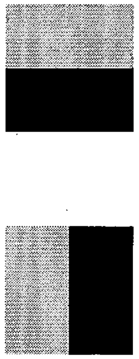
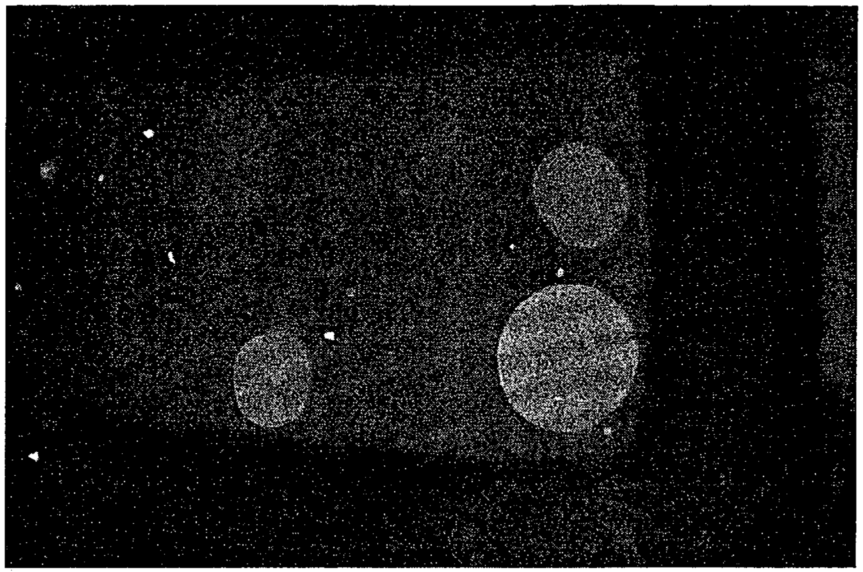
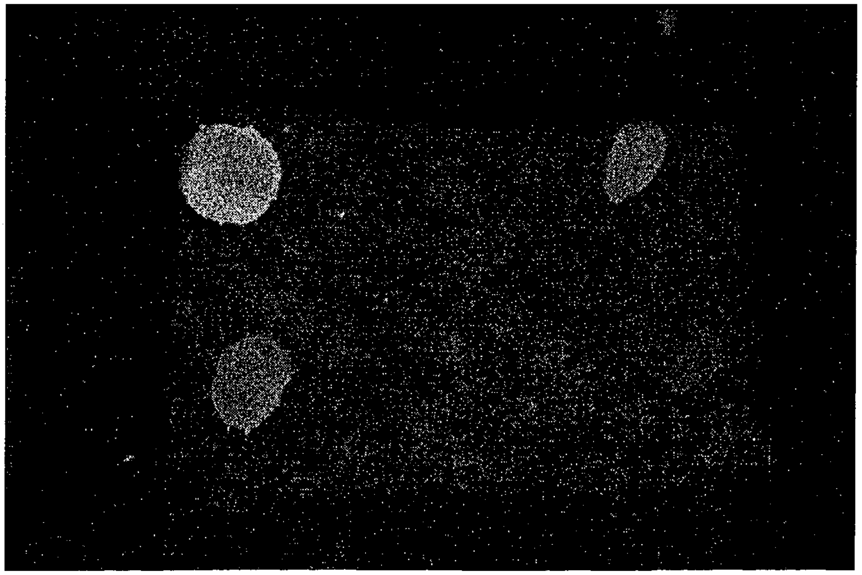
Figure S1





$X \cup Y$

17	56
55	107



$\overline{X} \cup \overline{Y}$

22	17
16	6

$(X \cup Y) \cap (\overline{X} \cup \overline{Y}) = 1$

S2

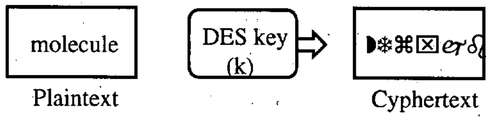
Molecular Computation and Oligonucleotide Arrays

Richard V. Connors and Michael C. Pirrung

1] Introduction:

The emergence of inexpensive, miniaturized computer circuitry has revolutionized the way information is stored and manipulated in modern society. The availability of small, powerful, desktop computers gives users unprecedented power. Desktop PCs connected by local networks, and, in turn, worldwide via the internet, has given rise to tremendous mobility of information. With the advent of this new information age, data storage and manipulation are so efficient that information management has become a central issue. The development and widespread use of the internet has fostered an intense interest in information security among its users, with just cause. Computers presently transfer and control so much money and information that their transactions present a lucrative opportunity for criminals. Moreover, the sensitive information contained in government computer systems is subject to attack by subversive foreign interests. Indeed, the relative anonymity that networks offer and the ingenuity of clandestine pirates has government fearing a future wave of info-crime. Government, paradoxically, can also curtail the development and use of security systems because they limit its ability to eavesdrop on private communications. It is a safe presumption that in the future, secure information exchange will be of vital national and corporate interest. Secure information exchange is offered by cryptography.

Modern cryptography is based on the idea that information stored as alphanumeric strings is fundamentally a sequence of binary numbers. Encryption involves applying a mathematical function (key) to an intelligible binary number sequence (plaintext), transforming it into a nonsense sequence (cyphertext) that can only be read by an individual who possesses the key. The Data Encryption Standard (DES), for instance, uses a unique 56 bit key to transform a 64 bit plaintext message into a 64 bit cyphertext message[1]. To crack DES, one would take a plaintext-cyphertext pair and apply all 2^{56} keys to the plaintext until the cyphertext is obtained. This process



is analogous to trying all possible combinations of a combination lock to open it. This shotgun approach strategy is the only known way to crack most digital and mechanical cryptosystems. The security of such systems is a function of the number of possible combinations. The DES[1], for example, has 2^{56} possible combinations, thus, one would have to try something approaching 2^{56} or 10^{17} different keys before the solution could be found. In general, the cracking of cryptosystems can be seen as a single type of problem with the following characteristics:

- The problem has a large number of possible solutions.
- Only one solution will solve the problem.
- Each potential solution may be checked quickly.

These characteristics are shared by a problem class that has attracted considerable interest among complexity theorists, the class NP-complete. NP-complete problems are notoriously difficult to solve. Most cryptosystems are based on NP-complete problems, hence, if an efficient algorithm was devised to solve problems in NP, many cryptosystems would be compromised. Until recently, no efficient algorithm was available that could solve NP-complete problems. The only algorithms available were variations on exhaustive search. In a pioneering paper, Adelman recently disclosed[2] that the directed Hamiltonian Path Problem (HPP), a famous NP-complete problem, could be solved in reasonable time using DNA and the tools of molecular biology. Before appreciating the implications of Adelman's discovery, a basic understanding of the concept of NP-completeness is essential.

2] Complexity theory:

In recent years, practitioners of complexity theory have focused much of their research efforts on the notion of ***NP**-completeness*, indeed, as Michael Garey and David Johnson noted in their ground-breaking book:[3] *Computers and Intractability - A Guide to the Theory of NP Completeness*, "few technical terms have gained such rapid notoriety as the appellation NP-complete." The concept of ***NP**-completeness* was first introduced by Stephen Cook[4] of the University of Toronto in 1971 and has since come to represent a growing number of inherently intractable problems that have confronted algorithm designers as they have sought to expand the limits of computation. Many commonly encountered problems in mathematics and the sciences are now known to be ***NP**-complete*, and the list is continually growing. Virtually every scientific and engineering endeavor from molecular modeling to spacecraft design faces ***NP**-complete* problems. In fact, the scientific process itself involves formulating and testing hypotheses that are invariably ***NP**-complete* problems. The computer security field is an important example of a specific area that is profoundly affected by the theory of ***NP**-completeness*. Most of the sensitive military and economic intelligence maintained by the US and other G-7 nations is protected by encryption schemes based on "intractable" ***NP**-complete* problems. The United States Data Encryption Standard (DES)[1], for example, is widely used by the US government to protect sensitive information. Clearly, with such wide-ranging implications, the study of complexity theory and the theory of ***NP**-completeness* will certainly attract intense interest from scientists and engineers for many years to come.

