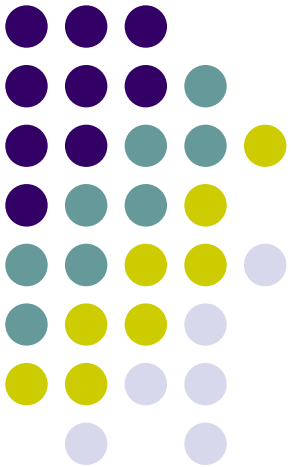
# CSCI 2570 Introduction to Nanocomputing

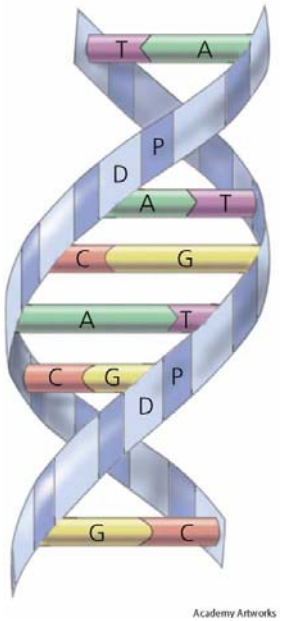
DNA Computing

John E Savage

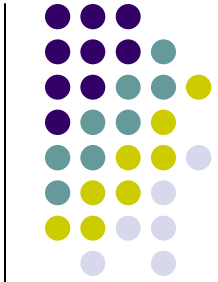


# DNA (Deoxyribonucleic Acid)

- DNA is double-stranded helix of nucleotides, nitrogen-containing molecules.
- It carries genetic information of cell, encodes information for proteins & can self-replicate.
- Base elements form rungs on double helix.
  - They occur in pairs: A-T (adenine-thymine), C-G (cytosine-guanine).
- Sugars and phosphates form sides of helix.

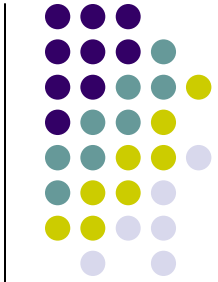


# RNA (Ribonucleic Acid)



- RNA synthesized from DNA.
  - Genetic information carried from DNA via RNA.
- RNA is a constituent of cells and viruses
- RNA consists of a long, single stranded chain of phosphate and ribose units of bases.
- Bases are adenine, guanine, cytosine and uracil.
- Determines protein synthesis and transmission of genetic information.
- RNA can also replicate.

# DNA Hybridization

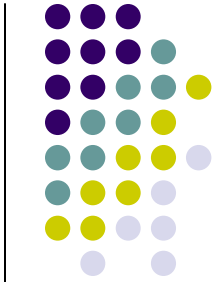


- We assume that only Watson-Crick complementary strings combine.
- Form oligonucleotides (2 to 20 nucleotides).
- General framework for computing with DNA:
  - Mix oligonucleotides in solution.
  - Heat up solution.
  - Cool down slowly to allow structures to form
- **We show that DNA is as powerful as a Turing machine!**

# DNA is a Form of Nanotechnology

- Double helix diameter = 2.0 nanometers.
- Helical pitch (dist. between rungs) = .34 nms.
- Ten base pairs per helical turn.
- ~3 x $10^9$ base pairs in human genome

# Computing with DNA



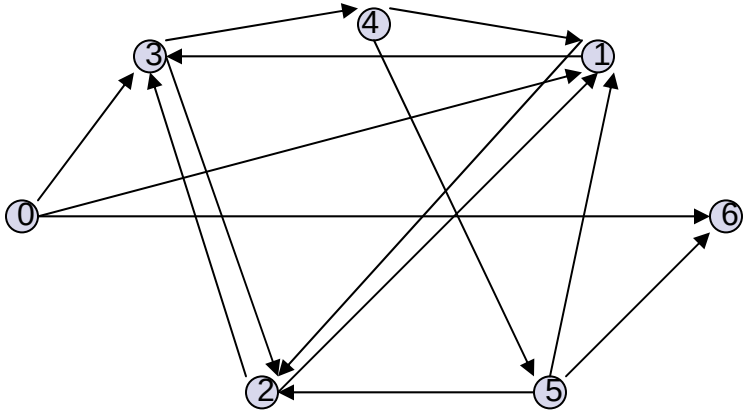
- Prepare oligonucleotides (“program them”)
- Prepare solution with multiple strings.
- Only complementary substrings q and $\underline{q}$ combine, e.g. q = CAG and $\underline{q}$ = GTC
- E.g. GCTCAG + GTCTAT =

```
GCTCAG
 | | |
GTCTAT
```

- 1D & 2D crystalline structures **self-assemble**

# Hamiltonian Path (HP) Problem

- Directed graph G = (V,E)
- Determine if there is a path beginning at $v_{in}$ & ending at $v_{out}$ that enters each vertex once.

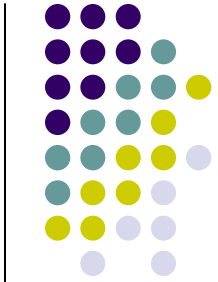



- This graph has HP from $v_{in} = 0$ to $v_{out} = 6$

# Why is Hamiltonian Path Problem Hard?

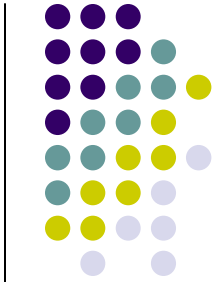
- Intuitively, the number of paths that must be explored grows exponentially with the size of the graph.
- Finding a Hamiltonian path using a naïve search algorithm requires exponential search time.
- Formally, it has been shown that the Hamiltonian Problem is NP-hard.

# HP Problem is NP-Hard



- NP is a class of important languages.
  - A problem *Q* (a set of **instances**) is in **NP** if for every "Yes" instance of the problem there is a witness to membership in *Q* whose validity can be established in polynomial time in the instance size.
- The hardest problems in **NP** are NP-complete.
  - For a problem *Q* to be **NP**-complete, *Q* must be in **NP** and every problem in **NP** must be reducible to *Q* in polynomial time. (Each problem can be solved by translating it to *Q*.)
- If any **NP**-complete problem is in **P** (or **EXP**), so is every other **NP**-complete problem.

# Adleman's Algorithm



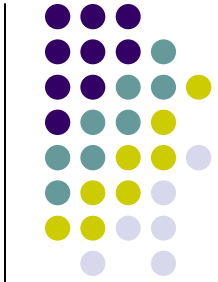
1. Generate random paths through the graph.
2. Keep paths starting with $v_{in}$ & ending with $v_{out}$
3. If the path has *n* vertices, keep only paths with *n* vertices.
4. Keep all paths that enter each vertex at least once.
5. If any paths remain, say "Yes". Otherwise say "No."

# Hybridization to Create Paths

- Adleman† denotes vertex *v* by DNA string (or **strand**) $p_v q_v$. Strands must long enough that they are unique.
- Edge (*u*,*v*) is denoted by $q'_u p'_v$ where p' and q' are the Watson-Crick complements of p and q
- Mix many copies of edge and vertex strands are put into solution along with copies of $p'_{in}$ and $q'_{out}$.
- Adleman used 20-mers in his experiments, |pq| = 20.

†"Molecular Computation of Solutions To Combinatorial Problem," Science, 266: 1021-1024, (Nov. 11) 1994.

# Generating Random Paths Through the Graph



- Edge strings $q'_u p'_v$ combine with vertex strings $p_v q_v$ to form **duplexes**, shown below.

