

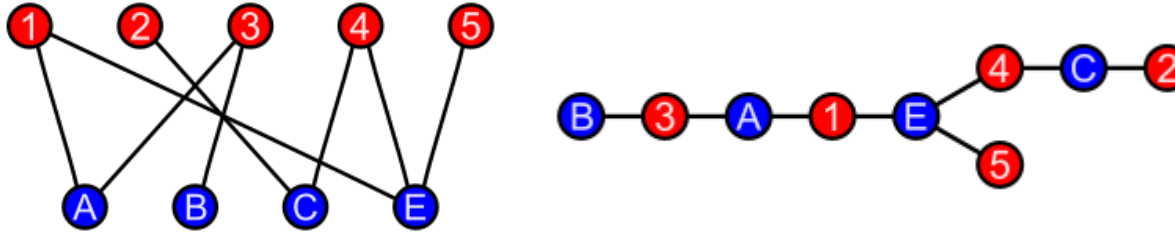


Maximum Bipartite Matching

Ruan Schoeman | Training Camp 2 | 14-15 Jan 2023



Definitions



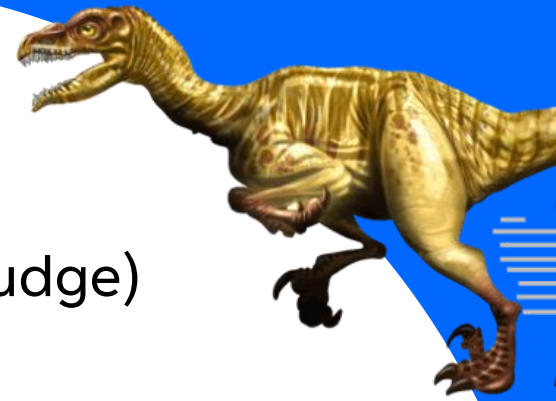
A **bipartite graph** is a graph whose vertices can be divided into two disjoint sets such that no two vertices in a set shares an edge.

A **matching** is a set of pairwise non-adjacent edges. The **cardinality** of a matching is the number of edges in the matching. All the vertices that are adjacent to an edge in the matching are called **saturated** by the matching.

A **maximal matching** is a matching such that its cardinality is maximal among all possible matchings for a given graph.

Sample Problem

(Gopher II on Kattis Online Judge)



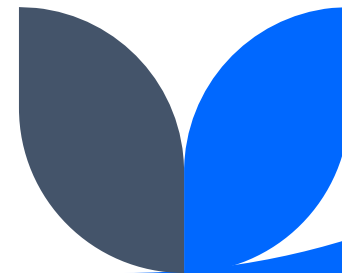
There are n gophers and m gopher holes, each at distinct (x, y) coordinates.

A hawk raptor arrives and if a gopher does not reach a hole in s seconds it is vulnerable to being eaten.

A hole can save at most one gopher.



All the gophers run at the same velocity v .

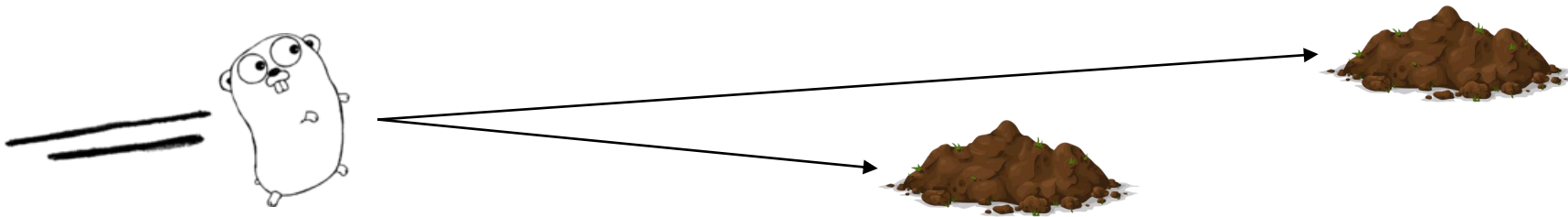
The gopher family needs an escape strategy that minimizes the number of vulnerable gophers.



Possible solution

We can consider a bipartite graph where:

- The vertices in the first set represent the gophers 
- The vertices in the second set represent the gopher holes 
- The edges represent which holes each gopher can reach in s seconds



A maximum matching selects an escape strategy where the largest number of gophers escape.

Slow Algorithm

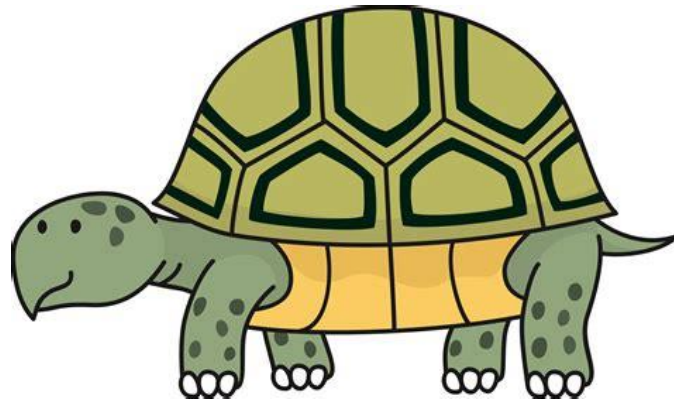
Brute Force

One way of finding the maximum matching is through brute force:

Generate every possible matching and select the largest one.

This has (very roughly) a time complexity of $\sim O(E!) > O(2^{E^x})$ where E is the number of edges, by my best estimation.

This is obviously way too slow for most problems.



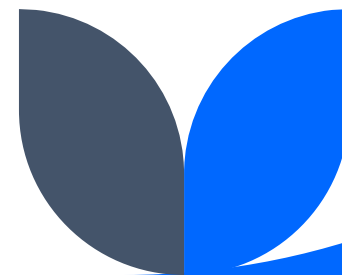
Proposed Algorithm

Kuhn's Algorithm - Definitions

A **path** of length k is a simple path containing k edges.

An **alternating path** (in respect to some matching) is a path whose edges alternately belong / do not belong to a matching.

An **augmenting path** (in respect to some matching) is an alternating path whose initial and final vertices are unsaturated, i.e., do not belong to the matching.



Proposed Algorithm

Kuhn's Algorithm – Background: Berge's lemma

A matching M is maximum \Leftrightarrow there is no augmenting path relative to the matching M .

Proven by French mathematician Claude Berge in 1957, although observed by Danish and Hungarian mathematicians Julius Peterson and Denés Kőnig in 1891 and 1931.

The proof can be found here:

https://cp-algorithms.com/graph/kuhn_maximum_bipartite_matching.html#proof



Proposed Algorithm

Kuhn's Algorithm – Algorithm

1. Start with an empty matching
2. While you can find an augmenting path, update the matching by alternating along this path
3. Repeat until you can no longer find an augmenting path
4. You now have a maximum matching

Proposed Algorithm

Kuhn's Algorithm – Finding augmenting paths

A simple way of finding augmenting paths is to look for an augmenting path starting at each vertex using breadth-first or depth-first traversal.

The algorithm is more convenient to describe if it is assumed that the graph is already split in two, but can be implemented in such a way that the input graph is not explicitly split in two.



Proposed Algorithm

Kuhn's Algorithm – Finding augmenting paths

First, look at all the vertices v in the first part of the graph, where $v = 1..n_1$.

If v is already saturated with respect to the current matching, skip it.

Otherwise, try to saturate it by looking for an augmenting path starting at said vertex.

Proposed Algorithm

Kuhn's Algorithm – Finding augmenting paths

To find an augmenting path starting at vertex v , you carry out a special breadth-first or depth-first (usually depth-first) traversal.

Initially, you start at v .

Look at all the edges (v, to) .

If to is not already saturated, then you've found an augmenting path, and you can simply add (v, to) to your matching.

Otherwise, if to is already saturated with the edge (to, p) , go along the edge, and now look for an augmenting path starting (v, to) , (to, p) .

To do this, simply continue your traversal from p , instead of v .

Now remove all of the even-indexed edges in your path to your matching and add the odd-indexed edges.



Proposed Algorithm

Kuhn's Algorithm – Finding augmenting paths

The traversal will either find an augmenting path and saturate v , or it will not find such a path, and, therefore, v can not be saturated.

After all the vertices $v = 1..n_1$ have been scanned, the current matching will be maximum.



Proposed Algorithm

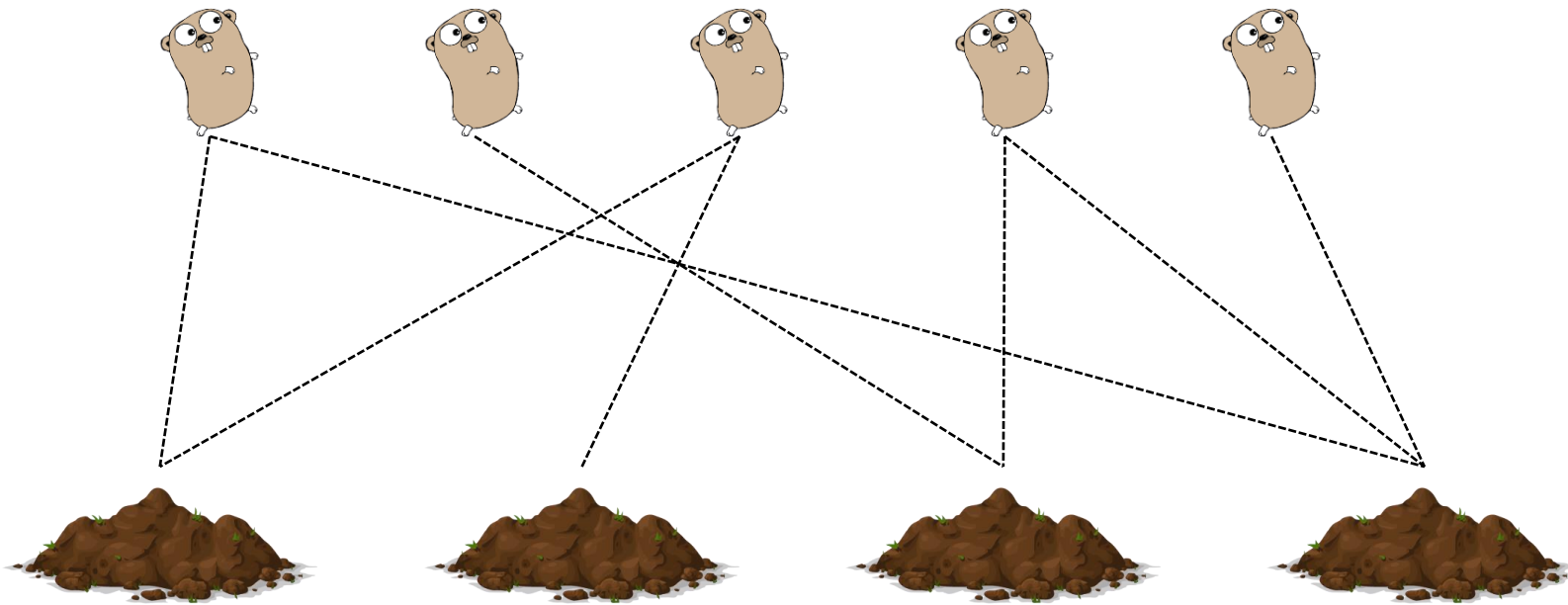
Kuhn's Algorithm – Time complexity

There is at most n traversals of the graph, where n is the number of vertices, so the time complexity is $O(nm)$ where m is the number of edges, so the time complexity is at most $O(n^3)$.

