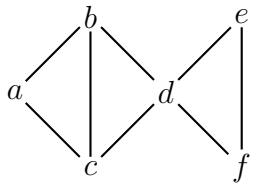
Lecture Outline:

- Foundations of molecular algorithms: Hamiltonian paths and Adleman's experiment
- Clarifying the model: building actual DNA tile glues
- Algorithm analysis: Defining the program size of a tile system
- Constructing triangles

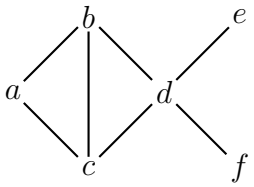
A quick aside on *NP* and *NP-completeness*:

- A problem is in *NP* (i.e. the set of all non-deterministic polynomial time algorithms) if we know no polynomial time algorithm that generates a solution to the problem, but we may verify whether or not a given solution in some "special" form can be verified in polynomial time.
- A problem L is *NP-complete* if all problems in NP can be reduced to L by way of a polynomial time algorithm.
- It can be instructive to think of a problem as a *proof*. Just because we may have an example of a solution does not mean we can prove that a solution exists. Alternatively, just because we cannot find a solution does not mean that we can prove the solution does not exist.

Hamiltonian Path Problem
Input: Given $G(V, E)$, is there a simple path that connects all vertices in V. (Note that a simple path does not allow cycles, which implies that the path hits each node exactly once.)



This graph *has* a Hamiltonian path, $a \to b \to c \to d \to e \to f$



This graph *does not have* a Hamiltonian path

The Hamiltonian Path problem is NP-complete. Given a path P, it is easy to check whether P is or is not a solution by a simple linear pass through the nodes (at each node in the path, we must check (i) that each vertex has not yet been visited and (ii) that there exists an edge from the current vertex to the next vertex).

Adleman's Experiment

Adleman constructed a polynomial-size tile system to solve this problem. His construction is illustrated by example below:

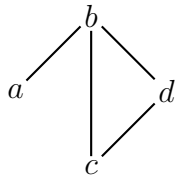


Figure 1: Example graph

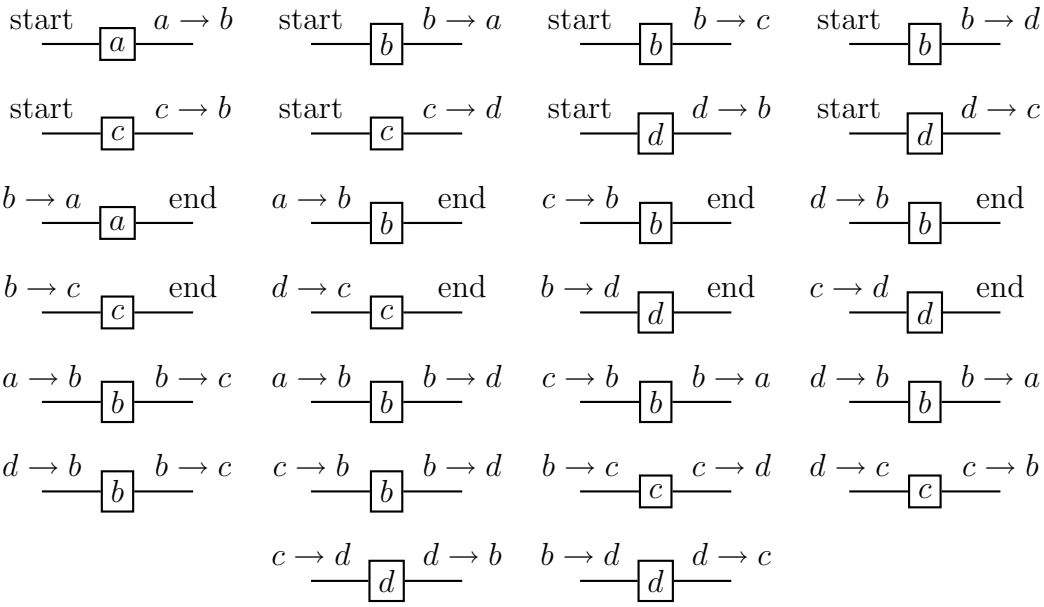


Figure 2: Tiles for solving Hamiltonian Path for Example Graph

The tile system defined in can be used to find any Hamiltonian paths of according to the following algorithm, after allowing the tiles to form in solution.