$q'_u p'_v$ $q'_v p'_w$

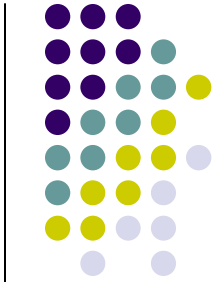
GTATATCCGAGCTATTCGAG CTTAAAGCTAGGCTAGGTAC

CGATAAGCTCGAATTTCGAT CCGATCCATGTTAGCACCGT

$p_v q_v$ $p_w q_w$

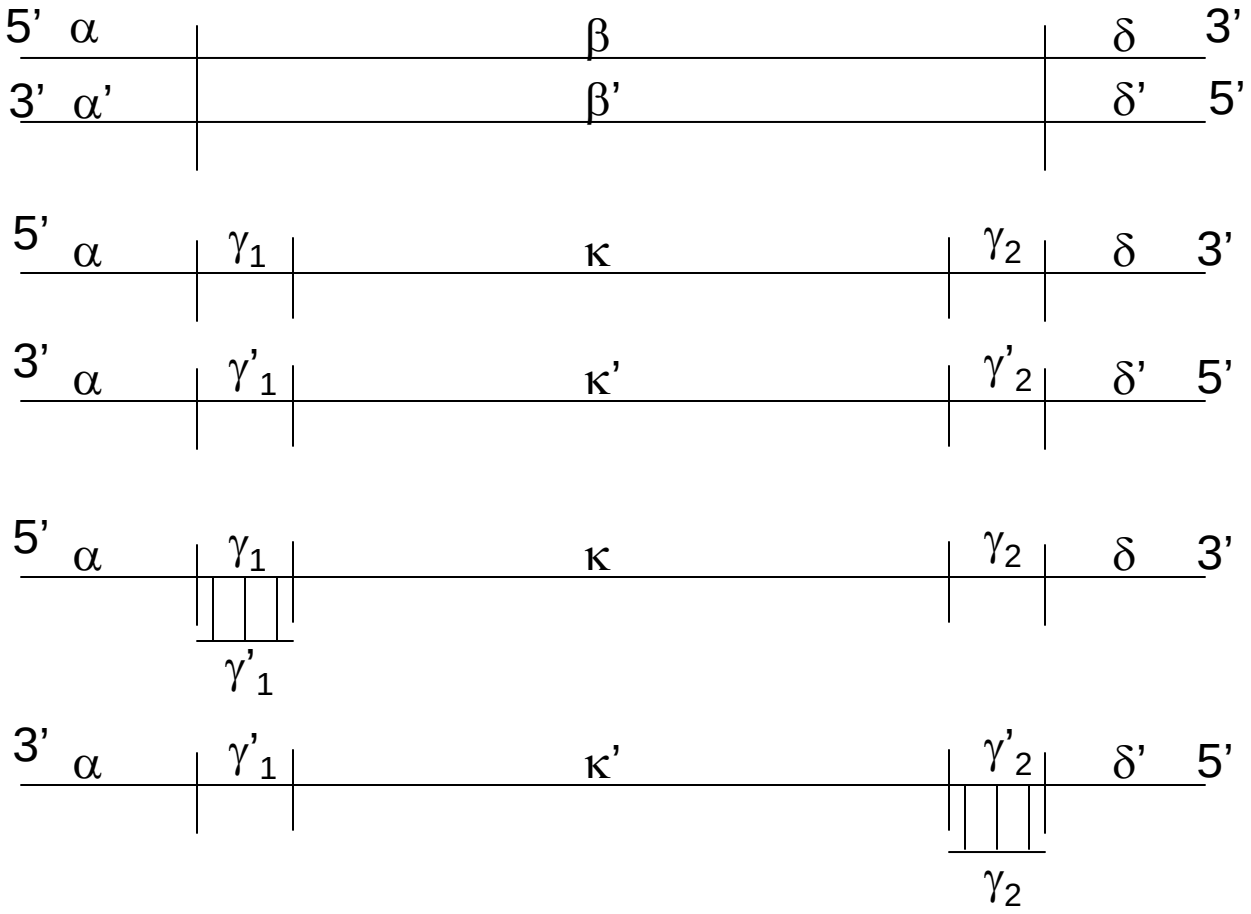
- Each duplex has two sticky ends that can combine with another duplex or strand
- For starting and ending vertices $p_v q_v$ and $p_w q_w$ add $p'_v$ and $q'_w$ so that duplexes with sticky ends $q_v$ and $p_w$ are produced.

# Implementing the Algorithm



- Use PCR to amplify strings starting with vertex $v_0$ and ending with $v_6$.