If, however, traversals only start from the first part of the graph time complexity is $O(n_1m)$, which is $O(n_1^2n_2)$ at worst. This shows that it is faster if the first part contains less vertices than the second.

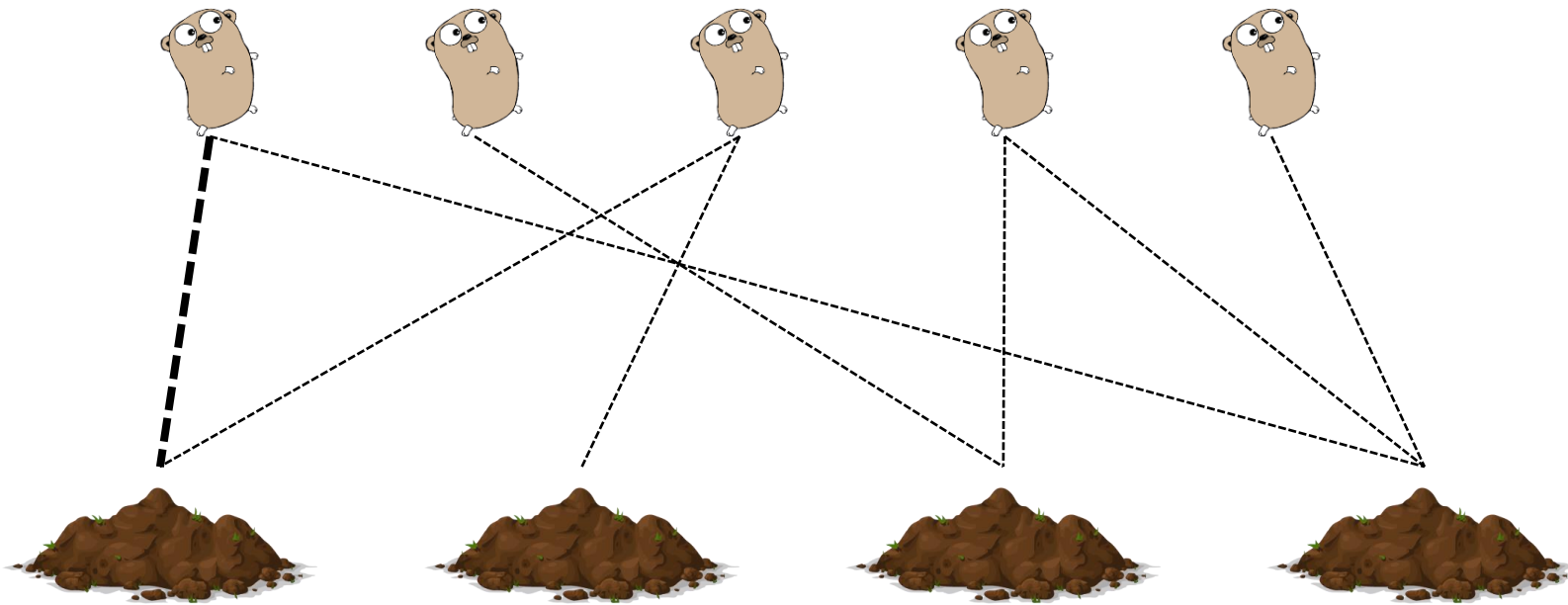
Proposed Algorithm

Kuhn's Algorithm – Example



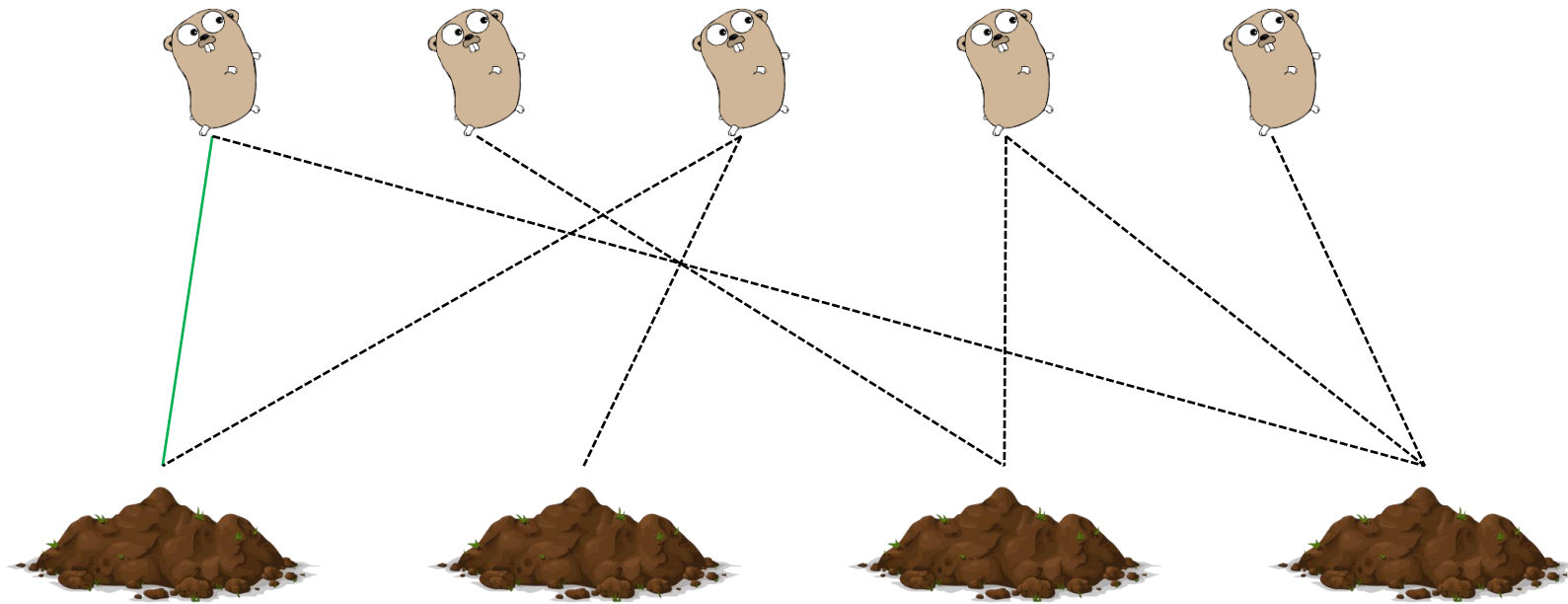
Proposed Algorithm

Kuhn's Algorithm – Example



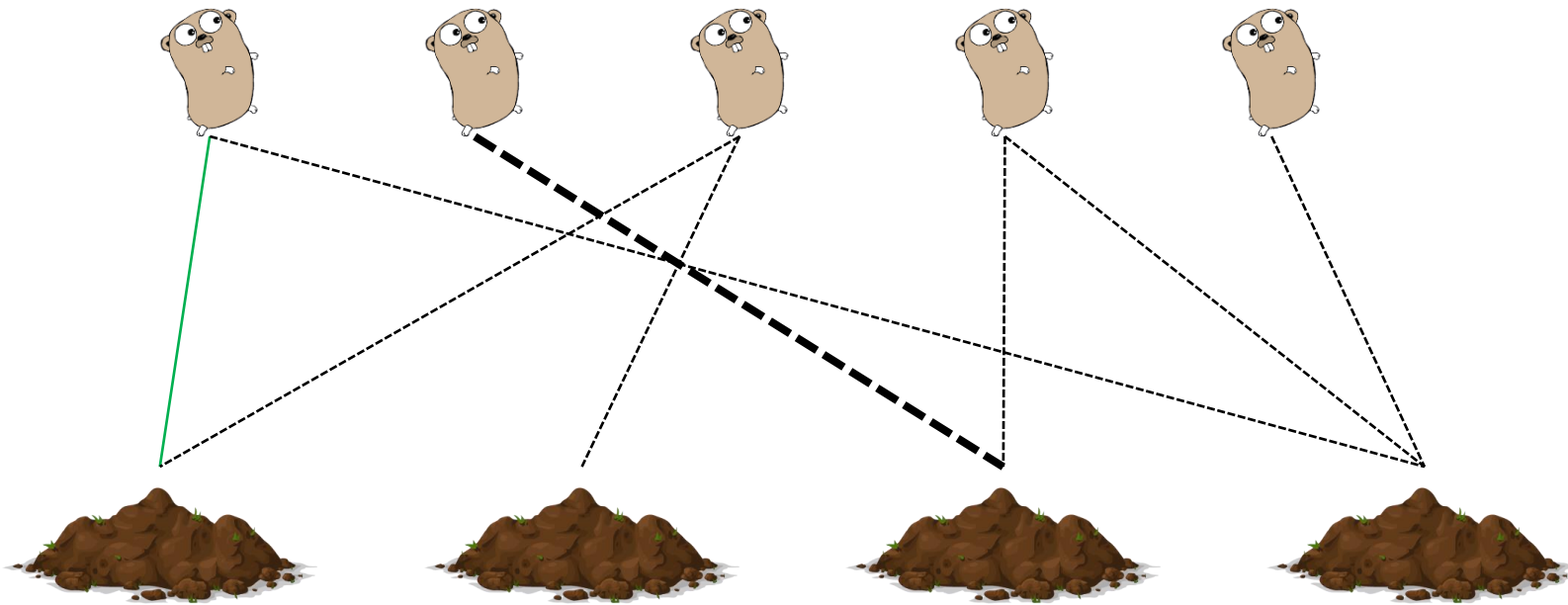
Proposed Algorithm

Kuhn's Algorithm – Example



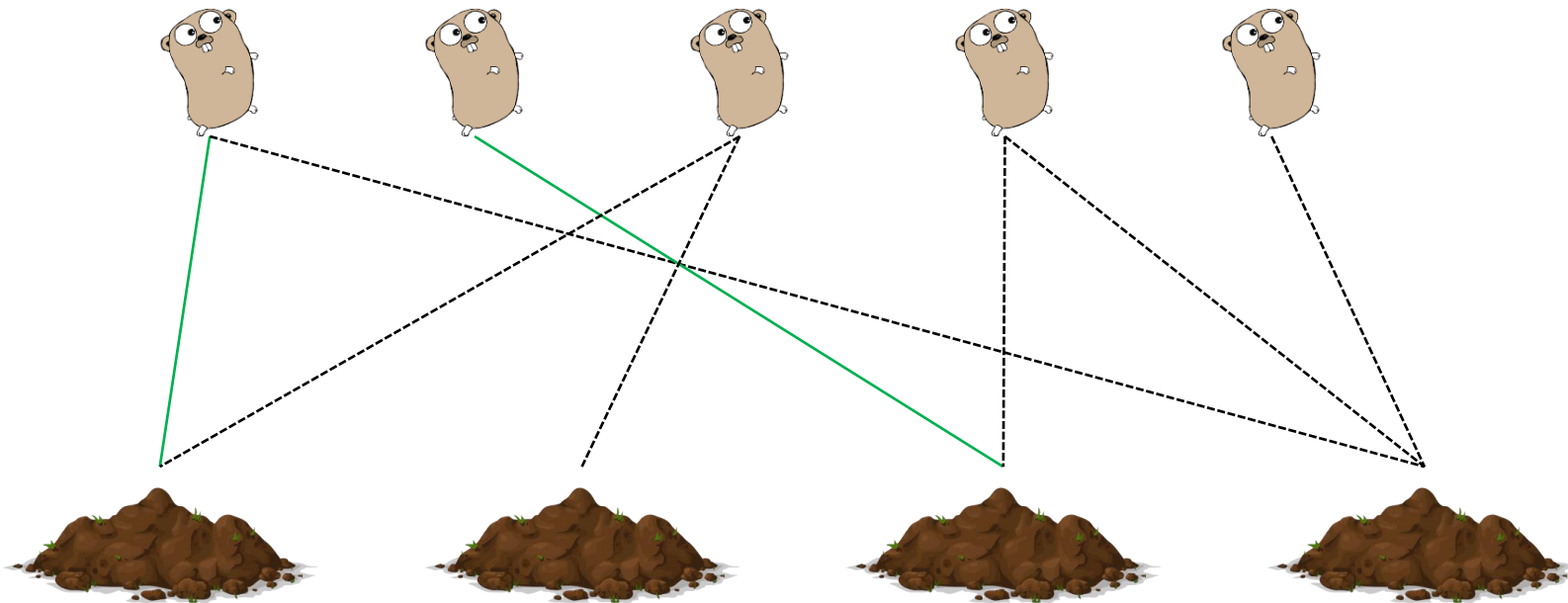
Proposed Algorithm

Kuhn's Algorithm – Example



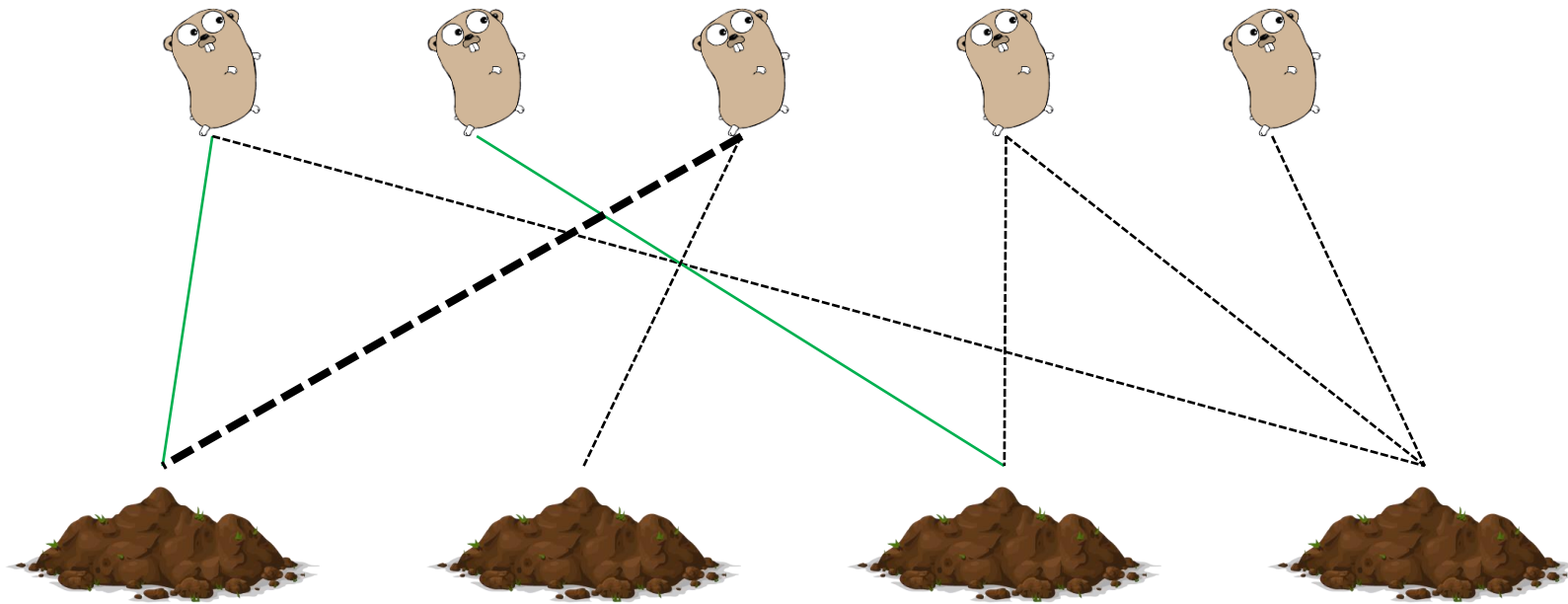
Proposed Algorithm

Kuhn's Algorithm – Example



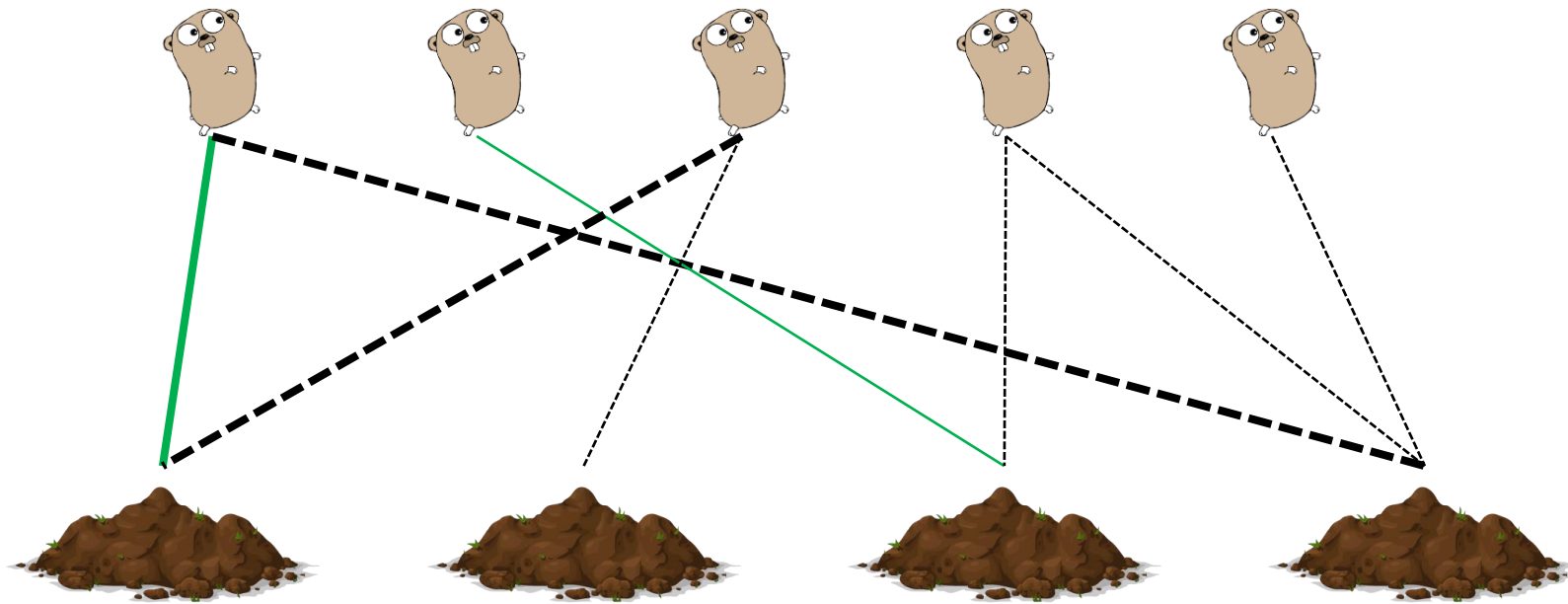
Proposed Algorithm

Kuhn's Algorithm – Example



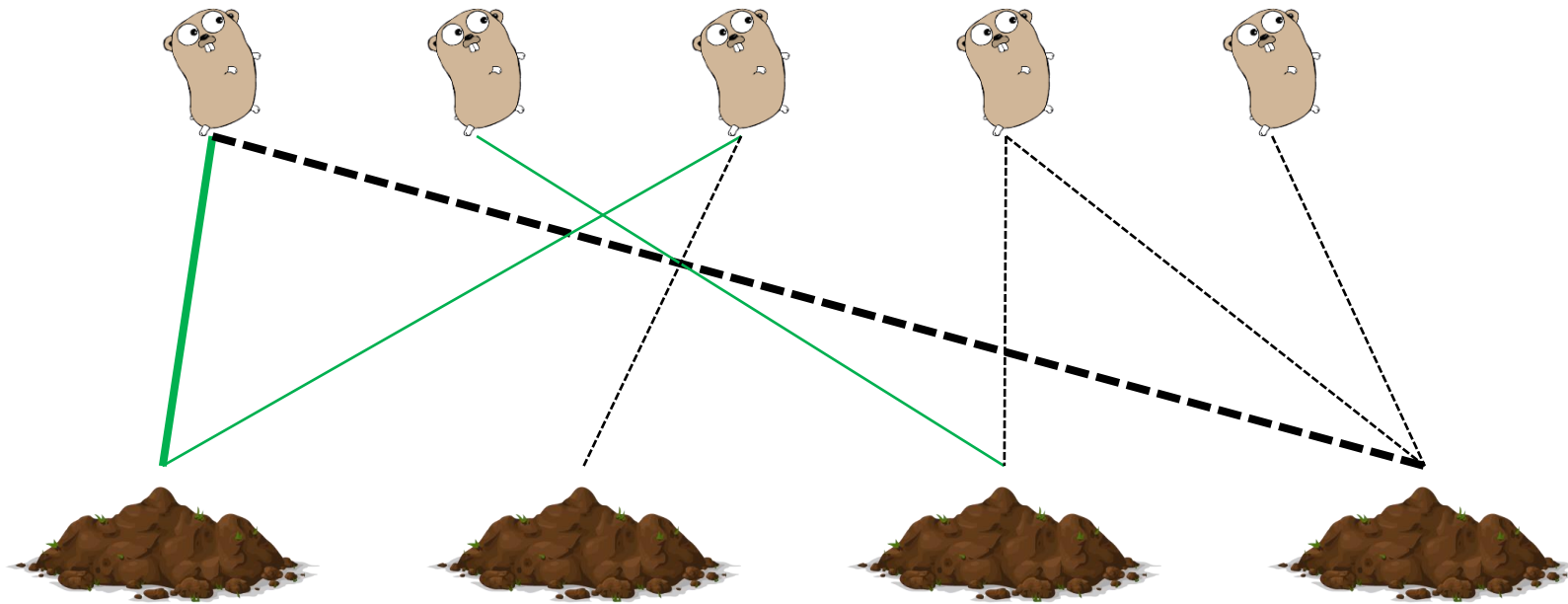
Proposed Algorithm

