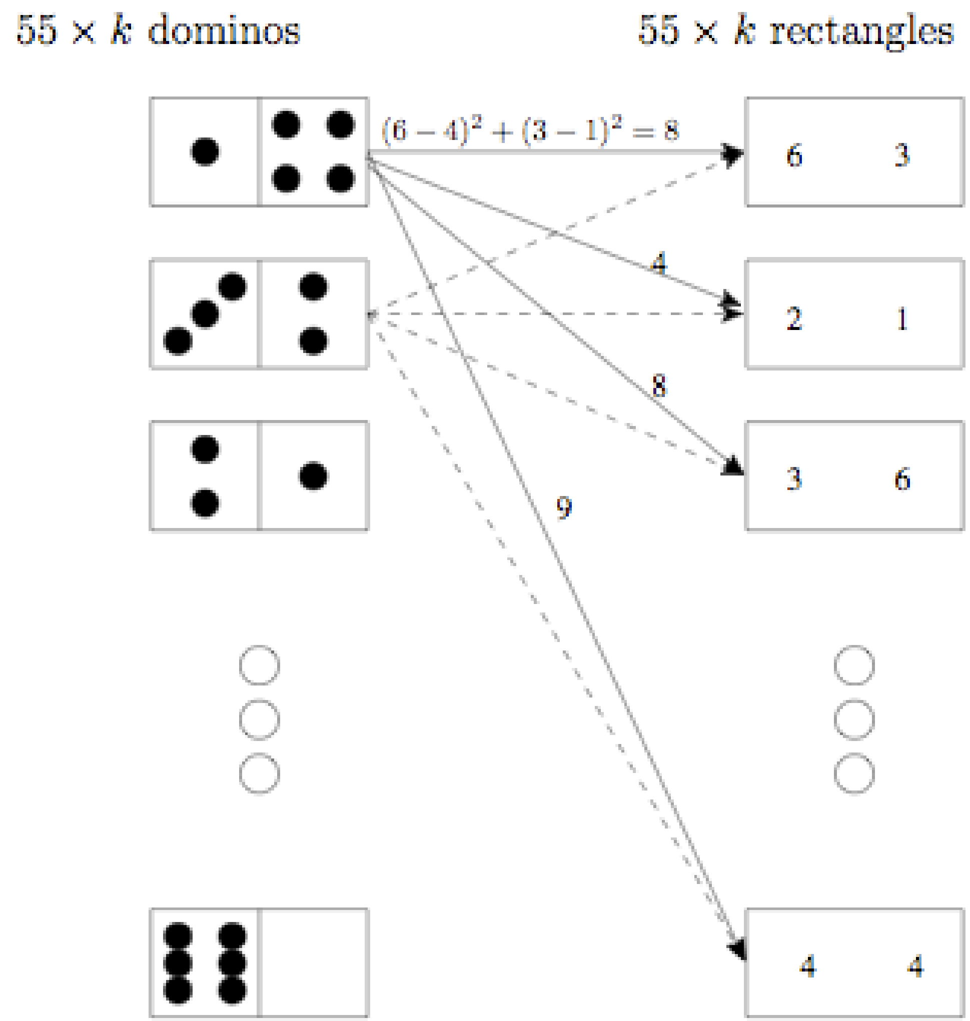
cover the grid with empty dominos (rectangles)