# Polymerase Chain Reaction (PCR) for String Amplification

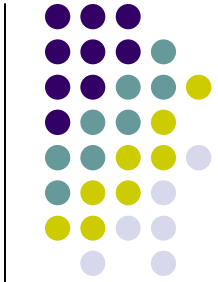



Separate double Strand of DNA

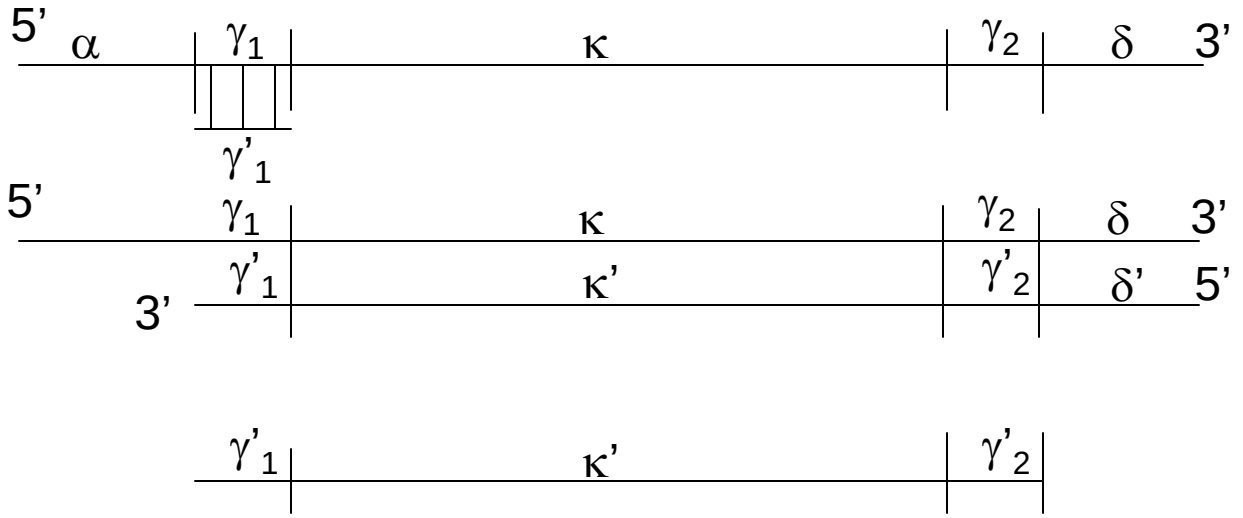
Identify short Substrings

Denature and bind complements of short strings.

# More on PCR



- Polymerase is large molecule that splits double stranded DNA and replicates from 5' to 3' starting it at double stranded section.

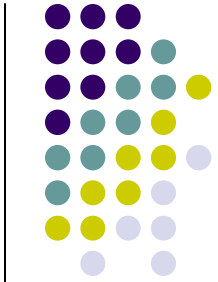



Hybridize $\gamma'_1$ with one strand, $\gamma_2$ with other

Shortened strand clipped at $\gamma_1$.

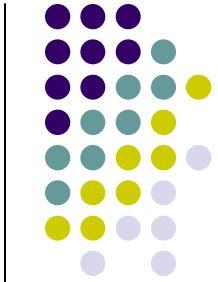
Shorten at $\gamma'_2$ and replicate.

# Chain Reaction

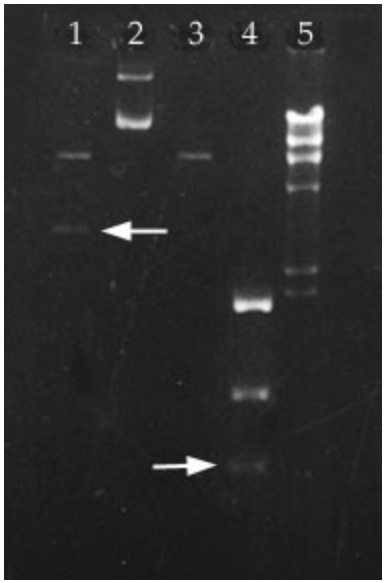


- Clip DNA subsequence at both ends
- Use polymerase to replicate between $\gamma_1$ & $\gamma_2$.
- Replication doubles substring on every step.
- Volume of targeted substring grows exponentially.

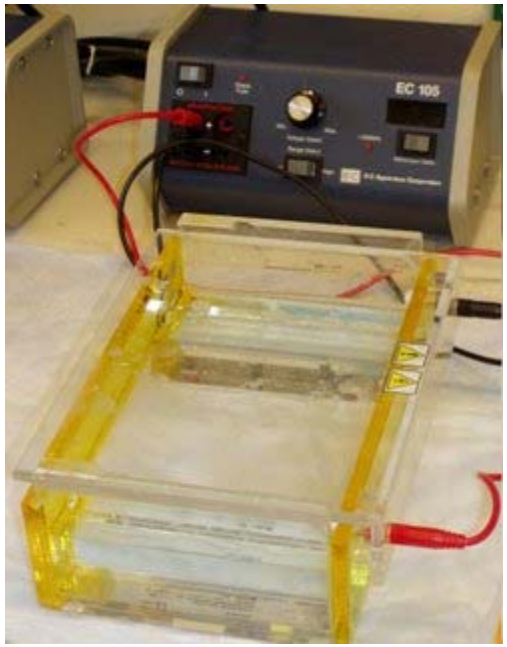
# Implementing the Algorithm



- Use gel electropheris to find strings denoting paths of seven vertices.

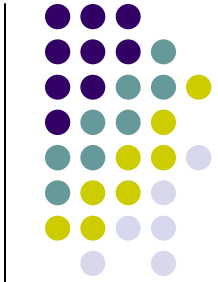


# Setup for Gel Electrophoresis



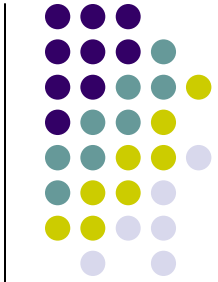

- Figure provided by Wikipedia

# Gel Electrophoresis



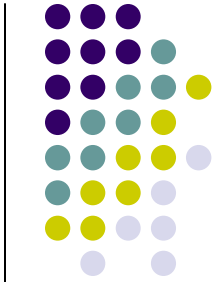
- Separates RNA, DNA and oligonucleotides.
- Nucleic acids are mixed with porous gel.
- Electric field moves charged molecules in gel.
- Distance a molecule moves is approximately proportional to inverse of logarithm of its size.
- Molecules can be seen through staining or other methods.
- Electrophoresis purifies molecules.

# Adleman's Algorithm



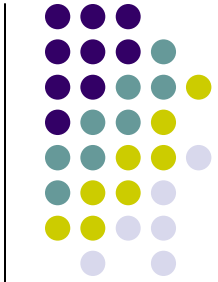
1. Generate random paths through the graph.
2. Keep paths starting with $v_{in}$ & ending with $v_{out}$
3. If the path has *n* vertices, keep only paths with *n* vertices.
4. Keep all paths that enter each vertex at least once.
5. If any paths remain, say "Yes". Otherwise say "No."

# Implementing the Algorithm



- Separate double helix into single stands.
- Separate out strings containing $v_0$ by attaching one copy of $p_0$ that has a magnetic bead attached to it.
- Of those that remain, repeat with $p_i$ for i = 1, 2, …, 6.
- The result are strings of length 7 that contain each of the vertices.
- Amplify the final set of strings using PCR. Use gel electrophoresis to determine if there are any solutions.

# Comments on Adleman's Method

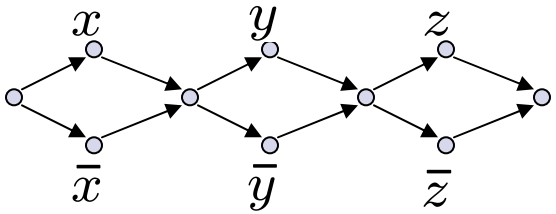


- Long strings $\{p_v\}$ needed to make unlikely that $p_v$ combines with a string other than $\underline{p}_v$.
  - Twenty base elements per string suffice
- Adleman's experiment required 7 days in lab.
  - String amplification, gel electrophoresis
  - Exponential volume of material needed to do tests.
- Method exploits parallelism
  - Nature has lots of parallelism.
  - Unfortunately reaction times are long (secs).

# Extending DNA Computing to Satisfiability

- SAT is defined by clauses: $(x \vee y) \wedge (\bar{x} \vee \bar{y})$

- A set of clauses is “satisfied” if exist values for variables s.t. each clause has value “True”.




- Create a double helix for each path (binary string) as in Adleman’s problem.

# Illustration of Lipton's Method

- SAT is defined by clauses: $(x \vee y) \wedge (\bar{x} \vee \bar{y})$

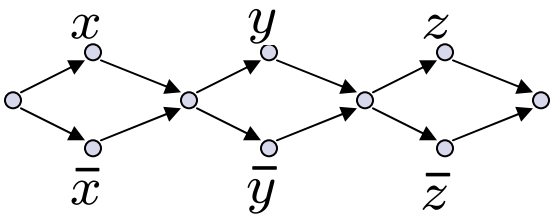
- Lipton† generates all "binary" strings in test tube $t_0$. Filter them according to clauses.
  - Extract strings with $x = 1$.
  - Extract strings with $x = 0$ and $y = 1$.
  - Combine the two sets in test tube $t_1$.
  - Repeat with tube $t_1$ on second clause, i.e. on x' = 1, y' = 1.

- If any strings survive, it's a "Yes" instance of SAT.

†"DNA Solution of Hard Computational Problems," R.J. Lipton, Science, vol 268, p542545m 1995

# Lipton's General Method for Computing Satisfiability

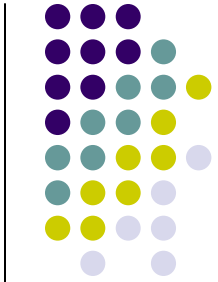
- Create many copies of all paths in $G_{binary}$ below.




Paths correspond to all binary strings

- For first clause produce test tube containing paths satisfying all of its literals.
- Repeat with the second and subsequent clauses.
- If all clauses can be satisfied, *it will be discovered with high probability*.

# Conclusion



- DNA-based computing offers interesting possibilities

- Most likely to be useful for nano fabrication
  - However, high error rates may preclude its use