# SOME GRAPH EXPLORATION

## 1. Basics

Let $G = (V, E)$ be a graph with $V = \{1, \dots, n\}$. That is, $E \subseteq V \times V$ is such that if $(i, j) \in E$ then $(j, i) \in E$. We will assume $G$ has no **self-loops**, i.e., edges of the form $(i, i)$. We call $j$ a **neighbor** of $i$ if $(i, j) \in E$. The **adjacency matrix** of $G$ is the matrix $A = A_G \in M_n(\mathbb{N})$ given by

$$A_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{else} \end{cases}.$$

**Proposition 1.** *Define $\mathbb{1}$ to be the $n \times 1$ matrix (i.e., column vector) with 1 in every slot; define $D = D_G \coloneqq A\mathbb{1}$. Then $D_{i,i}$ is the number of neighbors of $i$.*

*Proof.* Exercise. □

**Definition 1** (Laplacian)**.** *We define the* **graph Laplacian** *of $G$ to be the matrix $L = L_G \coloneqq D - A$.*

The following proposition summarizes the entries of $L$.

**Proposition 2.** $L_{i,j} = \begin{cases} D_{i,i} & i = j \\ -1 & (i,j) \in E \\ 0 & \textit{else} \end{cases}$

*Proof.* Exercise. □

Since we defined the set $V$ of nodes to be the set $\{1, \dots, n\}$, we may consider a real-valued vector $v \in \mathbb{R}^n$ to be an assignment of real numbers to the nodes of $G$; i.e., a vector $v$ may be seen as a function on $G$.

This begs the question of the action of $L$ on such a vector (i.e., function) $v \in \mathbb{R}^n$. The answer is given by the following lemma.

**Lemma 1.** *Let $v \in \mathbb{R}^n$, and define $w \coloneqq Lv$. Then*

$$w_i = \sum_{\substack{j: \\ (i,j) \in E}} (v_i - v_j).$$

*Proof.* Exercise. □

So for each $i$, $w_i$ measures the net difference between $v_i$ and its neighbors. We turn this local information into global measure of "how constant $v$ is" by multiplying $L$ by $v$ on both sides.

**Proposition 3.** *For $v \in \mathbb{R}^n$, the quantity $v'Lv$ is the sum of squares of differences between neighboring nodes' $v$-values.*[1] *That is,*

$$v'Lv = \sum_{\substack{(i,j) \in E \\ i<j}} (v_i - v_j)^2.$$

*Proof.* Exercise.
Hint: to get at the $i < j$ condition, remember that you only want to count each edge **once**! □

## 2. Search

I have at least two questions:

(1) How does the fact that nodes have 2, 3, or 4 neighbors manifest in the adjacency matrix?
(2) Can we exploit this structure to parallelize exploration?

[1] Here $v'$ denotes the usual transpose of $v$.

2.1. **Structure of the graph.** The first goal here is to write down the adjacency matrix for the 100 or so nodes surrounding the $3\times 3$ solved state. This is to start addressing question (1) above. See `explore_puzzle_space.py` for progress.

We'll be considering, e.g., a $3 \times 3$ puzzle configuration as a permutation of the set

$$\{1, 2, 3, 4, 5, 6, 7, 8, 0\}.$$

There are some particularities about where 0 goes in the null/solved permutation, but that shouldn't be too important now. Permutations will be denoted by $\sigma$.

Let $\sigma_0, \ldots, \sigma_{99}$ be the configurations of the 100 nodes surrounding the solved state. Let $\lambda(\sigma)$ denote the Lehmer encoding of the permutation $\sigma$. So $\lambda(\sigma) \in \{0, \ldots, 99\}$. Let $i_0, \ldots, i_{99}$ be such that $\lambda(\sigma_{i_0}) < \cdots < \lambda(\sigma_{i_{99}})$, and define $f$ by $f(\lambda(\sigma_{i_k})) \coloneqq k$.

Now, whenever a new node $n$ is discovered from parent $p$, keep track of number $\lambda(n.\sigma)$ as well as the pair

$$(\lambda(n.\sigma), \lambda(p.\sigma)).$$

The size of the set $\{n.\sigma\}$ will tell us how big to make the adjacency matrix, and the pairs $(\lambda(n.\sigma), \lambda(p.\sigma))$ will tell us which entries are nonzero.

## 3. Results

Not a true "results" section, but rather just explaining what we see. The file `spec_emb_2x2_-sqrt3-sqrt3.pdf` contains a spectral embedding of the state space of the $2 \times 2$ sliding puzzle along the two eigenvectors with eigenvalue $-\sqrt{3}$. Not very good. Also, because of the overcounting used, there's an extra point at the origin with no neighbors. That, and the crazy look of the graph are not good. This was done in Mathematica just to get something done; will convert to Python later.

## Meeting notes

**1/13/2026.** We agreed the following tasks should be done: (not ranked)

- Change the `PuzzleState` class around, maybe splitting it into two different classes. E.g., `Puzzle` and `State`
- Rewrite `explore_puzzle_space.py` to match `w_a-star_variants.py` with use of classes, arg parsing, etc.
- Add some information about search algorithms to the writeup. For instance, why does A* need a `.d` attribute?

*Classes.* The following class structures might work

| `Puzzle` Att/method | Functionality |
|---|---|
| `.current_state` | A `State` class: now |
| `.sol_state` | A `State` class: goal |
| `.is_solved()` | Checks `State` class |
| `.d` | Measures distance from parent |

| `State` Att/method | Functionality |
|---|---|
| `.config` | A 2-d array of integers |
| `.linear()` | Returns a copied 1-d array of integers |