Consider all DNA tile strings S_0
Let $S = \{s \in S_0$ such that length$(s) = n = |V|\}$
For each $v \in V$,
  Let $S = \{s \in S$ such that $s_i = v$ for some $1 \leq i \leq n\}$

Presuming there is a Hamiltonian path on the given graph, after the algorithm is completed we are left with DNA strings of length $|V|$ that pass through each of the nodes on the graph. If there is no Hamiltonian path for the given graph, then we will not have any DNA strings remaining after the filtration process.

Adleman himself conducted this experiment in 1994, publishing *Molecular computation of solutions to combinatorial problems* in Science **266**, Issue 5187, pp. 1021-1024. Upon successfully finding the Hamiltonian path of a six-vertex graph, he concludes, "This experiment demonstrates the feasibility of carrying out computations at the molecular level."

Indeed, this molecular algorithm is highly efficient when we consider that it only requires a polynomial number of tile types. (Because each tile may be considered a simple three node path, we require at most n^3 different tile types, and considerably fewer as the number of edges falls well below its $n(n-1)/2$ upper bound.) Additionally, the number of filtering operations that must be performed is linear in the number of vertices in the graph.

These successes do not tell the entire story; this computation succeeds only because of the fact that putting DNA tiles in solution is essentially equivalent to calling upon a parallel computer cluster with a very large number of processors. Each 'computer' constructs one possible walk on the given graph, and then those that are not Hamiltonian paths are filtered out. In order to have true success, we must have enough copies of each DNA tile in solution such that it is likely to obtain a Hamiltonian path of a graph by random generation (should a Hamiltonian path exist). Without any optimization to bias the random generation of walks towards constructing Hamiltonian paths, one requires $\mathcal{O}(n^n)$ copies of each tile in solution.

Construting DNA Tile Glues

How exactly will we be able to code the glues between our DNA tiles using DNA strings? We need a way to construct a number of different DNA strings (representing the number of glues) that will only attach to their complementary strings and not to any of the other strings encoding the other glues.

To assist in this process, we construct our glues by creating random strings of some length. The chemistry of the system is one such that complementary DNA strings of length U or greater will always bind together whereas those of length L or lesser will not.

Consider the following two 20-base DNA strands:

```
1 ACCTG ATTGA CTCAG TATGG
1 CGAAC TGAGT CAGGG GACCT
```

These two strands demonstrate the possible undesired interaction between two DNA strings. We realize that randomness will allow us to comfortably construct DNA glues without worrying about such undesired interactions.

The probability that any two randomly chosen bases are complementary is $\frac{1}{4}$. Therefore, the

probability that L pairs of bases chosen randomly are all complementary is 4^{-L}. Now, we are constructing n random strings (one for each glue), which means that there are $n(n-1)/2 < n^2$ ways to pair these strings. And finally, each pair of strands of length U may have complementary sequences of length L anywhere along the string, yielding $(U-L)^2 < U^2$ possible starting points.

Thus, we have upper bounded the probability of an undesired interaction of length L in n randomly generated strings of length U:

$$\mathbb{P}\{\text{Undesired Interaction}\} < U^2 n^2 4^{-L}$$

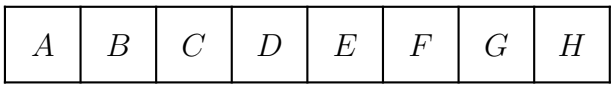
This probability is small enough that we can actually construct DNA tiles. For example, constructing 12 glues of 20 bases where the minimum interaction length for stability is 10 bases, the probability of undesired interaction is less than 5.4%.