Kuhn's Algorithm – Example



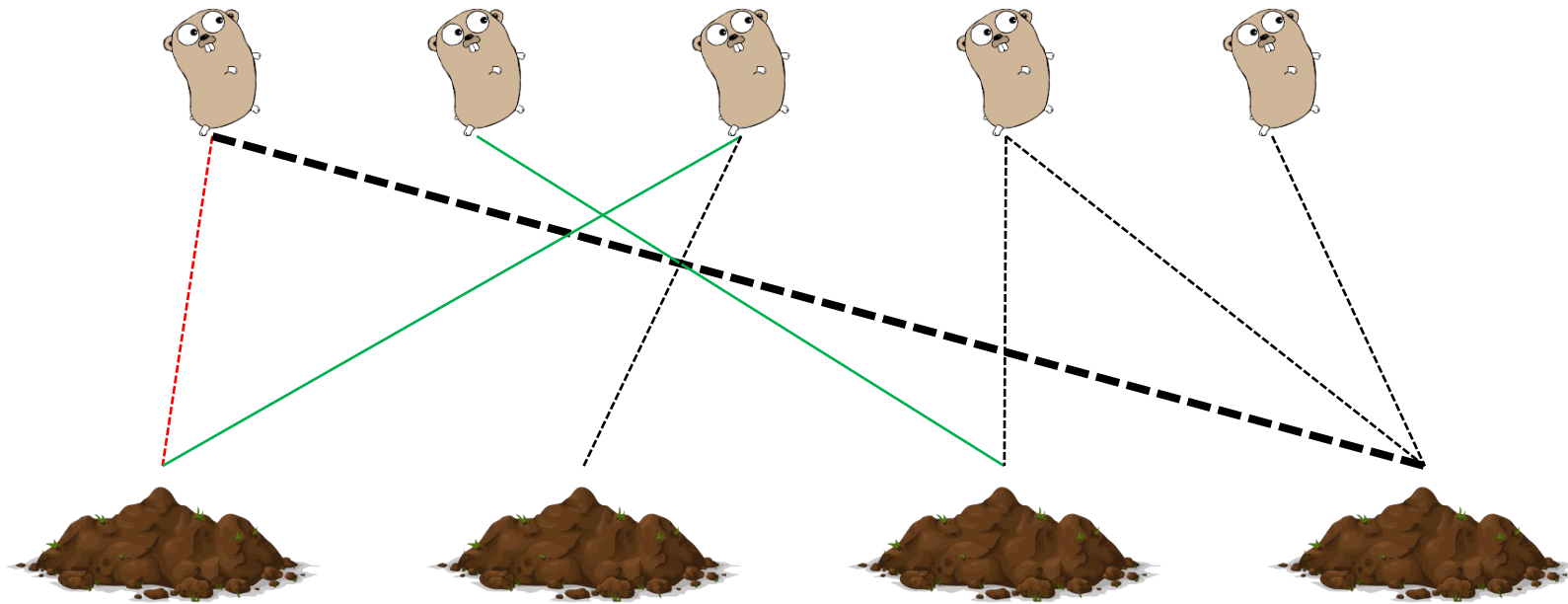
Proposed Algorithm

Kuhn's Algorithm – Example



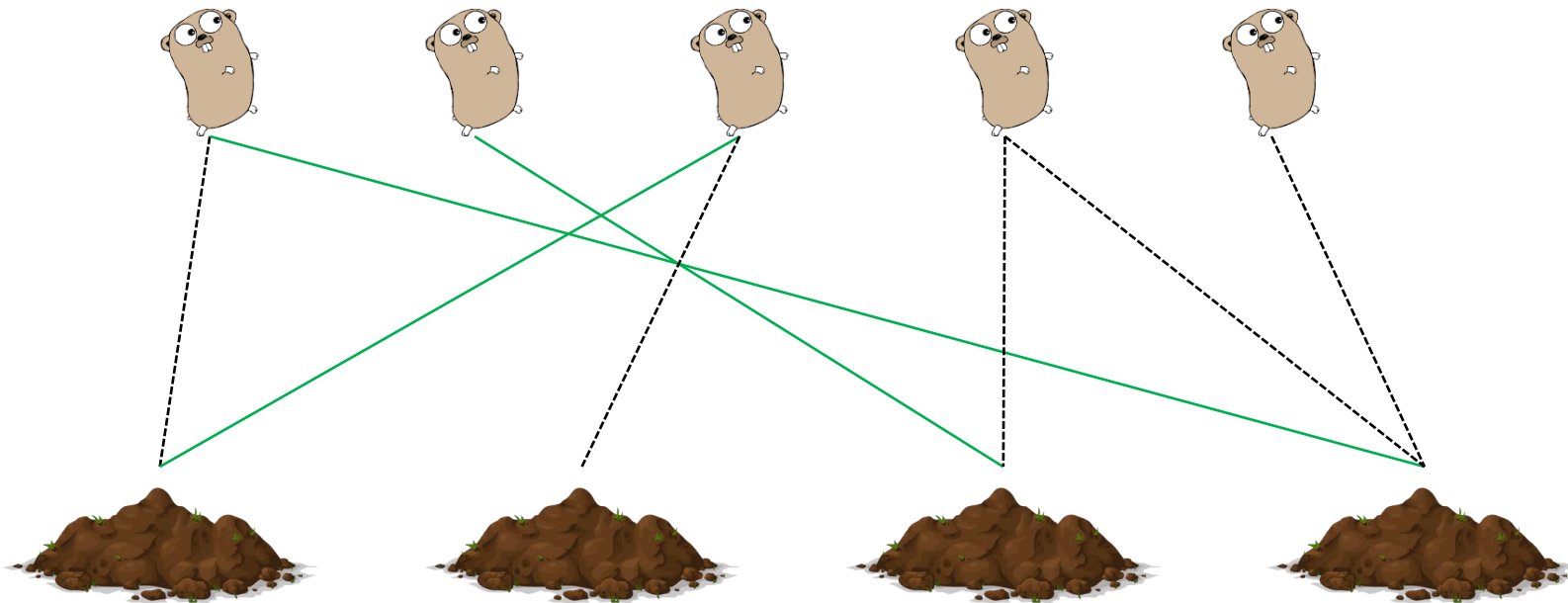
Proposed Algorithm

Kuhn's Algorithm – Example



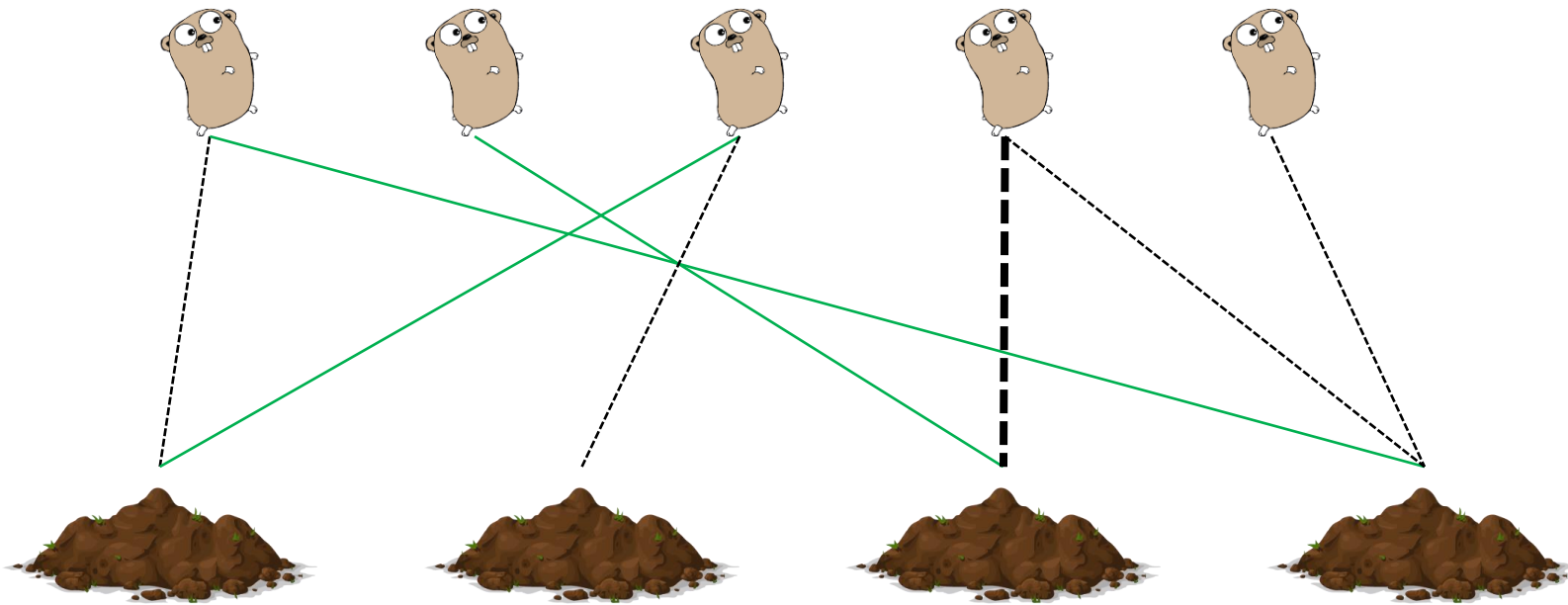
Proposed Algorithm

Kuhn's Algorithm – Example



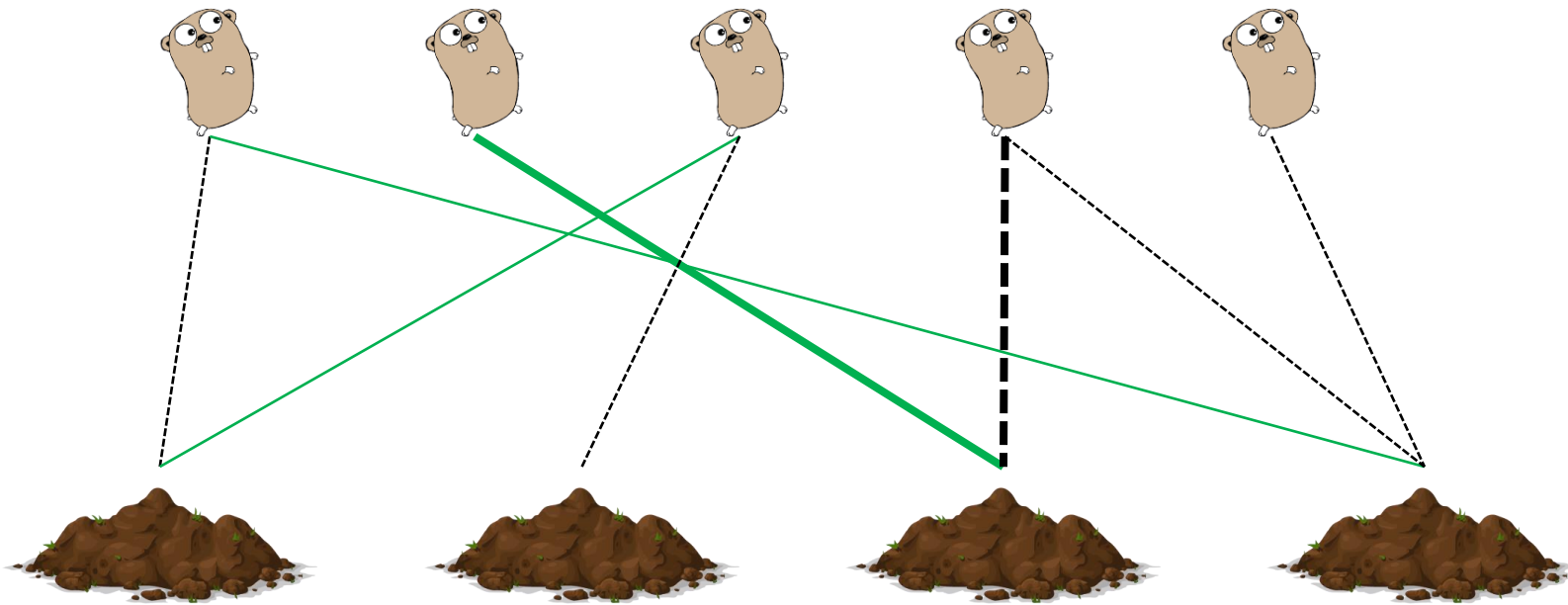
Proposed Algorithm

Kuhn's Algorithm – Example



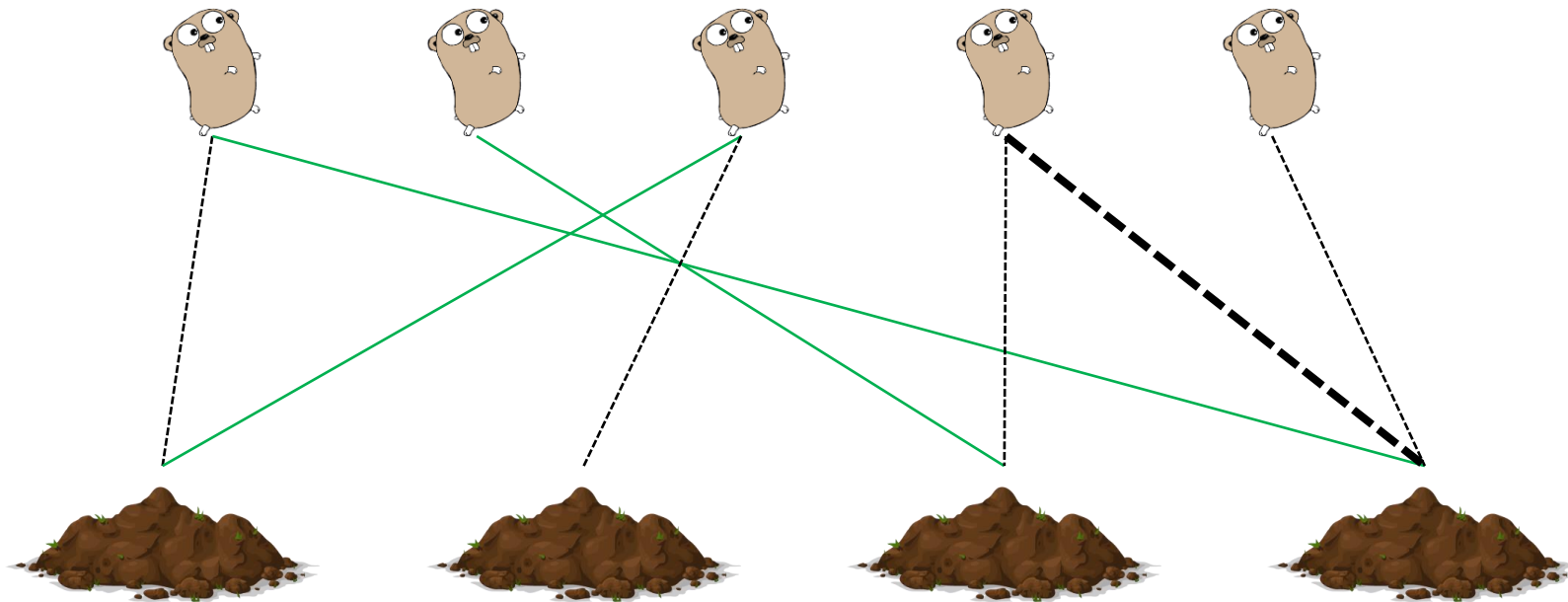
Proposed Algorithm

Kuhn's Algorithm – Example



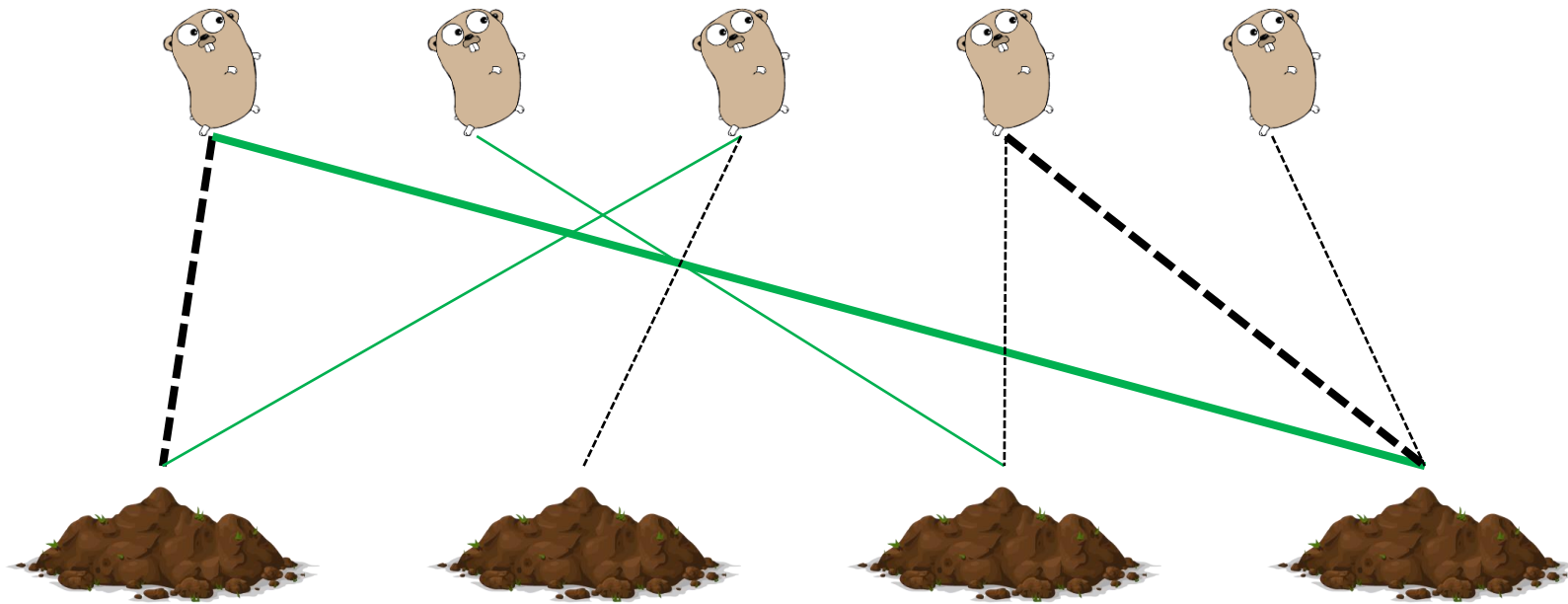
Proposed Algorithm

Kuhn's Algorithm – Example



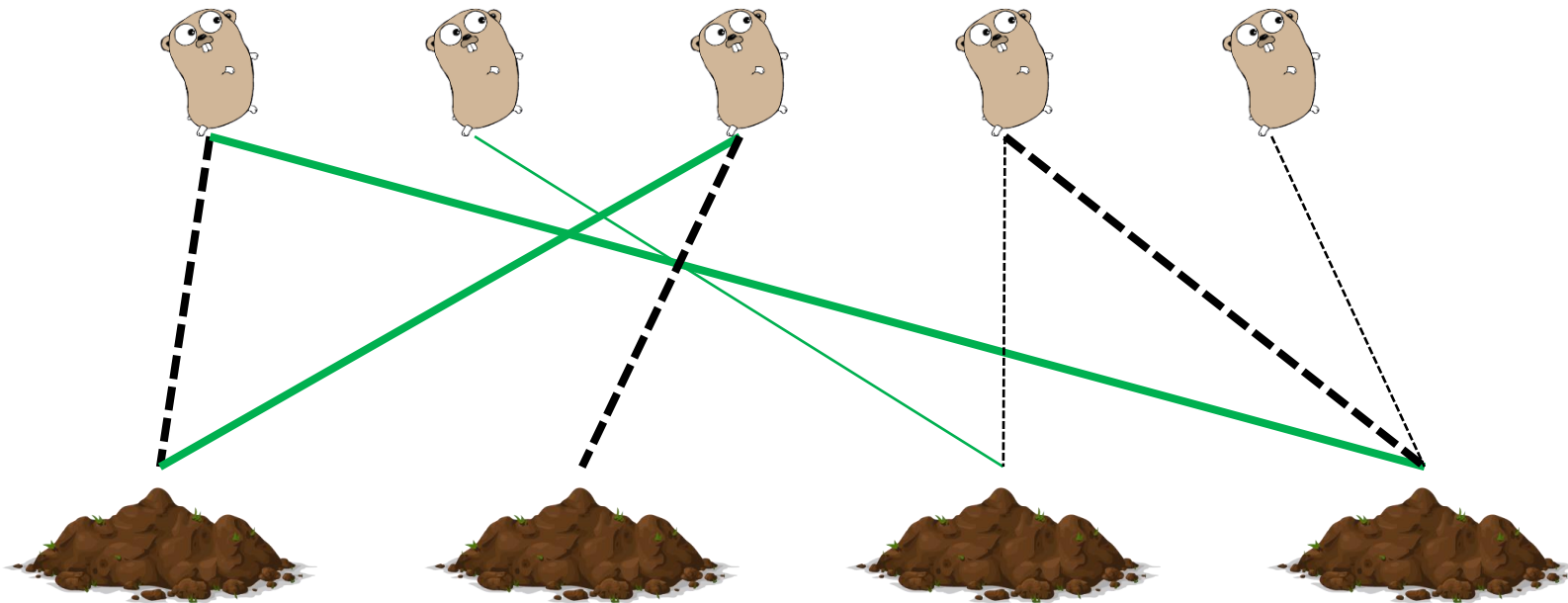
Proposed Algorithm

Kuhn's Algorithm – Example



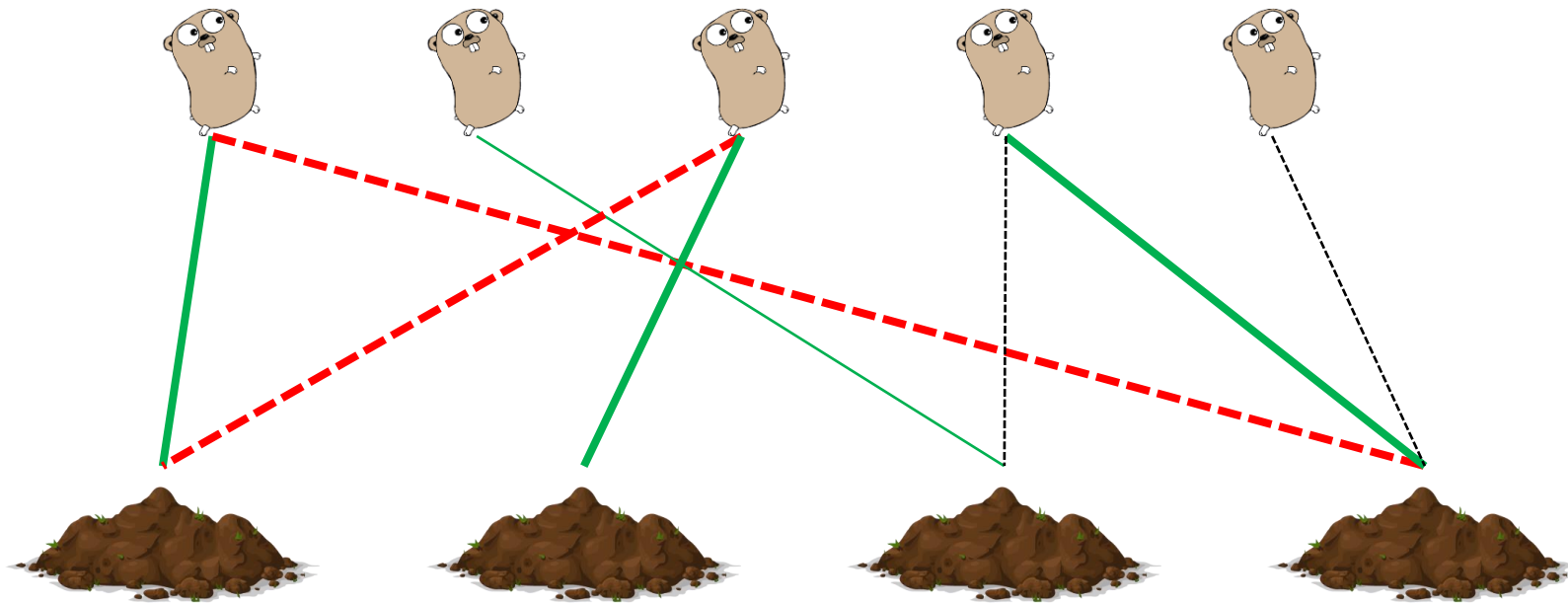
Proposed Algorithm

Kuhn's Algorithm – Example



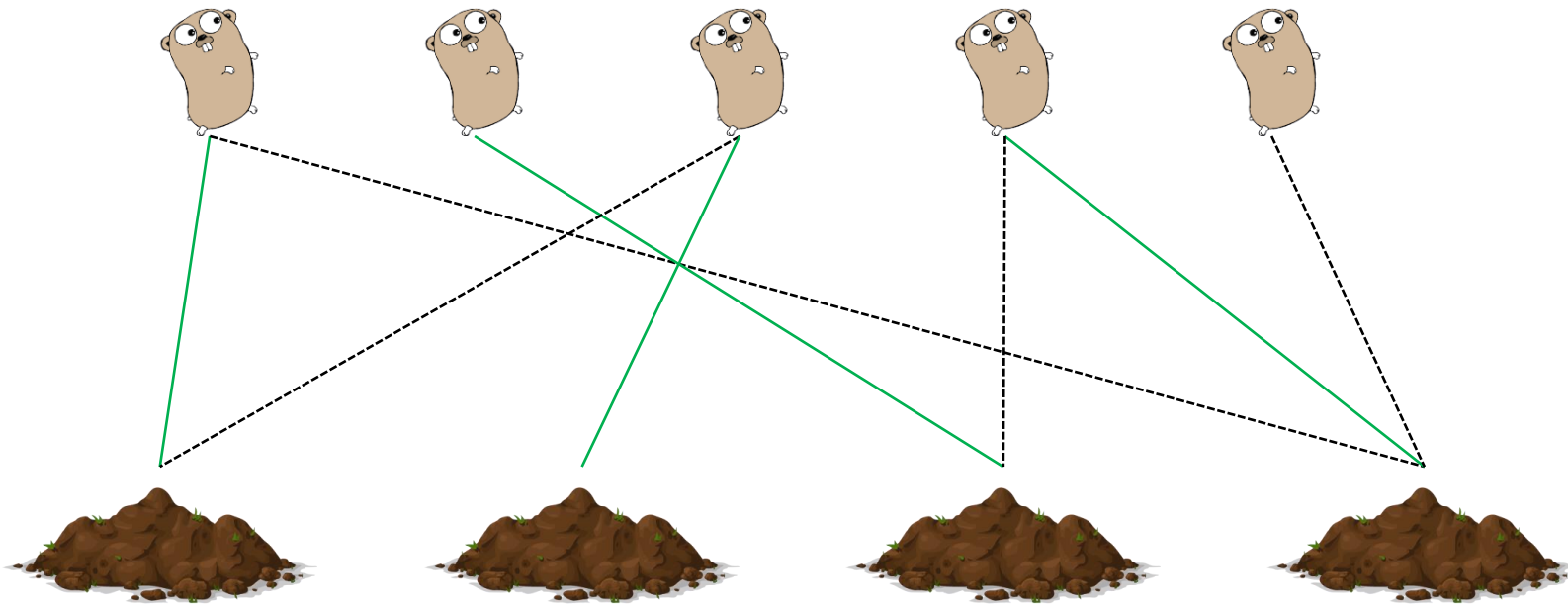
Proposed Algorithm

Kuhn's Algorithm – Example