2.1] Polynomial-time versus exponential-time:

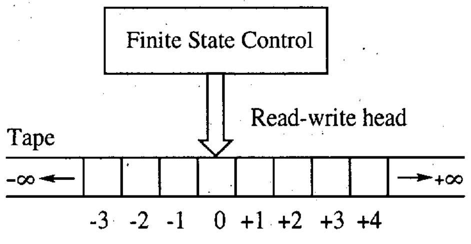
The degree to which the difficulty of a problem increases as its size increases is the central criterion by which complexity theorists judge whether or not the problem is intractable. Usually a *problem* is described by giving a general description of all its parameters or variables and stating what properties the solution is required to satisfy. An *instance* of a problem is obtained by specifying values for the variables. The time requirements of an algorithm designed to solve the problem are expressed in terms of a single variable, representing the *size*, which reflects the amount of input data needed to describe the instance. The *time complexity function* for an algorithm, which is a function of the difficulty of the problem, expresses the maximum amount of time required to solve a problem instance of a given size. Computer scientists have organized algorithms/problems (computations) into two basic classes, *polynomial-time* and *exponential-time*, based on the relationship between size and time complexity function. With polynomial-time computations, the time complexity function is a polynomial function of problem size. Historically, scientists have preferred to work with "easy" polynomial-time problems.

Many scientific laws are expressed as a relationship between two or more measured quantities. Reaction kinetics, for instance, involve following the appearance or disappearance of chemical species during the time course of a reaction. In epidemiological studies, certain behaviors, genes or environmental toxins are linearly correlated with disease states. Much more troublesome are the exponential-time computations, for which the time complexity function is an exponential function of problem size. This type of problem is commonly encountered in molecular or quantum mechanical calculations where the computation time rises astronomically as the number of atoms and bonds increases. Similarly, biochemists wrestle with the challenges of predicting the secondary and tertiary structures of proteins, problems that become insurmountable as the number of peptide residues increases. Pharmaceutical chemists are also confronted with this problem when attempting to create small molecules capable of binding to biologically relevant receptors. The table below clearly illustrates the chasm that opens between polynomial-time and exponential-time complexity functions as problem instance size (n) increases and why the polynomial- exponential-time dichotomy is a fundamental organizing principle in complexity theory. From the perspective of an algorithm designer or computer programmer, polynomial-time problems are easy to solve, meaning their solutions are arrived at relatively quickly. Conversely, while exponential-time problems may be manageable with a few variables, they quickly become intractable as the number of variables increases and the computing time required to find a solution becomes prohibitive. Most exponential-time algorithms are variants of exhaustive searches whereas polynomial-time algorithms are devised based on a deep understanding of the structure of a problem. There is consensus among computer scientists that a problem is not considered well-solved until a polynomial-time algorithm is designed for it and is considered *intractable* if no polynomial-time algorithm can possibly solve it.

Size		Time complexity function		
n	n^2	n^3	2^n	3^n
1	1	1	2	3
2	4	8	4	9
3	9	27	8	27
4	16	64	16	81
5	25	125	32	243
10	100	1000	1024	104
20	400	8000	10^6	10^9
30	900	27000	10^9	10^{12}
40	1600	64000	10^{12}	10^{18}

2.2] P versus NP:

Intractability has two root causes. In the first type, the problem is so difficult that an exponential amount of time is needed to discover a solution (i.e. the solution is too hard to find), and in the second type, the solution is so extensive that it cannot be described with an expression that is a polynomial function of the input length (i.e. the solution is too big to describe). Intractable problems of the second type are super-hard problems that have been termed "undecidable" by computer scientists. Pedagogically speaking, undecidable problems and more difficult problems can be considered an abstract universe in which *all problems* exist. The most difficult problems reside at the shadowy frontiers of this abstract universe and are therefore not well characterized (and may never be). The simplest problems, i.e. those in **P** (polynomial-time), form a well-characterized enclave within the abstract problem universe. Before discussing the **NP** problem types, it is necessary to understand the concept of deterministic and nondeterministic computation. Computer scientists formalize the notion of an algorithm using the *deterministic one-tape Turing machine* (DTM) model, which is illustrated below. The DTM consists of a finite state control, a read-write head, and an infinitely long tape divided into squares labeled ...-2,-1,0,1,2,...



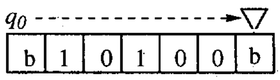
The deterministic one-tape Turing machine

A program for a DTM specifies the following information:

(1) A finite set S of tape symbols, including a subset $I \subset S$ of input symbols and a distinguished blank symbol $b \in S - I$.

(2) A finite set Q of states, including a distinguished start-state q_0 and two distinguished stop-states q_Y and q_N.

(3) A transition function $F: ([Q - \{q_Y, q_N\}], s \in S) \rightarrow (Q, s' \in S, \Delta)$



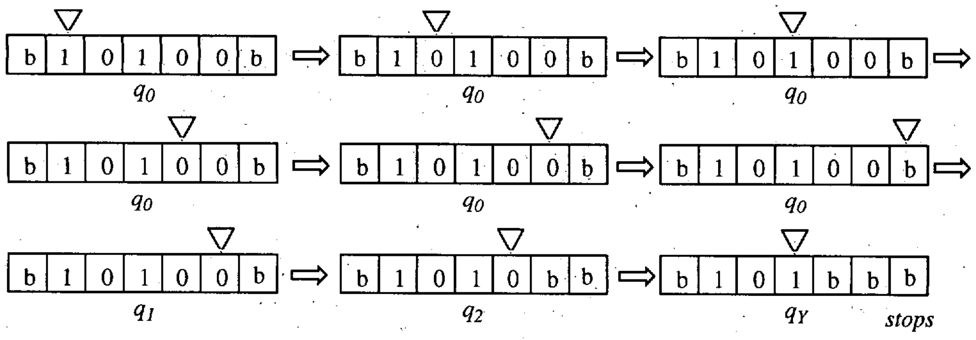
Loading the input string*x* onto the DTM tape

Execution of a DTM program begins at start-state=q_0 by writing the input string x (in this example taken from Garey and Johnson[3] the input string x=10100) onto the tape one square at a time beginning at square 1; all other squares initially contain the blank symbol b. After the input string is written to the tape, the read-write head returns to square 1 and then the rest of the program executes one step at a time. During each step the read-write head reads the contents (s) of the current square, writes in a new value (s') that is prescribed by the program, moves either (+1) or (-1) to the next square as prescribed by the program and then sets the current state q_x to the next q_{x+1}. If the current state is q_Y or q_N the program stops with the answer being "yes" if $q=q_Y$ and "no" if $q=q_N$. A rudimentary DTM program M is illustrated in tabular form below:

$S=\{0,1,b\}$; $I=\{0,1\}$; $Q=\{q_0,q_1,q_2,q_3,q_Y,q_N\}$

State	Transition Function $F(q,s,\Delta)$		
(q)	s=0	s=1	s=b
q_0	(q_0,0,+1)	(q_0,1,+1)	(q_1,b,-1)
q_1	(q_2,b,-1)	(q_3,b,-1)	(q_N,b,-1)
q_2	(q_Y,b,-1)	(q_N,b,-1)	(q_N,b,-1)
q_3	(q_N,b,-1)	(q_N,b,-1)	(q_N,b,-1)

The first step (q_0) of program M after the input string x is loaded proceeds as follows: the read-write head reads the value s=1 from square 1. Because the current state is q_0 and the current square contains s=1, the new values of q,s and Δ are taken from row q_0 and column s=1 of the program table. Thus $(q,s,\Delta)=(q_0,1,+1)$, so the read-write head writes s=1 to square 1 and, because Δ=+1, moves one square right to square 2. Finally, the state is set to q_0. This procedure continues until $q=q_Y$ or q_N at which time the program stops. The computation of M with input x is schematically depicted below:



The computation of the program M on input 10100

This computation stops after 8 steps in state q_Y which means the answer for 10100 is "yes." Generally, it is said that the DTM program M with input alphabet I accepts $x \in I^*$ (I^* is the set of all finite strings over alphabet I) if and only if M stops in state q_Y when applied to input x. The *language* L_M recognized by program M is given by L_M={ $x \in I^*$: conditions under which M accepts x}. For instance, in the above example program M recognizes the language L_M ={ $x \in \{0,1\}^*$: the rightmost two symbols of x are both 0} where $\{0,1\}^*$ is the set of all finite strings containing 0,1. This definition of language recognition does not require that M stop for all input strings in I^*, only for those in L_M. If x belongs to $\{I^*- L_M\}$ then the computation of M on input x might stop on q_N or run interminably. An algorithm is considered useful only if it stops on *all* possible strings over its input alphabet I^*. Stated in familiar terms, *"recognizing languages"* means *"solving problems,"* thus a DTM algorithm is said to solve a problem if it stops for all input strings over its input alphabet. The important thing to remember about the DTM is that, analogous to most modern CPUs, it is a single linear data processor that accesses a single sequential tape and can therefore perform only a very limited amount of work in a single step.

With a formal algorithm model in hand, one can formally define the time complexity function. The time required for execution of a DTM program M on input x is a function of the number of steps until q_Y or q_N is reached. Thus, for a DTM program M that stops on all inputs $x \in I^*$, its time complexity function $T_M : Z^+ \rightarrow Z^+$ (Z^+ is the set of all positive integers) is given by:

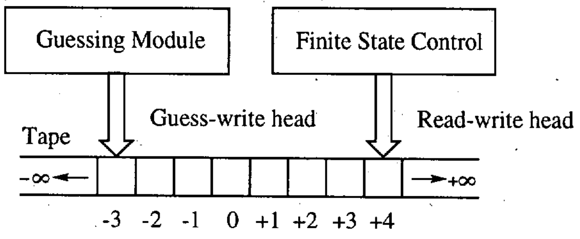
$$T_M(n)=\max\left\{ m: \begin{array}{l} \text{if there is an } x \in I^* \text{ with the length of x=n such} \\ \text{that computation of M on input x takes time m} \end{array} \right\}$$

Such a program M is called a polynomial-time DTM *program* if there exists a polynomial p such that for all $n \in Z^+$; $T_M(n) \le p(n)$. One can now formally define languages belonging to class **P** as follows:

P={L: if there is a polynomial-time DTM program M for which L=L_M}

To extend the formalism to problem classification, a decision *problem* Π (a problem with a yes/no solution) belongs to class **P** under the encoding scheme e if $L[\Pi, e] \in P$, that is, *if there is a polynomial-time DTM program that solves Π under encoding scheme e.*

On the surface, **NP** problems appear to be intractable exponential-time problems. For example, there is no known polynomial-time algorithm for the TRAVELING SALESMAN problem, which asks, given a set of cities, distances between the cities and a bound B, if there is a tour of all the cities having total length ≤B. Of course, combinatorics is the source of difficulty in these problems, the myriad combinations of intercity distances that increase exponentially with city number. What distinguishes **NP** problems from other exponential-time problems is that their solutions may be *quickly verified in polynomial-time*. While it is a simple matter to add up the intercity distances for an arbitrary ordering of cities in the TRAVELING SALESMAN problem, what makes the problem difficult is the huge number of city orderings that must be checked. Thus, it is the polynomial-time-verifiability exponential-time-solvability dichotomy that isolates **NP** problems as a distinct class. Computer scientists define **NP** in terms of what is called a *nondeterministic algorithm*. This algorithm consists of a guessing stage and a checking stage. For the TRAVELING SALESMAN problem, the guessing stage generates a random order of cities (hence the term *nondeterministic* algorithm) and the checking stage verifies the solution (yes/no) using normal deterministic computation. Nondeterministic computation is formalized in terms of the Nondeterministic Turing Machine model (NDTM), which is depicted schematically below:



The nondeterministic one-tape Turing machine

The NDTM has the same structure as the DTM except for the guessing module and the guess-write-only head, which work together to randomly generate a solution structure. Of course, a single NDTM will not compute a solution to an instance of the TRAVELING SALESMAN problem in polynomial-time (unless it produces a very lucky guess). To compute a solution in polynomial-time, it is necessary to employ an ensemble of NDTMs whose number is ≥ to the total number of potential solutions of the problem (hence the term *nondeterministic polynomial-time* [**NP**] computation). There is general agreement among computer scientists that the nondeterministic polynomial-time algorithm is merely an abstract device for understanding polynomial-time verifiability and is not a realistic algorithm because the requirement for massively parallel computation (an ensemble of NDTMs) is impractical. Of course, this attitude has changed since Adelman's pioneering use of DNA to solve problems in this complexity class (vide infra). In a very real sense, Adelman constructed[2] the first real-world NDTM from DNA, a biological nondeterministic computer; this may force computer scientists to reformulate their ideas about the nature of *intractability*.