grid of gray values

pattern

## 2. **Assignment**:

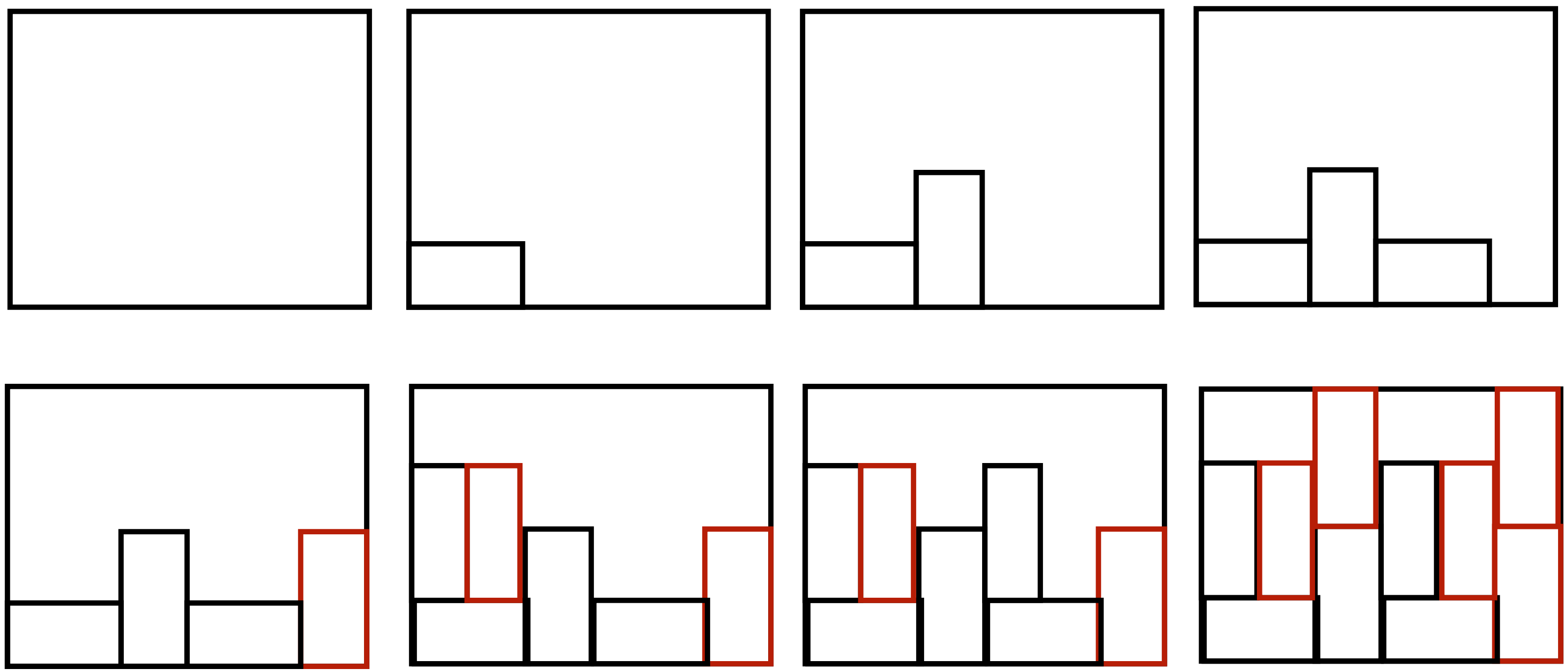


# A two-step approach

- Pattern: an arrangement of rectangles that covers the picture.
- Once the pattern is known **the remaining problem is polynomial**.
- Claim : any random pattern provide a good upperbound

# Generate a random pattern

- Randomly assign rectangles vertically of horizontally with simple propagation

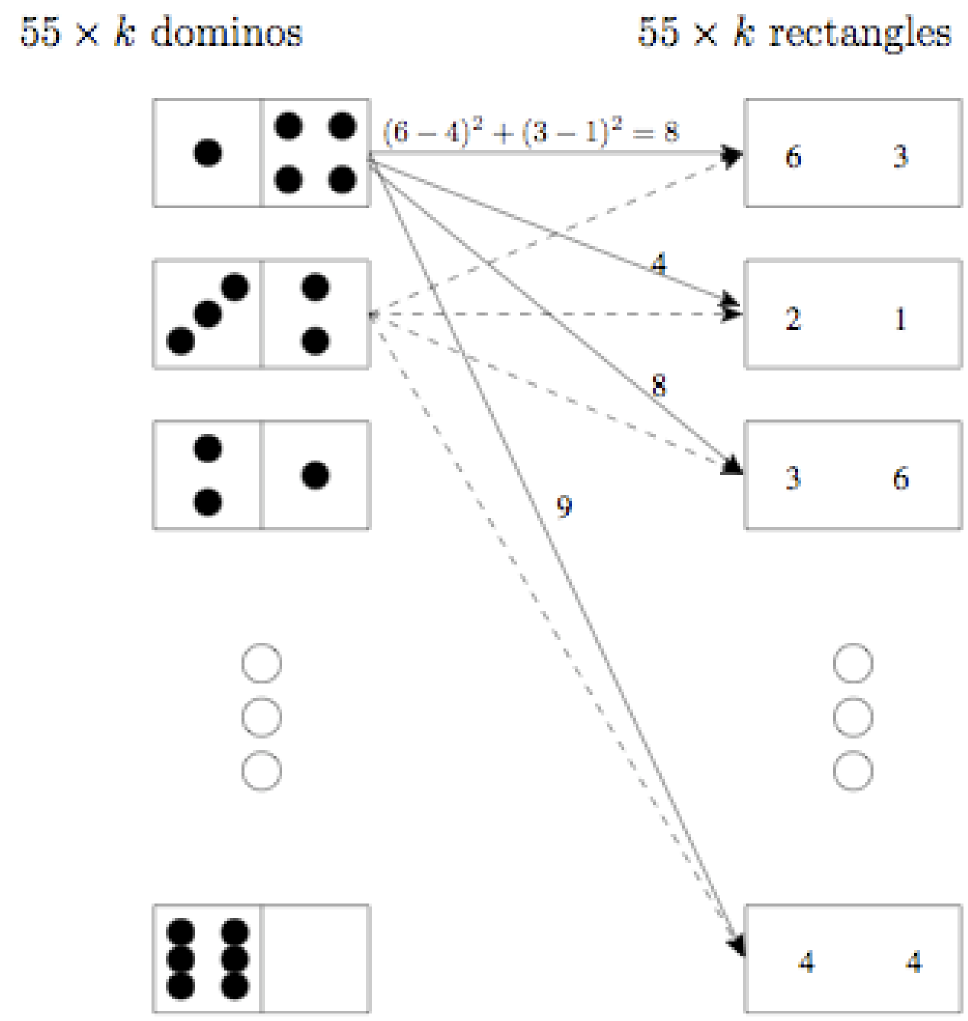


- Contradiction detection : a connected region of with an odd numbers of cells

- Partial restart by wiping out part of the grid

# Optimal assignment

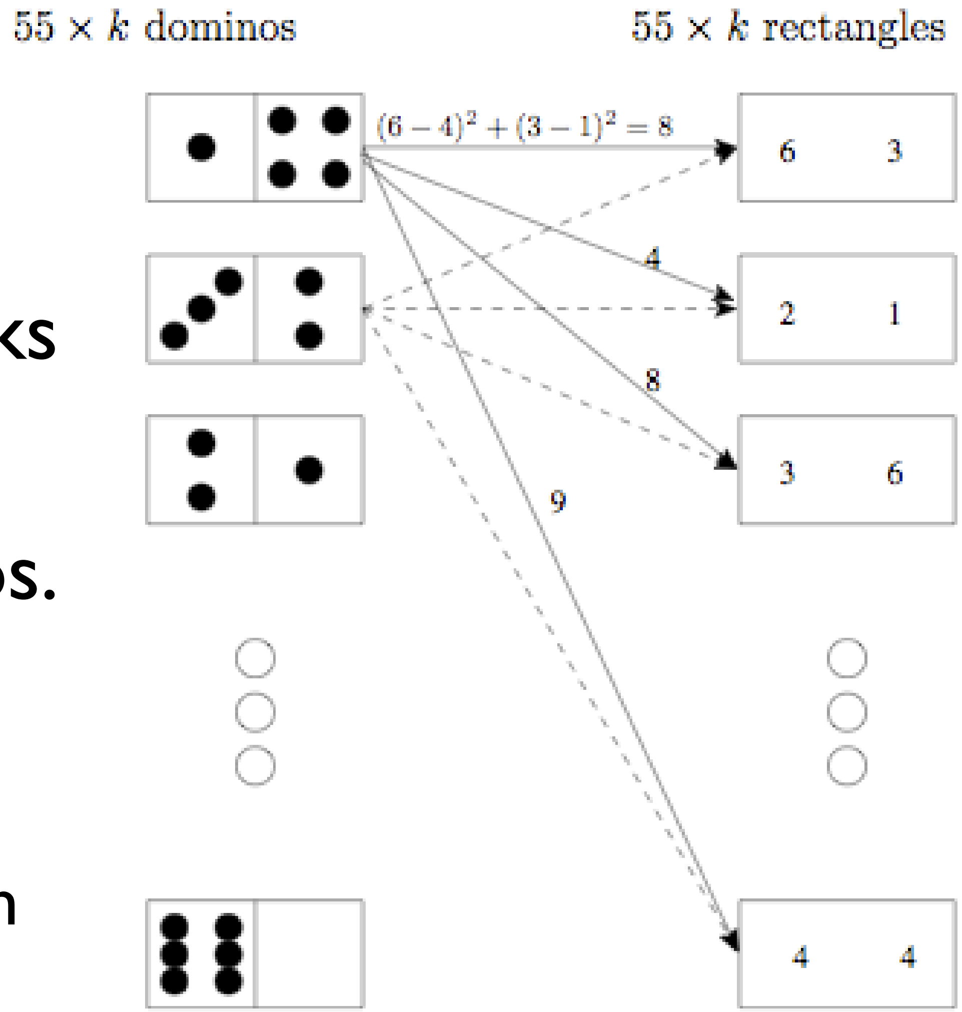
- The assignment problem is represented as a bipartite graph
- The cost of assigning a domino to a rectangle is given by its best orientation
- The **Hungarian** algorithm computes a minimum weight bipartite matching



# Optimal assignment

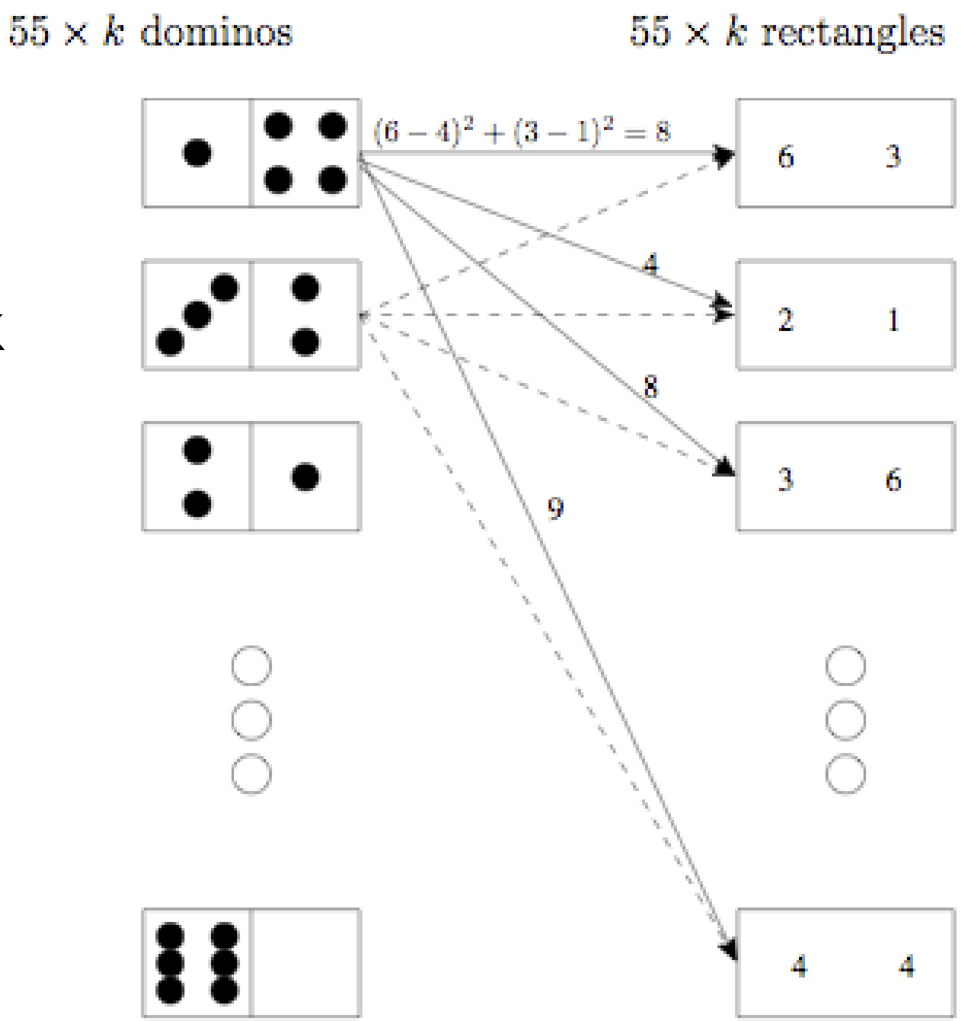
- Solving the assignment is polynomial
- Hungarian algorithm works in O(n^3) [Kuhn 55]
- k=100 gives 5500 dominos. The Hungarian does not scale !

  ~10 minutes of computation



# From n^3 to constant time

- Take advantage of **symmetries** :
  - Each domino is repeated **k** times
  - There are only **55 kinds** of rectangles defining **areas**
- **An area** is a set of rectangles of same cost

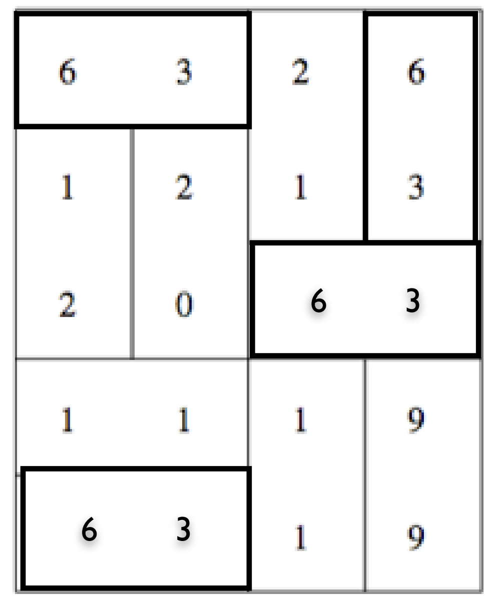


# From n^3 to constant time

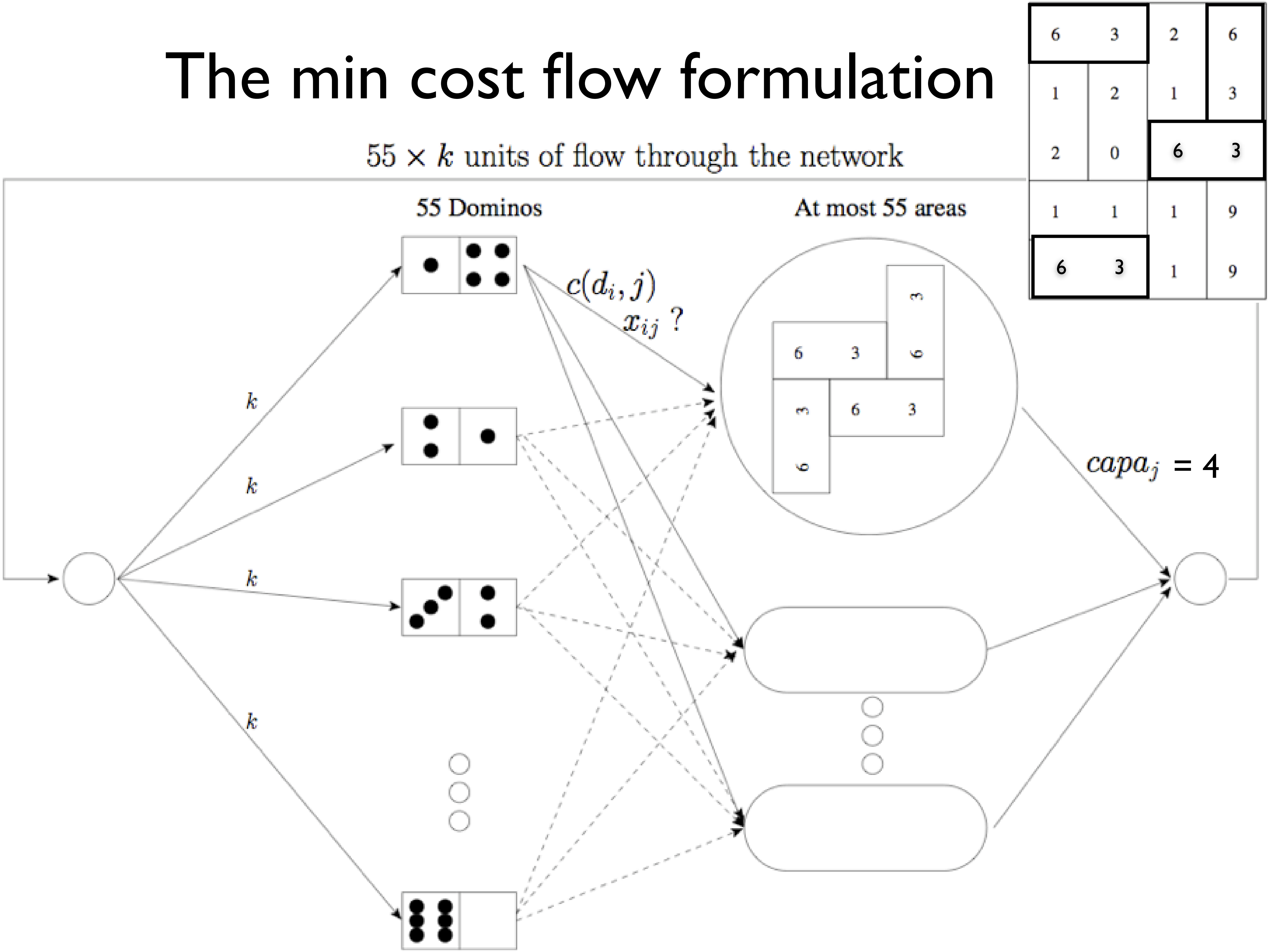
- Take advantage of **symmetries** :
  - Each domino is repeated **k** times
  - There are only **55 kinds** of rectangles defining **areas**
- **An area** is a set of rectangles of same cost

# From n^3 to constant time

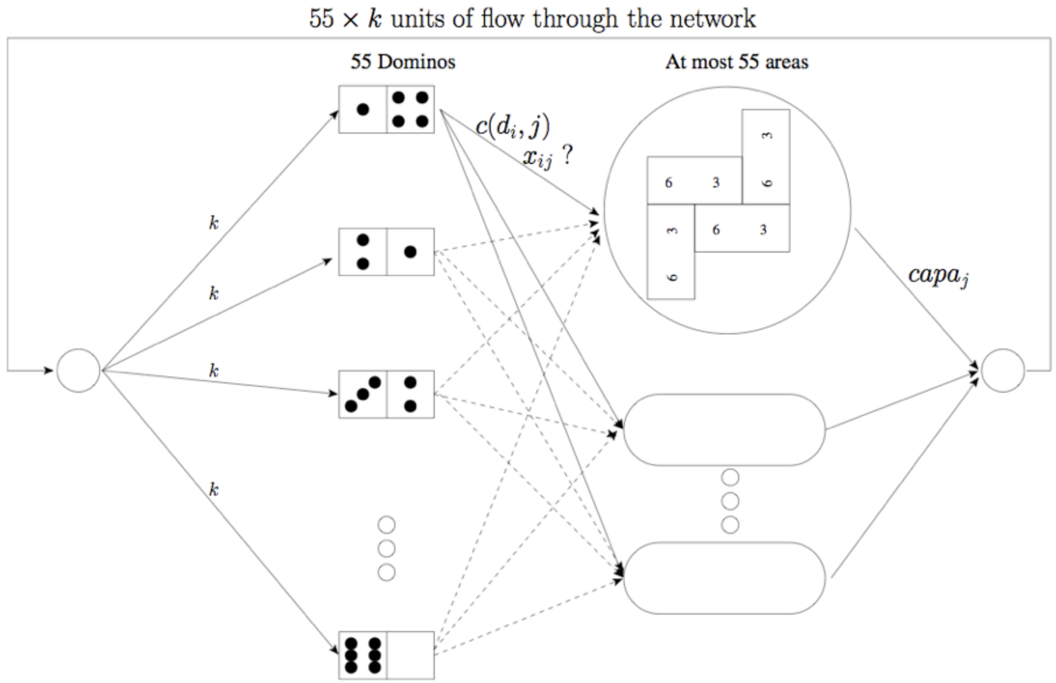
- Take advantage of **symmetries** :
  - Each domino is repeated **k** times
  - There are only **55 kinds** of rectangles defining **areas**
- **An area** is a set of rectangles of same cost