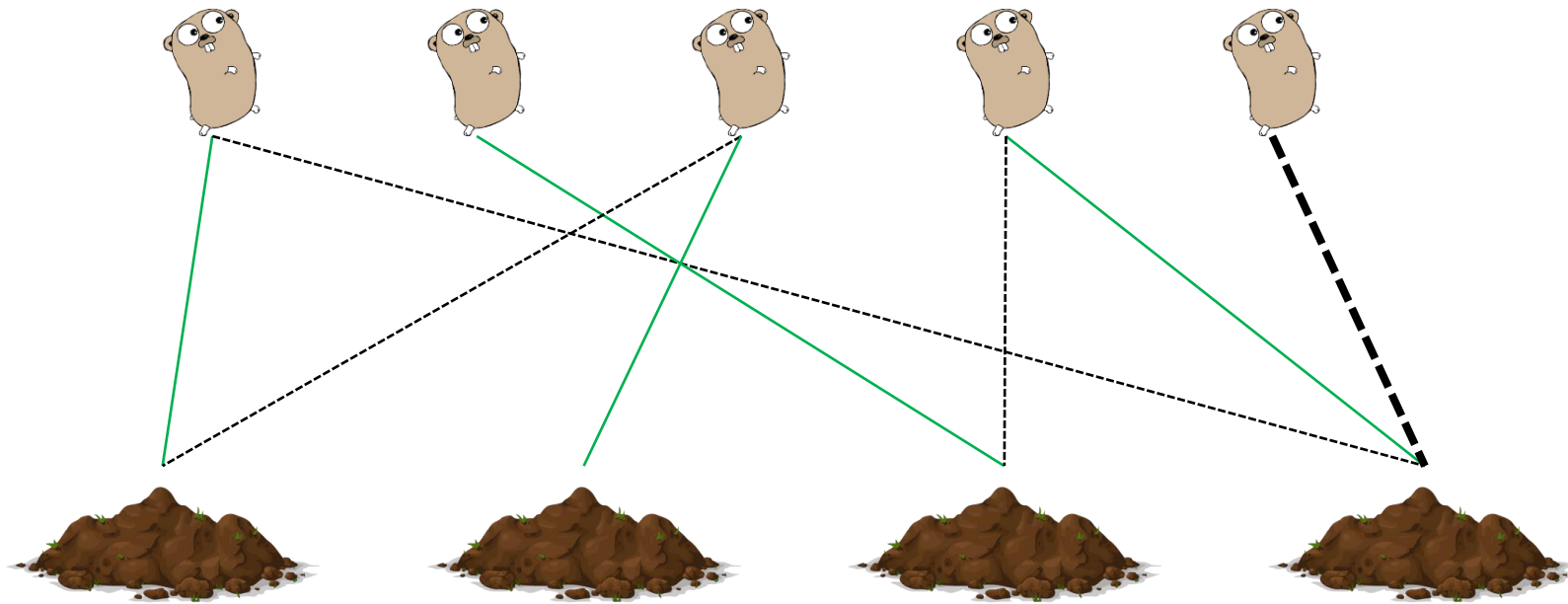
Proposed Algorithm

Kuhn's Algorithm – Example



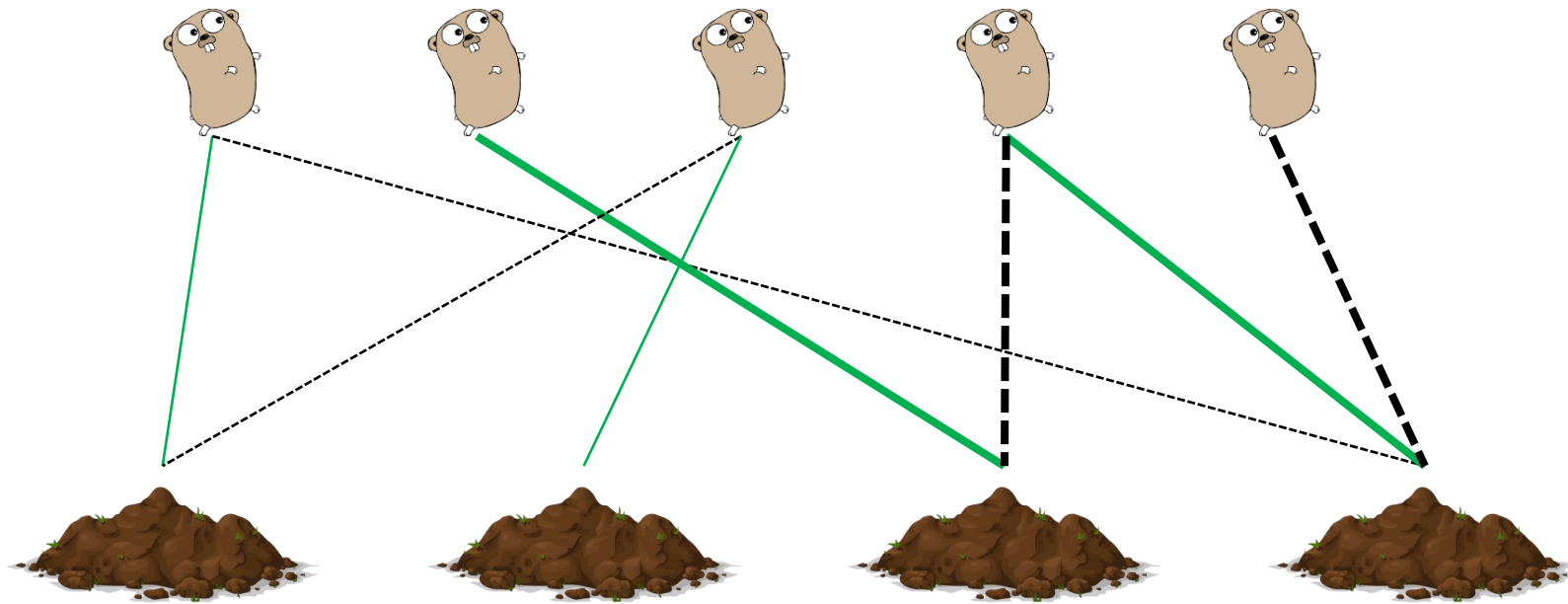
Proposed Algorithm

Kuhn's Algorithm – Example



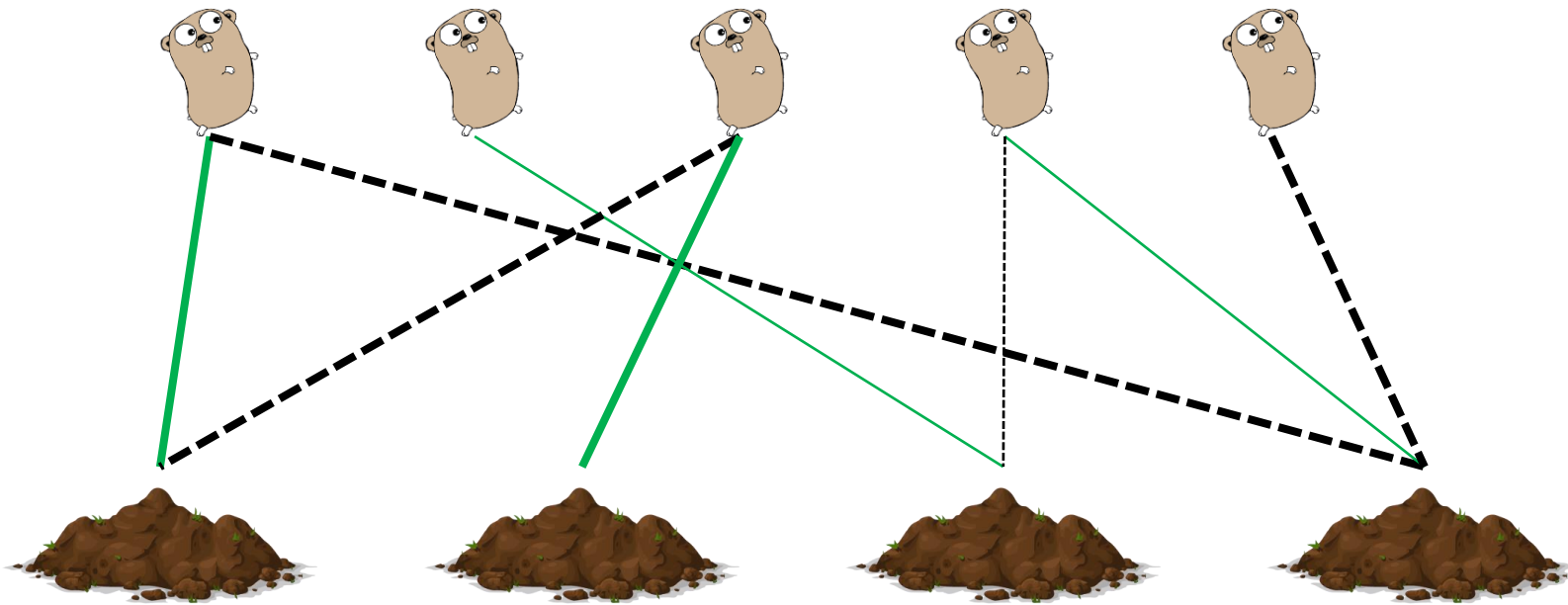
Proposed Algorithm

Kuhn's Algorithm – Example



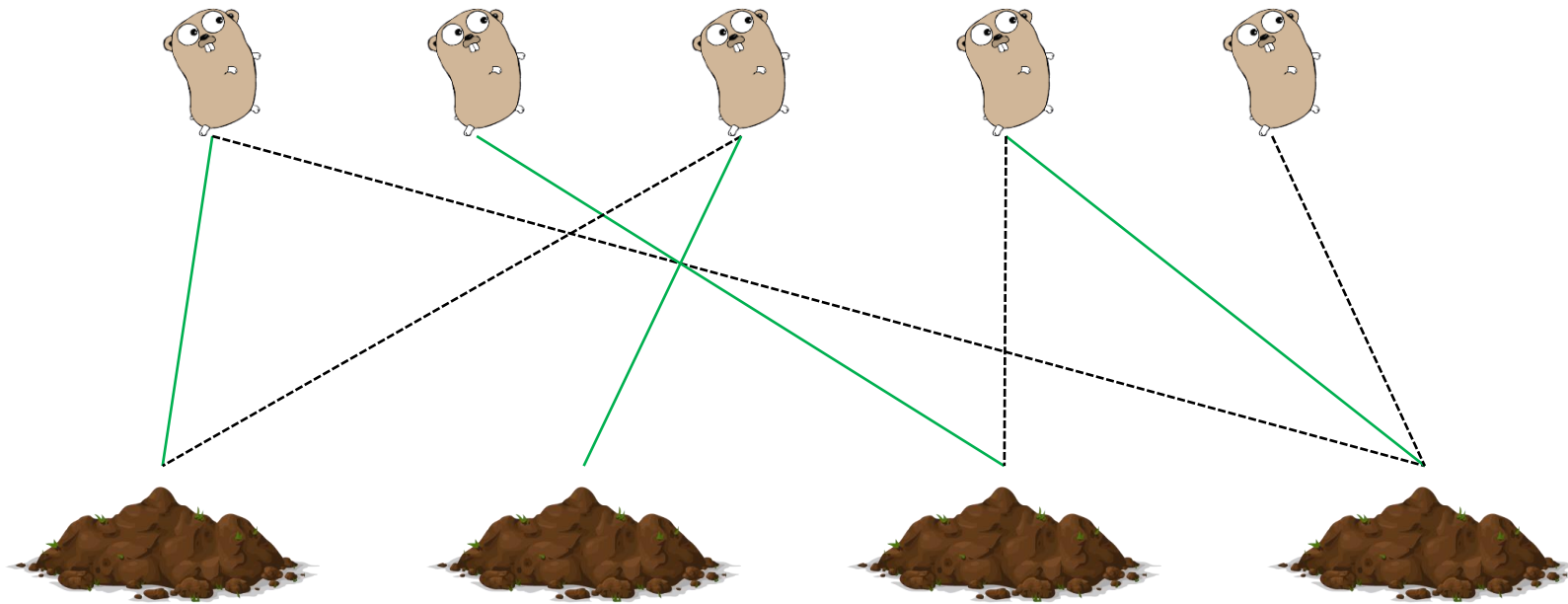
Proposed Algorithm

Kuhn's Algorithm – Example



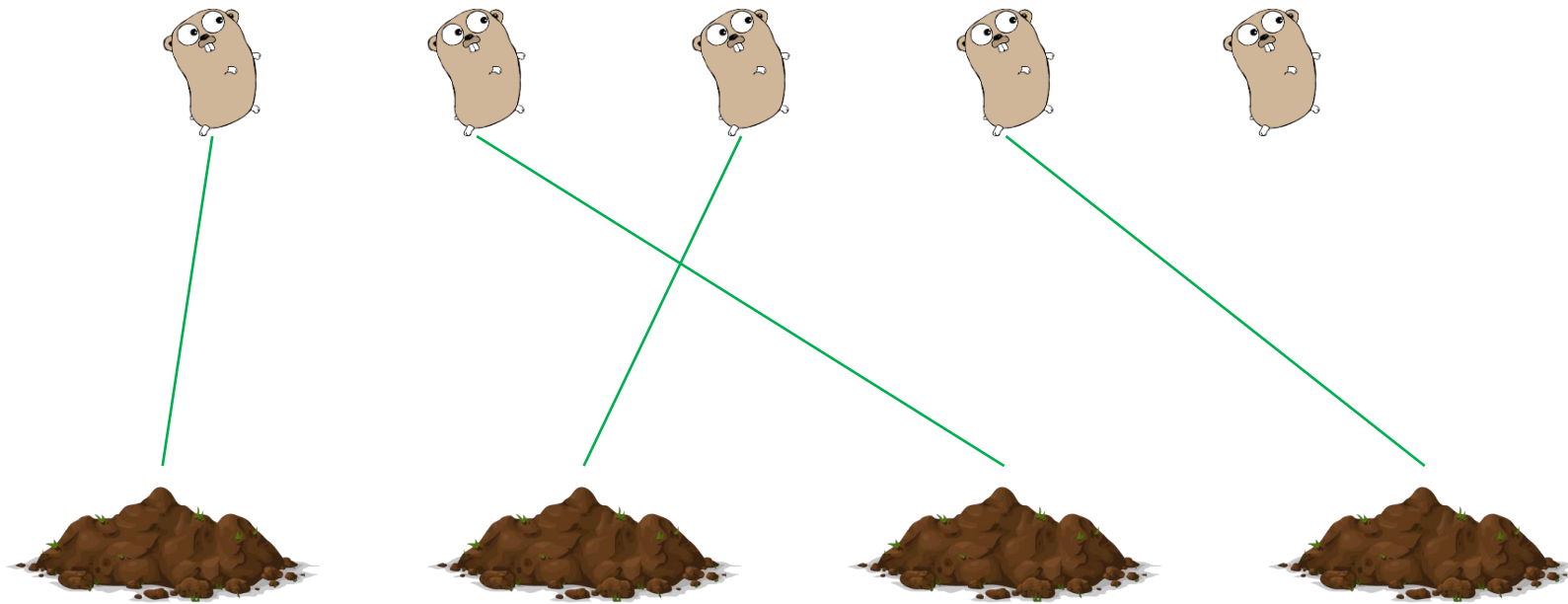
Proposed Algorithm

Kuhn's Algorithm – Example



Proposed Algorithm

Kuhn's Algorithm – Example



Proposed Algorithm

Kuhn's Algorithm – Implementation

(Definitions & Traversal)

```
vector<vector<int>> edges; // All the edges, gopher_i -> hole_i
vector<int> mt, v; // matching, visited; matching is in form of "which gopher get's to use this hole?"
vector<pair<float, float>> gophers;
float x, y;