The placement of class **NP** in the abstract problem universe described above requires insight into the relationship between **P** and **NP**. The question of whether **P=NP** or **P≠NP** has been extensively studied by computer scientists because of its implications for the relationship between problem *solvability* and solution *verifiability*. Problems in class **P**, for instance, are solvable in polynomial-time whereas **NP** problems are verifiable in polynomial-time. If **P=NP**, then devising an algorithm that solves an **NP** problem in polynomial-time should be possible. Considering that most computer encryption systems are based on hard **NP** problems (***NP**-complete*) whose solutions may be quickly verified (unlocked) in polynomial-time, developing an algorithm that *solves* the problem in polynomial-time would compromise the security of the encryption schemes the world over. Fortunately, the consensus among computer scientists is that **P≠NP** because, to date, despite considerable effort, no polynomial-time algorithms have been found for **NP** problems like the TRAVELING SALESMAN problem and its cousins. However, while there is a *strong* consensus that **P≠NP**, it has not been rigorously proven. The fact that it is safe to assume that **P≠NP** notwithstanding, the two problem classes share an interesting characteristic: any problem Π that can be solved by a deterministic polynomial-time algorithm A can also be solved by a nondeterministic algorithm B by using A for the checking stage of B and ignoring the guess. Thus $\Pi \in$ **P** implies $\Pi \in$ **NP**. So while **P≠NP**, because any problem in **P** can be solved by a nondeterministic algorithm, **P** must be a subset of **NP**, or stated formally **P⊆NP**.

2.3] Polynomial transformations and NP-completeness:

If **P≠NP** then discriminating between **P** and **NP-P** is vitally important because all problems in **P** can be solved by polynomial-time algorithms *whereas all problems in **NP-P** are intractable*. Complexity theorists employ a useful shortcut to facilitate proving that a problem $\Pi \in$ **NP-P**. Instead of showing that a new problem $\Pi' \in$ **NP-P**, Π' is *polynomially transformed* into an existing problem Π that is already known to be in **NP-P**. To examine this concept, we must return to the idea of languages L (remember that a language L is a set of input strings x that are all recognized by a DTM program M, i.e. a set of solutions). A *polynomial transformation* from language $L_1 \subseteq I_1^*$, belonging to problem Π_1, to language $L_2 \subseteq I_2^*$, belonging to problem Π_2, is a function f: $I_1^* \rightarrow I_2^*$ that satisfies the following two conditions:

(1) There is a polynomial-time DTM program that computes f
(2) For all $x \in I_1^*$, $x \in L_1$ if and only if $f(x) \in L_2$

In the parlance of complexity theory, if the language L_1, belonging to Π_1, can be polynomially transformed into language L_2, belonging to Π_2, then one writes $L_1 \propto L_2$; the mechanics of polynomial transformation comes from the following three lemmas:

(1) If $L_1 \propto L_2$ then $L_2 \in$ **P** implies $L_1 \in$ **P**, or $L_2 \notin$ **P** implies $L_1 \notin$ **P**
(2) If $L_1 \propto L_2$ and If $L_2 \propto L_3$ then $L_1 \propto L_3$
(3) If L_1 and L_2 belong to **NP**, L_1 is ***NP**-complete*, and $L_1 \propto L_2$, then L_2 is ***NP**-complete*

If Π_1 and Π_2 are decision problems, with associated encoding schemes e_1 and e_2, whenever there exists a polynomial transformation from $L[\Pi_1,e_1]$ to $L[\Pi_2,e_2]$ it is understood that $\Pi_1 \propto \Pi_2$. In short, a polynomial transformation translates one problem type, about which little is known, into second problem type, about which much is known. *In so doing, all the computational characteristics of the second problem can be applied to* *the first*. For instance, if Π_1 can be solved by a polynomial-time algorithm then so can Π_2, and if Π_1 is intractable then so is Π_2 (lemma-**1**), and furthermore, if $\Pi_1 \propto \Pi_2$ then Π_2 is "just as hard" as Π_1.