# The min cost flow formulation



# Optimal assignment




- There exists flow algorithm whose complexity does not depend on the capacities nor flow amount :
  - The Enhanced Capacity Scaling algorithm O((mlog(n))(m + nlog(n))
  - The Successive Shortest Path is enough for our needs O(n^2mU) where U is the maximum capacity

# Optimal assignment: min cost flow vs hungarian

- The flow formulation provides a constant time answer to the assignment problem !

| **k** | Flow Time (s) | Hungarian Time (s) |
|---:|---:|---:|
| 9 | 0.23 | 0.47 |
| 25 | 0.15 | 6.87 |
| 49 | 0.15 | 50.17 |
| 121 | 0.17 | 734.69 |
| 2500 | 0.31 | - |
| 10000 | 0.63 | - |

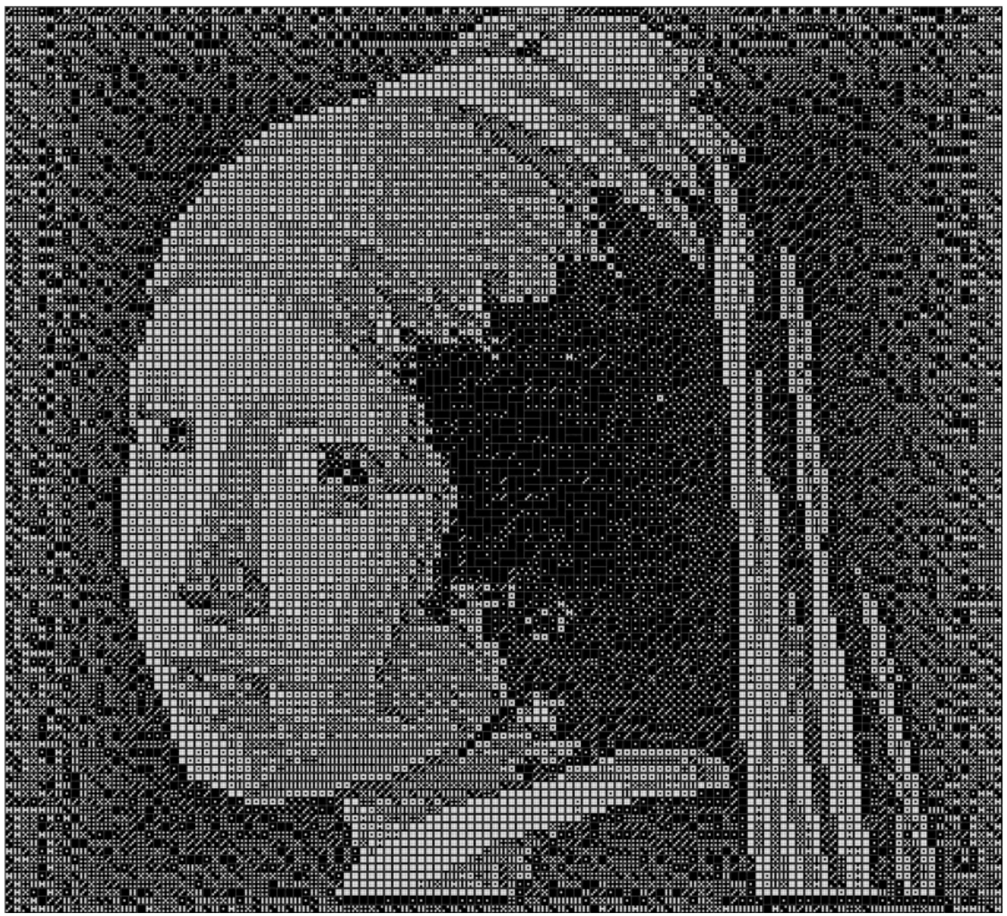
# First results

- Upper bounds obtained with a single random pattern

| | ILP | | Two steps (100 runs) | | | Gap | |
|---|---|---|---|---|---|---|---|
| k | Opt Cost | Time (s) | Avg Cost | Min Cost | Avg Time (s) | Avg Gap (%) | Min Gap (%) |
| 1 | 1192 | 1.04 | 1260 | 1222 | 0.02 | 5.96 | 2.52 |
| 4 | 4844 | 13.8 | 5228 | 5139 | 0.04 | 7.99 | 6.09 |
| 9 | 11255 | 65.9 | 12183 | 12013 | 0.07 | 8.26 | 6.73 |
| 25 | 33673 | 325.62 | 36265 | 35998 | 0.12 | 7.71 | 6.90 |
| 49 | 69585 | 7030.29 | 74075 | 73639 | 0.13 | 6.45 | 5.83 |
| 121 | 171961 | 9797.55 | 181768 | 180991 | 0.16 | 5.72 | 5.25 |
| 225 | 376176 | 44895.86 | 386870 | 386326 | 0.17 | 2.84 | 2.69 |

- Very good upperbounds but a visually detectable gap to the optimal solution

# Gap between
# ILP and random pattern + flow



Optimal solution

Upper bound (5.8 %)

# Searching among patterns

- We now know how to solve the problem very efficiently (constant time) once the pattern is known

How do we find the right pattern ?