Formulating Tile Systems as Computational Devices

A tile system is analogous to a program for constructing some desired shape. In this analogy, it is informative to equate we equate the *program size* with the number of different tiles used. The goal then is to find the smallest program size that allows us to create the desired shape. In fact, we have theoretical lower bounds on program size that we may hope to approach.

Lemma (Pumping Lemma)**.** *To construct a line of length n, one must use a minimum of n different tile types.*

Proof. Consider a line of length n with different tiles:



Assume, by way of contradiction, that two tiles are actually the same (without loss of generality, say they are tiles B and F). If this is the case, then in the above, we may replace F with B, and then connect CDE to its right. However, now a problem arises that B may connect to the right of E. In essence, if B and F are the same tile then we have constructed a line of infinite length:

We have thus reached a contradiction; therefore, we must use a minimum of n distinct tiles to construct a line of length n. □

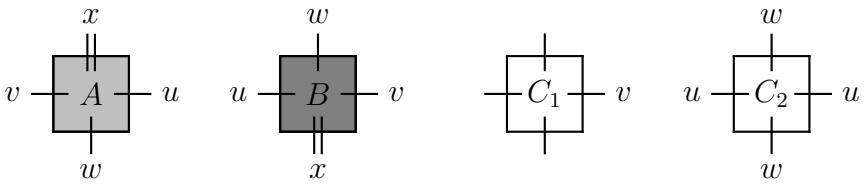
Lemma. *To construct a square with side length n, one must use $\Omega_{io}(\log n / \log\log n)$ distinct tile types.*

Proof. : Using Kolmogorov Complexity. Each tile uses at most four different glues, and thus there are at most $4k$ glues for the entire system, yielding $\log 4k$ bits of information. Therefore, k tiles each with 4 glues of $\log 4k$ bits of information yield total system information $4k \log 4k$. In order to describe a square with side length n tiles, we must have at least $\log n$ bits of information.

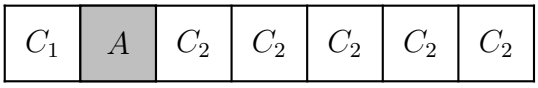
This yields an inequality that lower bounds the number of tiles: $4k \log 4k \geq \log n$. We satisfy this equality if and only if the number of tiles k is $\Omega_{\text{io}}(\log n / \log\log n)$. □

Constructing Triangles

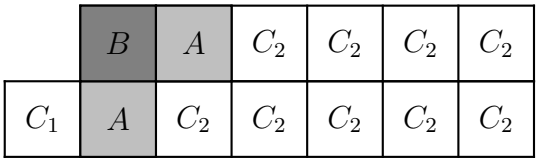
Here we follow the *Rothemund-Winfree Construction* that allows us to create a triangle. Consider the following four types of tile:



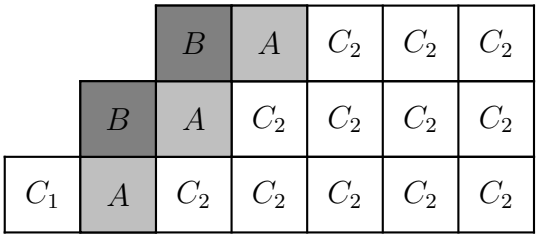
If we are somehow able to construct the following program seed, then we will end up with a triangle. For the time being, we will ignore the problem of constructing this seed, because it is actually quite complex to do this. Say the temperature is $\tau = 2$.



There is only one stable site to attach a tile—we may place a B tile above the A tile. After this happens, the site immediately to the right of this new B tile will allow attachment of another A tile. Then, each of the sites to the right will have support from the left to attach a C_2 tile, and this process will repeat, filling in the row until the end of the base structure:



Not surprisingly, this same process will happen again, adding another row:



And therefore the termination condition is as follows: