Introduction to Biocomputing

Computing with DNA (2)

DNA computing

- Bio-molecules (e.g., DNA, RNA, proteins) may be well suited for computations
  - Energy-efficiency
  - High density of stored information
  - Massive parallelism
- It can be proved that in theory, *any algorithm may be implemented* using bio-molecules
  - Store the input on bio-molecules, using a well-chosen design
  - Act on the molecules encoding the input using biotech tools
  - Extract the output from the final content of the test tube
- Practical demonstrations?

Adleman’s experiment

- 1994: the first demonstration of a molecular computation
- Problem of choice: the Hamiltonian Path Problem (HPP)
  - HPP: given a directed graph, decide whether or not it has a Hamiltonian path (NP-complete)
- Tested an instance of the problem with 7 nodes and 12 edges

Adleman’s experiment design

- The nodes of the graph: single stranded DNA of length 20
- The edges of the graph: single stranded DNA of length 20 (based on the strands for the nodes)
- The paths of the graph: double stranded DNA – combinations of the strands for nodes and edges
  - On one strand we have the sequences corresponding to the nodes on the path
  - On the complementary strand we have the sequences corresponding to the edges on the path

Adleman’s algorithmic solution

- Input: a directed graph with *n* nodes, v_{in}, v_{out}
- 1. Randomly generate paths in *G*
- 2. Reject all paths that do not begin in v_{in} and do not end in v_{out}
- 3. Reject all paths that do not involve exactly *n* nodes
- 4. For each node *v* of *G*, reject all paths that do not pass through *v*
- Output: YES if any paths remain, NO otherwise

Adleman: the general idea

- Generate (in parallel) all possible answers
- Reject the wrong answers
- Detect if there are some molecules left in the solution

Other practical demonstrations

- In this lecture
  - SAT
  - DES
  - Chess problems
- Others
  - Tic-tac-toe
  - Poker
  - Logical gates
  - Databases
  - Arithmetic
  - …

The SAT problem

- A logical formula *u* built from variables $x_1, x_2, \ldots$, and the connectives ~, *OR*, & (negation, disjunction, conjunction)
  - *Example:*

    $$u=(\sim x_1 \text{ OR } \sim x_2 \text{ OR } x_3) \;\&\; (x_2 \text{ OR } x_3) \;\&\; (\sim x_1 \text{ OR } x_3) \;\&\; \sim x_3$$
- Truth-value assignment: $f:\{x_1, x_2, \ldots\} \rightarrow \{0,1\}$
- Given *u*, one can compute *f(u)*
- *u* is *satisfiable* if there is a truth-value assignment *f* such that *f* makes u TRUE: $f(u)=1$

Example

$u = (\sim x_1 \text{ OR } \sim x_2 \text{ OR } x_3) \,\&\, (x_2 \text{ OR } x_3) \,\&\, (\sim x_1 \text{ OR } x_3) \,\&\, \sim x_3$

- $f(x_1)$=FALSE, $f(x_2)$=TRUE, $f(x_3)$=FALSE
  - $f(u)$=TRUE
- $f(x_1)$=TRUE, $f(x_2)$=TRUE, $f(x_3)$=TRUE
  - $f(u)$=FALSE

The SAT problem

- Problem: for a given logical formula *u* decide if *u* is satisfiable or not
  - In other terms, decide if there exists a truth assignment that makes *u* TRUE
- Complexity: NP-complete problem
  - There is no solution essentially better than exhaustive search through all 2^k possible truth assignments (k variables)
  - Exponential-time complexity for the best known algorithms

SAT

- Simplifications (?)
  - *Conjunctive normal form*:
    - Any logical formula can be written as a sequence of conjunctions u_1 & u_2 & u_3 &…& u_n
    - Each clause u_i is a sequence of disjunctions $t_{i,1}$ OR $t_{i,2}$ OR … OR $t_{i,ki}$, with each t being a variable or the negation of a variable
    - Example:

      $u=(\sim x_1$ OR $\sim x_2$ OR $x_3)$ & $(x_2$ OR $x_3)$ & $(\sim x_1$ OR $x_3)$ & $\sim x_3$
  - 3-SAT
    - Each clause consists of exactly 3 variables or negation of variables
    - Example:

      $u=(\sim x_1$ OR $\sim x_2$ OR $x_3)$ & $(x_2$ OR x_3 OR $x_4)$ & $(\sim x_1$ OR x_3 OR $x_5)$
- 3-SAT remains NP-complete

Bio-algorithm for SAT

- R.Lipton, 1995: *Using DNA to solve NP-complete problems*
- Idea
  - exhaustive search, made feasible by the massive parallelism of DNA strands
- Sketch of the algorithm
  - generate all possible truth assignments and reject those not satisfying the formula

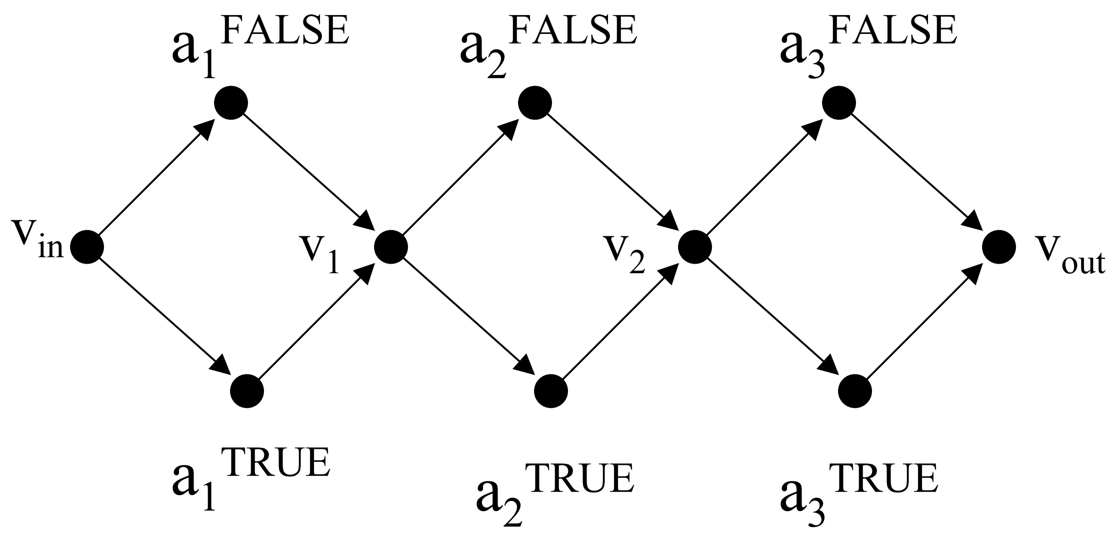
Lipton’s approach to SAT

- Take advantage of Adleman’s idea: reduce SAT to a graph problem
- Major step forward: the initial “soup” is the same for all formulas with the same number of variables
  - In Adleman’s solution, different graphs need different initial “soups”

Lipton's graph

- Idea: looking for truth assignments is essentially the same as finding a path in a graph
- Example:

$u=(\sim x_1 \text{ OR } \sim x_2 \text{ OR } x_3) \& (x_2 \text{ OR } x_3) \& (\sim x_1 \text{ OR } x_3) \& \sim x_3$

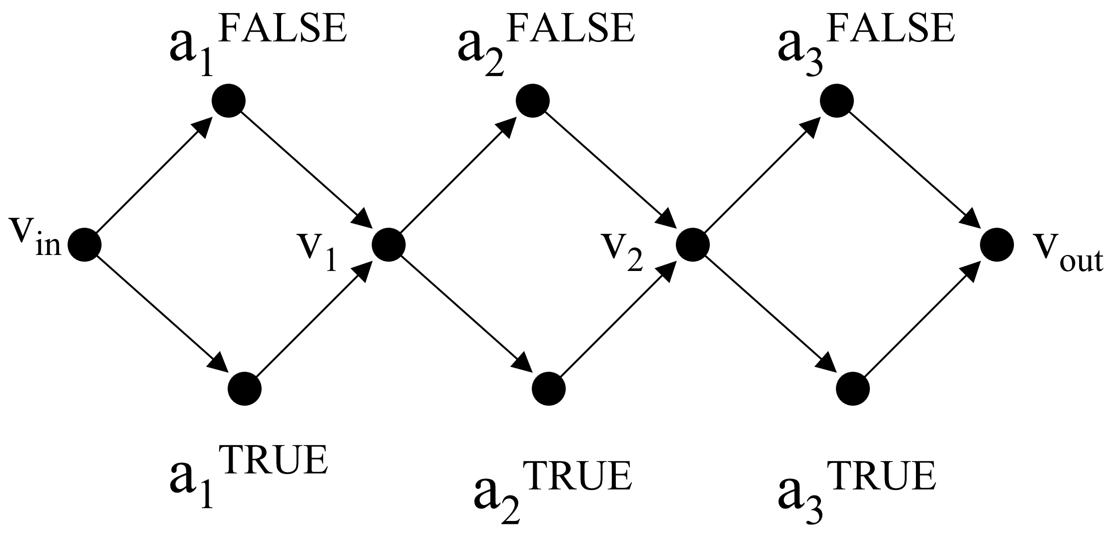


Lipton’s graph

- Example:

$$\boldsymbol{u=(\sim x_1 \text{ OR } \sim x_2 \text{ OR } x_3)\ \&\ (x_2 \text{ OR } x_3)\ \&\ (\sim x_1 \text{ OR } x_3)\ \&\ \sim x_3}$$

- Looking for a truth assignment is the same as looking for a path from v_{in} to v_{out}
- Picking a node from top is to give FALSE to the corresponding variable, node from the bottom is to give value TRUE



Experiment design

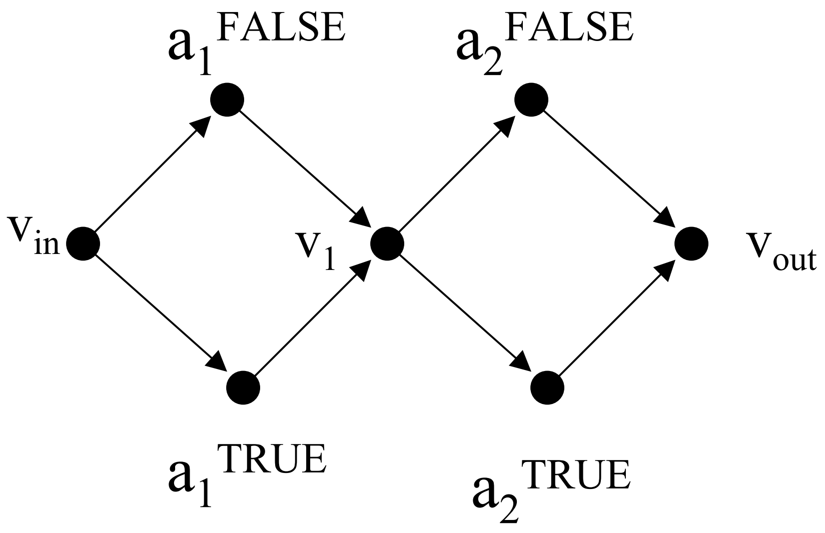
- Vertices (nodes) encoded in 20-mer single stranded DNA
- Edges: 20-mer single stranded DNA designed as in Adleman's experiment
- Lipton encoding: produce an initial "soup" of DNA that encodes all paths from v_{in} to v_{out}
  - As in Adleman's experiment
    - mix all single strands
    - Allow them to hybridize
    - PCR with v_{in} and v_{out} as primers
    - Result: amplify those paths starting in v_{in} and ending in v_{out}

Experiment design

- Note:
  - The graph is the same for all formulas with the same number of variables
  - The initial soup is the same for all formulas over k variables
  - On the other hand, the exact algorithm changes from formula to formula

The SAT problem

- Example: $z=(x_1 \; OR \; x_2) \; \& \; (\sim x_1 \; OR \sim x_2)$
- The associated graph



Bio-implementation

- Example: $z=(x_1 \textit{ OR } x_2) \ \& \ (\sim x_1 \textit{ OR } \sim x_2)$
- Initial test tube: all possible paths from v_{in} to v_{out}, i.e., all possible truth assignments
- Step 1: separate those strands which satisfy the first clause of the formula: $(x_1 \textit{ OR } x_2)$
  - Either x_1 or x_2 must be assigned TRUE
  - Separate from the initial soup those DNA strands containing the sequence a_1^{TRUE}
  - Separate from the initial soup those DNA strands containing the sequence a_2^{TRUE}
  - Mix the two test tubes: all the strands in the result satisfy the first clause
- Step 2: from the result of step 1 (amplified by PCR) separate those strands satisfying the second clause $(\sim x_1 \textit{ OR } \sim x_2)$
  - Either x_1 or x_2 must be FALSE
  - Separate those DNA strands containing the sequence a_1^{FALSE}
  - Separate those DNA strands containing the sequence a_2^{FALSE}
  - Result: Sequence the remaining DNA sequences (if any) to get all possible truth assignments satisfying the original formula

Complexity of the algorithm

- m clauses in the formula: m steps
- k variables: at most k merge and separate
- Problem: cope with the errors
  - Get the result with high probability
- Recent advances
  - Braich, Chelyapov, Johnson, Rothemund, Adleman (2002): 3-SAT problem with 20 variables – exhaustive search through more than 1 million possible solutions

DES

- DES= Data Encryption Standard, 1977
- Secret key cryptography
- Encripts a 64 bit message using a 56 bit key
- Breaking DES = finding the secret key, knowing the encryption of a certain text

Breaking DES

- Classical approaches:
  - Differential cryptanalysis (several days on an electronic computer, needs high number of pairs)
  - Dedicated hardware (expensive, specific to DES, 7 hours)
  - Internet-based (massive parallelism !)
- Bio-approach: it is very general (applicable to any encryption on 64 bits), 1 day of work (with some preprocessing)

DES algorithm

- Plain text – 64 bits; encrypted text – 64 bits; key – 56 bits, expanded to 64 bits
- Composed of 16 rounds
- Each round is based on
  - XOR on 48 bits
  - P-box: permutes the bits of the input
  - S-box: maps 6 bits into 4 bits based on a given table

Plan of an attack on DES based on DNA

- Given a function $f:\{0,1\}^m \rightarrow \{0,1\}^n$ (e.g., the DES encryption), construct a solution T_f containing all pairs (k,f(k))
  - In other terms, for a given text, encrypt that text using all possible keys
  - Compare then with what the system encrypts (denote E that encryption) and find the key it uses
    - Separate those molecules in T_f containing E
      - Result: molecules encoding (k,E) – pairs (key, encryption)
    - Sequence the first half of the molecule to find the key k

Notes on the DES attack

- T_f depends only on the plain text M_0. Denote T_f by DES(M_0)
- DES(M_0) contains 2^{56} DNA strands: less than one liter of water !
- Having DES(M_0), Eve can break many DES systems with very little cost (one day work): generate (M_0,E_0) and find the key as above
  - Eve must be able to use the cryptosystem to encrypt M_0 and then she compares the result with her DNA database to find the key

Constructing DES(M_0)

- Encode all possible 56-bit string into a DNA solution: less than one liter of water
  - Done as in Adleman's and Lipton's experiments
  - Design oligos for each bit, separated by some spacers
  - Allow the oligos to hybridize with each other to form all possible 56-bit strings
- Implement the primitives of the DES circuit: XOR, S-box, P-box
- Apply the algorithm on all 56-bit strings in parallel
- Look for the encoded text E and read the key

Implementing XOR gates

- XOR = exclusive OR
  - Definition:

    a *XOR* b = 1 iff either a=1, or b=1, but not both

    a *XOR* b = 0 iff a and b have the same value
- Easy to prove:

  x XOR y = (x OR y) & (~x OR ~y)
- Already implemented in SAT
  - We know how to implement disjunctions and conjunctions, see Lipton's experiment

Implementing S-boxes

- S-box: essentially a function f: $\{0,1\}^6 \rightarrow \{0,1\}^4$
  - The function f is known from the specifications of DES
- f has 16 possible values
- Implementation: separation
- Example:
  - f(z)=0000 iff z=a or z=b;
  - Separate from the initial soup those strands containing the sequence a
  - Separate from the initial soup those strands containing the sequence b
  - Merge the two test tubes to get all strands giving value 0000 to the function f

Summary of the DES attack

- Construct the initial solution, based on annealing and separation
  - Difficult: 4 months of work (these are old estimates, possibly less time nowadays!)
- Find the (plain_text,encrypted_text) pair
  - One must be able to use the encryption system ONCE
- Apply separation and sequencing on the initial solution to find the key: less than 1 day work in the lab
- Virtues
  - General method: applicable to any 64 bits encryption
  - The initial soup may be used to break any DES
  - Possibility to build such an initial soup and then sell it to anyone interested
- Note: technical errors not taken into account

A chess problem

- Place knights on a chessboard so that no knight is threatening another
- Princeton University (2000): a bio-implementation based on RNA (instead of DNA as in the other experiments shown here)
  - Authors: Faulhammer, Cukras, Lipton, Landweber
  - The problem was considered for a 3X3 chess board

A chess problem

- The problem: can be reduced to SAT
  - Associate to each square a variable that is true if and only if you place a knight on it
  - Once you decide to place a knight on a square, the sqaure which are menaced by it must remain free
  - Clearly, one may write a logical formula to describe the connections:

((~h & ~f) OR ~a) & ((~g & ~i) OR ~b) & ((~d & ~h) OR ~c) & ((~c & ~i) OR ~d) & ((~a & ~g) OR ~f)

A chess problem

- Novelty of the implementation:
  - Use RNA instead of DNA
  - Destructive approach: destroy the unacceptable solutions, rather than separate the acceptable ones (more accurate, more likely to automate)

RNA implementation

- 3X3 chessboard: 10 bit strings (one for backup, the other nine for the nine squares of the board)
- Encode in RNA all possible 10 bit strings as see in the other experiments
- To destroy a string containing 1 on position a:
  - If it contains 0 on position a, then the RNA molecule contains the specific sequence we have designed for that value
  - Add the complementary DNA sequence that sticks to the targeted RNA sequence (!)
  - Use enzyme RnaseH: chews up RNA/DNA hybrids, leaves “normal” RNA alone

RNA implementation

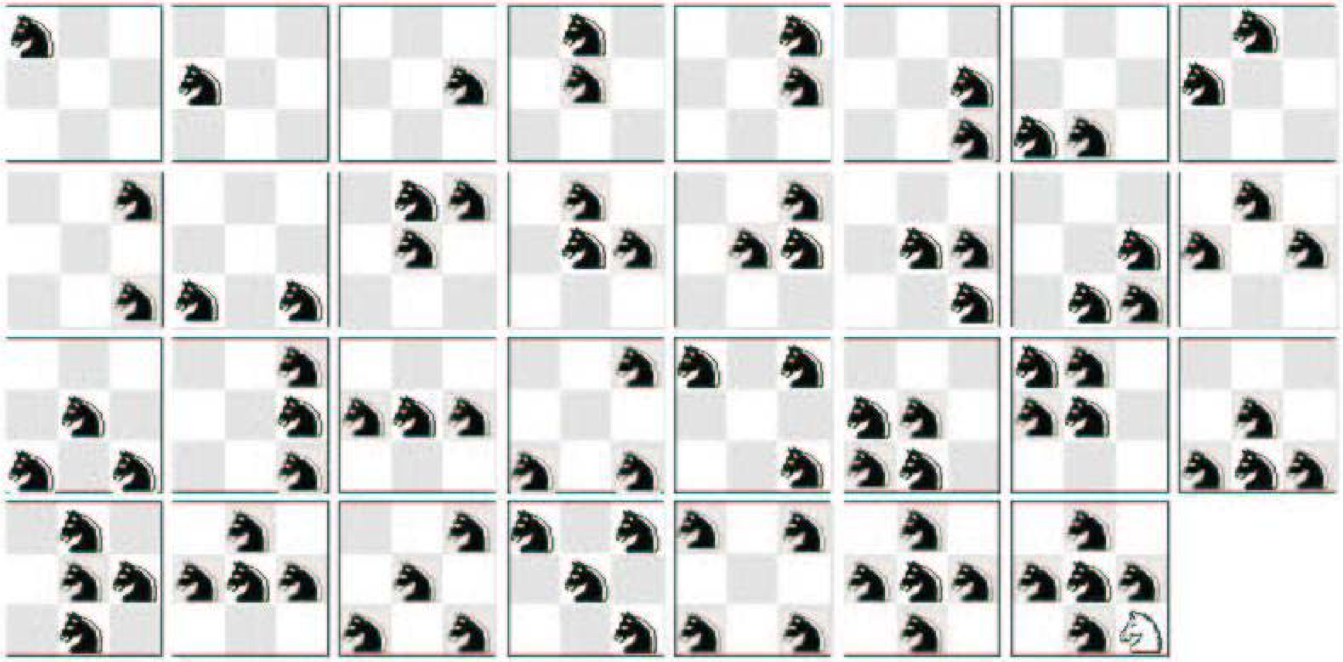
- Example: satisfy the formula

  ~a OR (a & ~h & ~f)
- Divide the solution into two tubes
  - Tube 1
    - Destroy the strings which make first clause false: destroy the strings with 1 on position a
    - Add a DNA strand complementary to the RNA strand encoding a=1
    - Add the enzyme to destroy the hybrids
  - Tube 2
    - Destroy the strings which make the clause false: a=0, h=1, AND f=1
    - Add 3 DNA strands complementary to the RNA for a=0, f=1, h=1

RNA implementation - results

- Out of the final solutions, 43 molecules were randomly chosen and sequenced
- 42 were correct, 1 was wrong
- Altogether, 126 knights placed correctly, 1 wrong: 97.7% success rate

RNA computer plays chess



Molecular Computing – Successes and Challenges

- What can we compute with DNA ?
  - “Killer” application is needed – challenge for computer scientists
  - Better algorithms than exhaustive search – same comment
  - We need better biotech tools to control the molecules (do they exist already?) – challenge for biotech
  - Cope with the errors: impact on the size of the solutions (in number of strands)
  - How much can we compute – SAT up to 70-80 variables → impact on the size of the solutions (in number of strands)

Molecular Computing – Successes and Challenges

- Positive side
  - Applications to biotechnology: e.g., a SAT implementation used to execute Boolean queries on a "wet" database, based on some tags (IDs)
  - Useful in specialized environments: e.g., extreme energy efficiency or extreme information density required
  - Provide the means to control biochemical systems just like electronic computers provide the means to control electromechanical systems

Molecular Computing – Successes and Challenges

- Bad news
  - At this moment, we cannot control the molecules with the precision the physicists and electrical engineers control electrons
  - Need of a breakthrough in biotechnology: more automation, more precise techniques
  - Example:
    - HPP may be solved nowadays on electronic computers for graphs with 13 500 nodes
    - Adleman’s approach scaled up for graphs with 200 nodes needs more DNA than the weight of the Universe

One last thought

- Adleman:

“So here it is (the cell), the most amazing tool-chest you have ever seen. We know it is a great tool-chest, because it was used to build you and me. And even though we are very clumsy in our use of the tools right now, and even though molecular biology has made only a small portion of them available to us so far, we can already use them to build a computer. And if you can build a computer, then presumably many other exciting things can be built.

So, this is the challenge of molecular science: *take the tools and build something great*.”