Dinic's Algorithm and Lozenge Tilings: the "Undo" Mechanic

LP

Abstract

We illustrate how Dinic's blocking-flow algorithm finds a perfect matching in a bipartite graph arising from a lozenge tiling problem. The exposition focuses on the augmenting-path "undo" mechanic that uses residual (backward) edges to correct greedy mistakes.

1 The L-shape toy model (bipartite mapping)

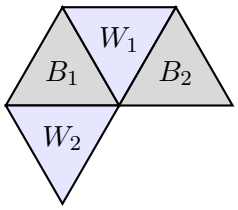
Consider a region of the triangular grid whose faces are colored in checkerboard fashion: every up-pointing triangle is *black* and every down-pointing triangle is *white*. A *lozenge* (rhombus) covers exactly one black triangle and one white triangle that share an edge. A *lozenge tiling* of the region is a partition of all faces into lozenges — equivalently, a perfect matching in the bipartite adjacency graph.

As a running example we take a small "L-shaped" region consisting of four triangles: two black (B_1, B_2) and two white (W_1, W_2). The adjacency structure is:

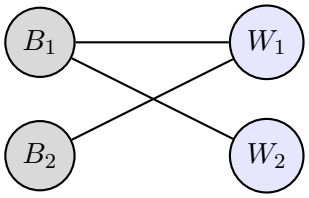
- B_1 touches both W_1 and W_2,
- B_2 is tucked into the corner and touches *only* W_1.

The geometric region and its bipartite graph are shown below.

Geometric region



Bipartite graph



2 The flow network and the greedy mistake (Epoch 1)

To find a perfect matching we embed the bipartite graph in a flow network. Introduce a *source* S with a directed edge of capacity 1 to every black node, and a *sink* T with a directed edge of capacity 1 from every white node. All black-to-white edges also have capacity 1. A maximum flow of value $|B|$ in this network corresponds to a perfect matching.

DFS phase and the greedy mistake

Dinic's algorithm works in *epochs*. Each epoch begins with a BFS from S that assigns *levels* to nodes in the residual graph, then a DFS explores the resulting level graph looking for S-T paths. In Epoch 1 the BFS on the original network gives

$$S = \text{level } 0, \quad B_1, B_2 = \text{level } 1, \quad W_1, W_2 = \text{level } 2, \quad T = \text{level } 3.$$

The DFS proceeds as follows:

1. Start at S. Outgoing edges lead to B_1 and B_2 (both level 1). Suppose B_1 comes first in the adjacency list.
2. From B_1: edges to W_1 and W_2 (level 2). Try W_1 first.
3. From W_1: edge to T (level 3). The sink is reached.
4. Push one unit of flow along $S \to B_1 \to W_1 \to T$, saturating all three edges.
5. Continue from S. Try B_2 next.
6. From B_2: the only edge leads to W_1, but $W_1 \to T$ is now saturated — no residual capacity to reach T. Dead end.
7. Backtrack. No more options from S.

Epoch 1 ends with only one unit of flow — an incomplete matching. (Had B_2 been first in the adjacency list, the DFS would have matched $B_2 \leftrightarrow W_1$ and then $B_1 \leftrightarrow W_2$, finding a perfect matching in a single epoch with no undo needed.)

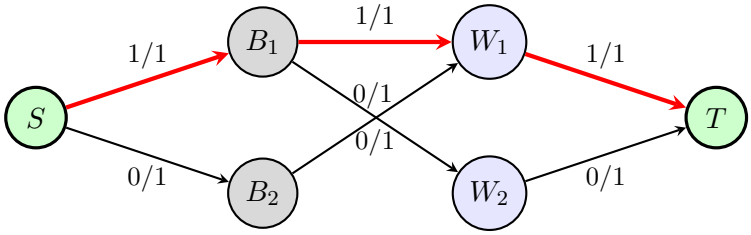


Figure 1: State after Epoch 1. Red edges carry one unit of flow. The greedy path $S \to B_1 \to W_1 \to T$ leaves B_2 stranded. Edge labels show flow/capacity.

3 The residual graph and the "undo" magic (Epoch 2)

Pushing one unit of flow along an edge $u \to v$ reduces its *forward residual capacity* from 1 to 0 and simultaneously creates a *backward residual edge* $v \to u$ of capacity 1. The backward edge encodes the option to *undo* previously committed flow.

In our example, the three saturated edges $S \to B_1$, $B_1 \to W_1$, and $W_1 \to T$ each lose their forward capacity and gain a backward residual edge. Of these three, only $W_1 \to B_1$ matters: it reverses an *original* bipartite edge and encodes the possibility of rerouting the matching. The

other two ($B_1 \to S$ and $T \to W_1$) merely reverse the artificial source/sink plumbing and carry no matching information. The relevant residual edges are:

$$\textit{Forward (unused in Epoch 1):} \quad S \to B_2,\ B_1 \to W_2,\ B_2 \to W_1,\ W_2 \to T$$
$$\textit{Backward (the undo edge):} \quad W_1 \to B_1$$

These are shown in Figure 2 (forward edges in black, the backward edge $W_1 \to B_1$ dashed in orange).

Epoch 2 begins with a fresh BFS from S on this residual graph:

1. Level 0: S.
2. Level 1: B_2 (via $S \to B_2$). Note that B_1 is *not* reachable from S directly — the edge $S \to B_1$ is saturated.
3. Level 2: W_1 (via $B_2 \to W_1$).
4. Level 3: B_1 (via the backward edge $W_1 \to B_1$). This is the crucial step: B_1 re-enters the level graph at a deeper level, reached through the undo edge.
5. Level 4: W_2 (via $B_1 \to W_2$).
6. Level 5: T (via $W_2 \to T$).

The shortest S-T path in the residual graph now has length 5, compared to length 3 in Epoch 1 — a strictly longer layered path, as Dinic's algorithm guarantees.

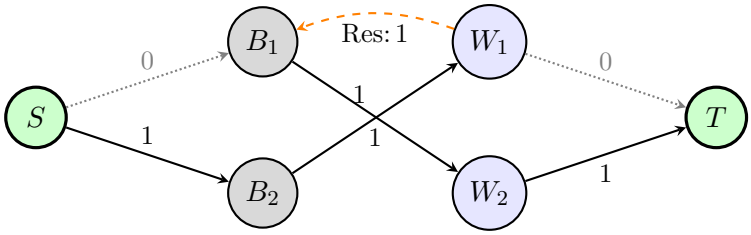


Figure 2: Residual graph before the Epoch 2 DFS. The dashed orange edge $W_1 \to B_1$ is the backward residual created by the Epoch 1 flow. Dotted gray edges have zero residual capacity. Labels show residual capacity.

4 The augmenting path resolution

The DFS of Epoch 2 walks the level graph built by the BFS above:

1. Start at S (level 0). The only level-1 neighbor is B_2.
2. From B_2 (level 1): edge to W_1 (level 2).
3. From W_1 (level 2): the forward edge $W_1 \to T$ is saturated, but the *backward* edge $W_1 \to B_1$ (level 3) has residual capacity 1. Take it.

4. From B_1 (level 3): edge to W_2 (level 4).

5. From W_2 (level 4): edge to T (level 5). Sink reached.

The augmenting path is

$$S \;\to\; B_2 \;\to\; W_1 \;\xrightarrow{\text{backward}}\; B_1 \;\to\; W_2 \;\to\; T.$$

What pushing one unit of flow does

We now trace the effect of sending one unit of flow along this path. There are two kinds of edges on the path — forward edges (unused until now) and one backward edge — and they behave differently.

Forward edges increase their flow from 0 to 1:

$S \to B_2$	flow: $0 \to 1$	S now supplies B_2
$B_2 \to W_1$	flow: $0 \to 1$	B_2 claims W_1
$B_1 \to W_2$	flow: $0 \to 1$	B_1 claims W_2 (its alternate neighbor)
$W_2 \to T$	flow: $0 \to 1$	W_2 drains to the sink

The backward edge $W_1 \to B_1$ does not add new flow; it *cancels* one unit of Epoch 1 flow on the original forward edge $B_1 \to W_1$:

$$\text{net flow on } B_1 \to W_1 : \quad 1 - 1 = 0.$$

In matching terms: B_1 "undoes" its Epoch 1 match with W_1, freeing W_1 to be claimed by B_2 instead.

Summary of all edge flows after both epochs:

Edge	Net flow	
$S \to B_1$	1	(from Epoch 1, unchanged)
$S \to B_2$	1	(new in Epoch 2)
$B_1 \to W_1$	0	(was 1, cancelled by backward edge)
$B_1 \to W_2$	1	(new in Epoch 2)
$B_2 \to W_1$	1	(new in Epoch 2)
$W_1 \to T$	1	(from Epoch 1, unchanged)
$W_2 \to T$	1	(new in Epoch 2)
Total $S \to T$	**2**	

The edges carrying unit flow form exactly two S-to-T paths: $S \to B_1 \to W_2 \to T$ and $S \to B_2 \to W_1 \to T$.

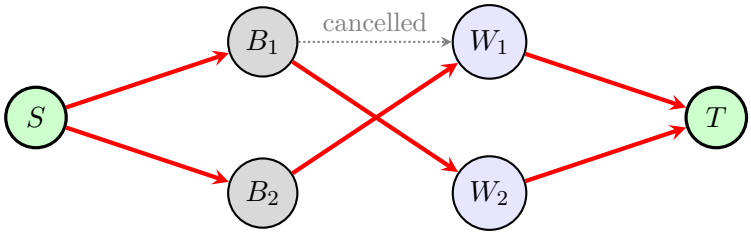


Figure 3: Final flow after both epochs. Red edges carry one unit of flow, forming two disjoint S-T paths. The dotted gray edge $B_1 \to W_1$ is the Epoch 1 assignment that was cancelled by the backward edge in Epoch 2.

After augmentation the total flow is 2, equaling $|B|$, so the algorithm terminates. Reading off the edges with unit flow gives the **perfect matching**:

$$\boxed{B_1 \leftrightarrow W_2, \qquad B_2 \leftrightarrow W_1.}$$

Every black triangle is paired with a white triangle, and every triangle is covered exactly once — a valid lozenge tiling of the L-shaped region.

Remark 4.1. The backward-edge mechanism is not specific to this toy example. In a large triangular region with hundreds of triangles, the DFS may need to "undo" long chains of tentative matches before settling on a global perfect matching.

5 Correctness and termination

Correctness follows from the max-flow min-cut theorem: a flow f is maximum if and only if the residual graph G_f contains no S-T path. The algorithm terminates because it cannot cycle:

Proposition 5.1 (Monotonicity of shortest-path distance)**.** *Let d_k denote the length of the shortest S-T path in the residual graph at the start of epoch k. Then $d_{k+1} > d_k$.*

Proof. In epoch k, flow is sent only along edges from level ℓ to level $\ell + 1$ in the BFS layering. Any new residual edge created goes from level $\ell + 1$ back to level ℓ — a "wrong-direction" edge. Using one in the next BFS costs two levels, so $d_{k+1} \geq d_k + 1$. □

Since d_k is a positive integer bounded by $|V|$, the number of epochs is at most $|V| - 1$. When the BFS of some epoch fails to reach T, no augmenting path exists and the current flow is maximum.

Remark 5.2. On a unit-capacity bipartite network with n nodes on each side and m edges, at most $O(\sqrt{n})$ epochs are needed, each taking $O(m)$ time — giving total running time $O(m\sqrt{n})$. This is the Hopcroft–Karp bound [HK73].

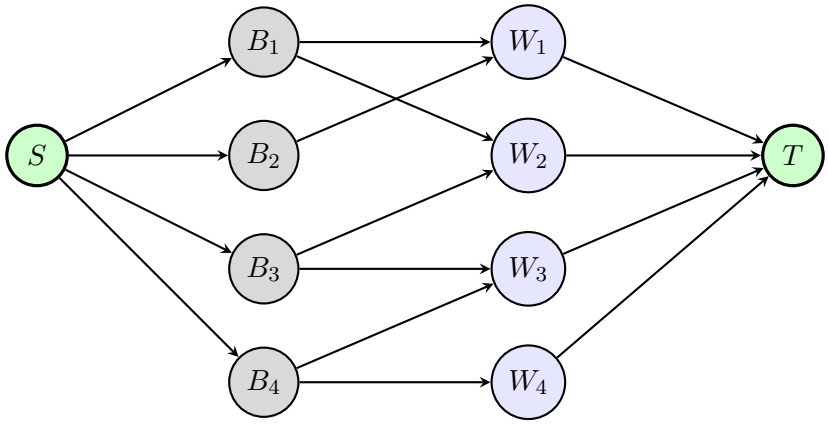
6 A larger example: the cascade of undos

Consider a 4×4 bipartite graph with the following adjacency (a chain, not a cycle):

$$B_1 \sim \{W_1, W_2\}, \quad B_2 \sim \{W_1\}, \quad B_3 \sim \{W_2, W_3\}, \quad B_4 \sim \{W_3, W_4\}.$$

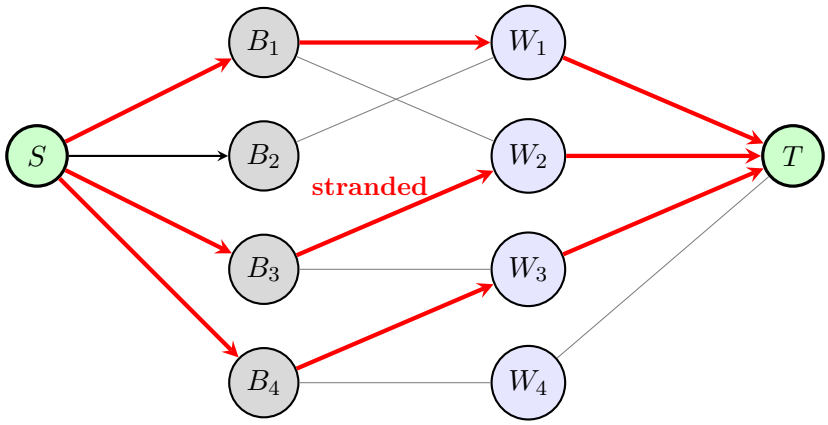
The unique perfect matching is B_2-W_1, B_1-W_2, B_3-W_3, B_4-W_4, but the greedy DFS will not find it in one epoch.

Initial flow network



Epoch 1. The DFS from S tries each B_i in order. B_1 grabs W_1 (its first neighbor). B_2 tries W_1 — saturated, dead end. B_3 grabs W_2. B_4 grabs W_3. Flow $= 3$; B_2 is stranded.

After Epoch 1: flow $= 3$, B_2 stranded



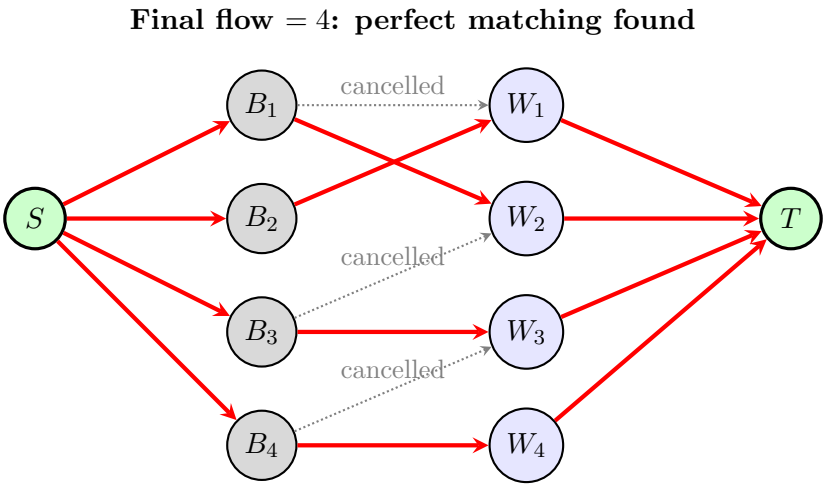
Epoch 2 — the cascade. The BFS on the residual graph assigns levels:

$$S(0) \to B_2(1) \to W_1(2) \xrightarrow{\text{back}} B_1(3) \to W_2(4) \xrightarrow{\text{back}} B_3(5) \to W_3(6) \xrightarrow{\text{back}} B_4(7) \to W_4(8) \to T(9).$$

The DFS finds the augmenting path of length 9 with *three* backward edges — a cascade of three undos:

- B_2 claims W_1, forcing B_1 to undo its match with W_1.

- B_1 claims W_2, forcing B_3 to undo its match with W_2.
- B_3 claims W_3, forcing B_4 to undo its match with W_3.
- B_4 claims W_4 (free) and reaches T.



The final matching is $\boxed{B_1\text{-}W_2,\ B_2\text{-}W_1,\ B_3\text{-}W_3,\ B_4\text{-}W_4.}$ One stranded node triggered a chain reaction that rearranged all four previous assignments.

References

[HK73] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. 5