int s, vl, n, m, dist; // seconds, velocity, n_gophers, m_holes, max_distance (time * velocity)
bool augment(int i) {
    if (v[i]) return false;
    v[i] = true;
    for (auto e : edges[i]) {
        if (mt[e] == -1 || augment(mt[e])) {
            mt[e] = i;
            return true;
        }
    }
    return false;
}
```

Proposed Algorithm

Kuhn's Algorithm – Implementation

(IO & Main loop)

```
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    while (true) {
        edges.clear();
        mt.clear();
        v.clear();
        gophers.clear();
        cin >> n;
        if (cin.eof()) break;
        cin >> m >> s >> vl; dist = s * vl;
        edges.resize(n);
        for (int i = 0; i < n; ++i) {
            cin >> x >> y;
            gophers.push_back(make_pair(x, y));
        }
        for (int i = 0; i < m; ++i) {
            cin >> x >> y;
            for (int j = 0; j < n; ++j) {
                float gx = gophers[j].first, gy = gophers[j].second;
                float dx = x - gx, dy = y - gy;
                float d = dx * dx + dy * dy;
                if (d <= dist * dist) edges[j].push_back(i);
            }
        }

        mt.assign(m, -1); // start with empty matching
        for (int i = 0; i < n; ++i) {
            v.assign(n, false);
            augment(i);
        }

        int t = 0;
        for (auto gopher : mt) {
            if (gopher != -1) ++t;
        }
        cout << (n - t) << '\n';
    }
}
```

Proposed Algorithm

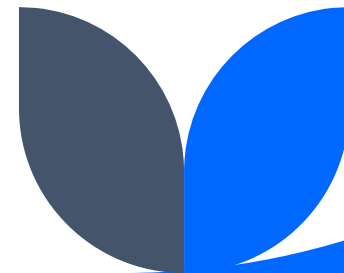
Kuhn's Algorithm – An optimisation

A simple optimisation one could add is to first find an arbitrary matching using some heuristic before running the algorithm, instead of starting with the empty matching.