- Main Idea : the pattern only matters where the grey values are varying.
- A change of the pattern that would not affect the size of the areas on the flow graph has no effect over the optimal assignment
- Our approach consists in slightly pertubating the pattern in a local search manner to affect the capacities of the areas and improve the flow.

# A LNS algorithm

1. Identify the regions of the grid where the cost varies
2. Randomly select a point from those areas and remove M dominos around it
3. Enumerate all possible patterns (LNS step) that can fill the hole :
    - Update incrementally the flow (**sensitivity analysis** of the flow)
    - Store new solution if improvement
4. Return to 2 until a stopping criterion is met

# Selecting the points of interest

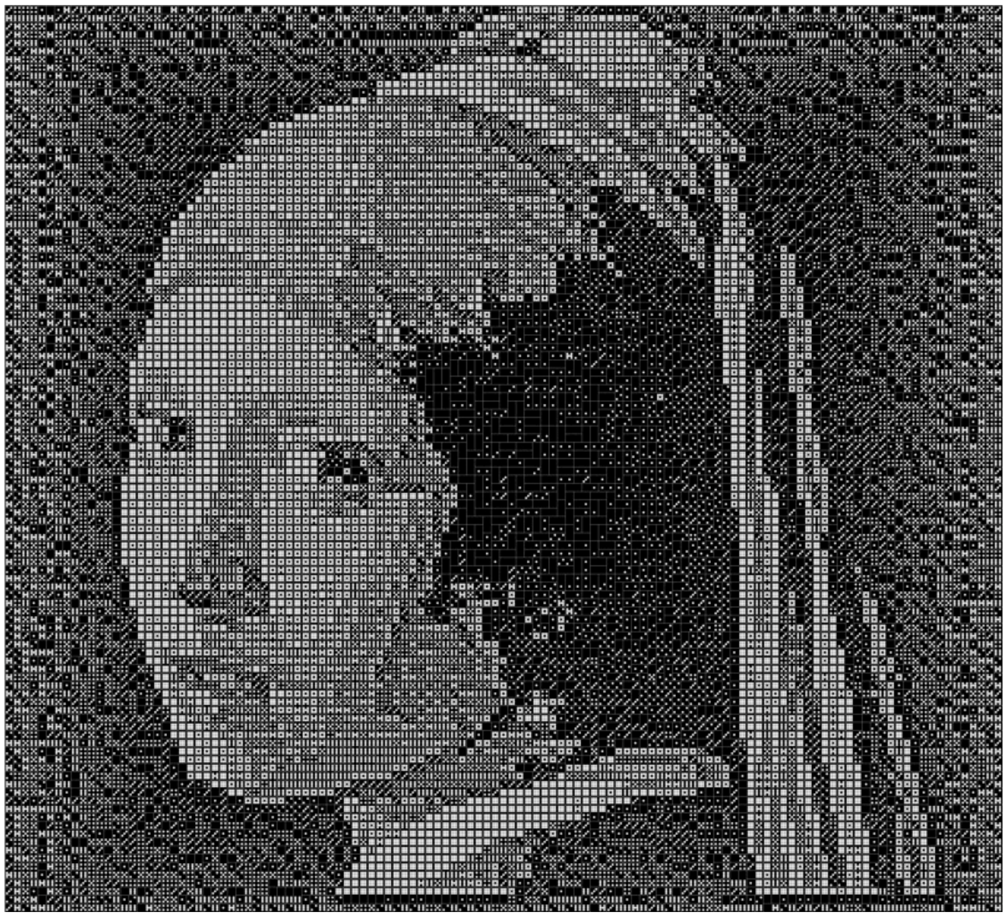
Result of the FAST (Features from Accelerated Segment Test) algorithm on the “Girl with a Pearl Earring”

# Final results

- High quality portraits (gap around 2%)
- Orders of magnitude of speed up (seconds vs hours)

| | ILP | | LNS patterns | | |
|---|---|---|---|---|---|
| **k** | Opt Cost | Time (s) | Cost | Time (s) | Gap (%) |
| 1 | 1192 | 1.04 | 1207 | 8.32 | 1.26 |
| 4 | 4844 | 13.8 | 4903 | 14 | 1.22 |
| 9 | 11255 | 65.9 | 11512 | 14.54 | 2.28 |
| 25 | 33673 | 325.62 | 34498 | 15.72 | 2.45 |
| 49 | 69585 | 7030.29 | 70977 | 17.66 | 2 |
| 121 | 171961 | 9797.55 | 175669 | 27.34 | 2.16 |
| 225 | 376176 | 44895.86 | 380408 | 32.71 | 1.13 |