One simple heuristic is to just iterate over all the vertices in one part and add a random edge to the matching if you can.

Another good heuristic is as follows: At each step, find the vertex of the smallest degree that is not isolated, select any edge, and add it to the matching. A lot of the time it even builds the maximum matching for random graphs!

Note that you will have to modify your main function to check that vertices are not already part of the matching before searching for augmenting paths.



Notes

- Kuhn's algorithm is a subroutine in the Hungarian algorithm
- There are more efficient algorithms than Kuhn's, such as the Hopcroft-Karp-Karzanov algorithm, that runs in $O(\sqrt{nm})$ time.
- The minimum vertex cover problem is NP-hard for general graphs, but you can use maximum matching algorithms to solve it in polynomial time for bipartite graphs.

Problems

[Kattis – Gopher II](#)

[Kattis – Borders](#)

Sources

cp-algorithms.com/graph/kuhn_maximum_bipartite_matching.html

Sources - Images

https://en.wikipedia.org/wiki/Bipartite_graph#/media/File:Simple_bipartite_graph;_two_layers.svg

https://en.wikipedia.org/wiki/Bipartite_graph#/media/File:Simple_bipartite_graph;_no_crossings.svg

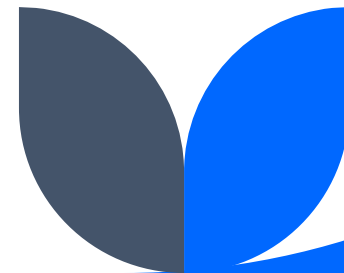
<https://go.dev/blog/gopher>

[https://en.wikipedia.org/wiki/Go_\(programming_language\)#/media/File:Golang.png](https://en.wikipedia.org/wiki/Go_(programming_language)#/media/File:Golang.png)

<https://pixabay.com/vectors/dirt-soil-nature-gardening-earth-576619/>

https://openclipart.org/image/2400px/svg_to_png/301049/publicdomainq-tortoise.png

<https://factorcode.org/logo.png>





Questions?