# Gap between ILP and LNS



Optimal solution (k = 49)
~7030s

LNS solution (k = 49)
~18s

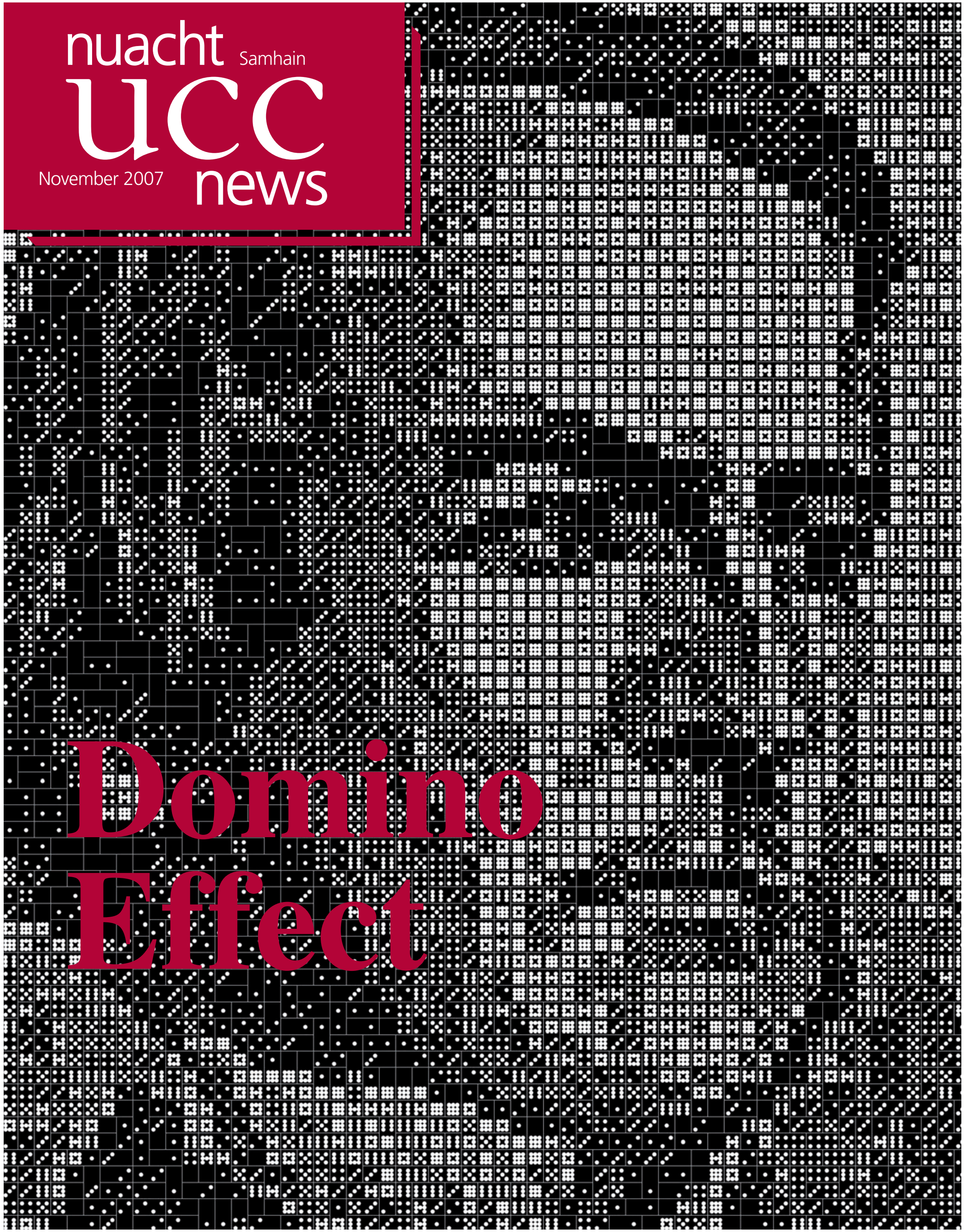
# Applications

- Children love it !


Science discovery event 2007 in Cork

# Applications

- People finally know what you are doing at work



# Applications

- Teaching OR with fun :
  - graph algorithms (Hungarian, Min cost flow and sensitivity analysis)
  - search techniques (depth first search with simple propagation, LNS)
  - algorithm from computer vision (FAST)

# Conclusion

- An efficient and scalable approach based on a reformulation of the problem as a min cost flow problem
- Orders of magnitude of improvements compared to the integer linear approach
- Massive success with kids and great teaching tool
- http://4c.ucc.ie/~hcambaza/


Questions ?

# Applications

- Packing with positioning cost ?