Polynomial transformations are also transitive (lemma-**2**), for instance, if $\Pi_1 \propto \Pi_2$ and if $\Pi_2 \propto \Pi_3$ then $\Pi_1 \propto \Pi_3$. Similarly, whenever both $L_1 \propto L_2$ and $L_2 \propto L_1$ (both $\Pi_1 \propto \Pi_2$ and $\Pi_2 \propto \Pi_1$), L_1 and L_2 (thus Π_1 and Π_2) are said to be *polynomially equivalent* and to belong to an *equivalent class*. Class **P**, for example, is an example of an equivalence class consisting of the computationally easiest languages (problems). The class ***NP**-complete* is also an example of a distinct equivalence class that contains the hardest languages (problems) in **NP**. Formally, a language L is ***NP**-complete* if $L \in$ **NP** and, for all other languages $L' \in$ **NP**, $L' \propto L$. Equivalently, a decision problem Π is ***NP**-complete* if $\Pi \in$ **NP** and, for all other decision problems $\Pi' \in$ **NP**, $\Pi' \propto \Pi$. Recalling the inference from lemma-1, if $\Pi_1 \propto \Pi_2$ then Π_2 is "just as hard" as Π_1, and that proving Π to be ***NP**-complete* requires showing that *every other problem* Π' in **NP** polynomially transforms into Π, leads to the conclusion that ***NP**-complete* problems are "just as hard" as any other problems in **NP** and, therefore, that ***NP**-complete* problems are the most difficult problems in **NP**. Thus, if any ***NP**-complete* problem can be solved with a polynomial-time algorithm A, then all **NP** problems can be solved with A. Furthermore, if any **NP** problem is intractable, then all ***NP**-complete* problems are intractable.

Clearly, showing that every problem Π' in **NP** can be transformed into an ***NP**-complete* problem candidate Π is not an easy task. Thus lemma-3 is of great value because given a single ***NP**-complete* problem Π_{NP}, one can prove any problem Π to be ***NP**-complete* merely by showing:

(1) $\Pi \in$ **NP**

(2) $\Pi \propto \Pi_{NP}$

2.4] Six basic NP-complete problems:

In a landmark 1971 paper, Stephen Cook proved[4] that the Satisfaction problem (SAT) is ***NP**-complete* (Cook's theorem), and therefore, *as hard or harder than any other problem in **NP***. Since then, hundreds of problems have been classified as ***NP**-complete* by showing that they can be polynomially transformed into SAT. Subsequent to Cook's discovery, Richard Karp compiled[5] a list of 21 ***NP**-complete* problems that he classified by polynomial transformation to SAT. Of these 21 problems, 6 have emerged as the core set of ***NP**-complete* problems with which most others are proven ***NP**-complete;* they are:

Six basic *NP-complete* problems.

(1) **3-Satisfiability (3SAT)**

INSTANCE: Given a set of clauses $C=\{c_1, c_2, \ldots c_m\}$ on a finite set U of variables such that $|c_i| = 3$ for $1 \le i \le m$.

QUESTION: Is there a truth assignment for U that satisfies all the clauses in C?

(2) **3-Dimensional Matching (3DM)**

INSTANCE: A set $M \subseteq W \times X \times Y$, where W, X and Y are disjoint sets having the same number q of elements.

QUESTION: Does M contain a matching, that is, a subset $M' \subseteq M$ such that $|M'| = q$ and no two elements of M' agree in any coordinate?

(3) **Vertex cover (VC)**

INSTANCE: A graph G=(V,E) and a positive integer $K \leq | V |$.

QUESTION: Is there a vertex cover of size K or less for G, that is, a subset $V' \subseteq V$ such that $| V' | \leq K$ and, for each edge $\{u,v\} \in E$, at least one of u and v belongs to V'?

(4) **Clique**

INSTANCE: A graph G=(V,E) and a positive integer $J \leq | V |$.

QUESTION: Does G contain a clique of size J or more, that is, a subset $V' \subseteq V$ such that $| V' | \geq J$ and every two vertices in V' are joined by an edge in E?

(5) **Hamiltonian circuit (HC)**

INSTANCE: A graph G=(V,E)

QUESTION: Does G contain a Hamiltonian circuit, that is, an ordering $\{v_1, v_2, \ldots v_n\}$ of the vertices of G, where $n=| V |$, such that $\{v_n, v_1\} \in E$ and $\{v_i, v_{i+1}\} \in E$ for all I, $1 \leq i < n$?

(6) **Partition**

INSTANCE: A finite set A and a "size" $s(a) \in Z^+$ for each $a \in A$.

QUESTION: Is there a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A-A'} s(a)$$

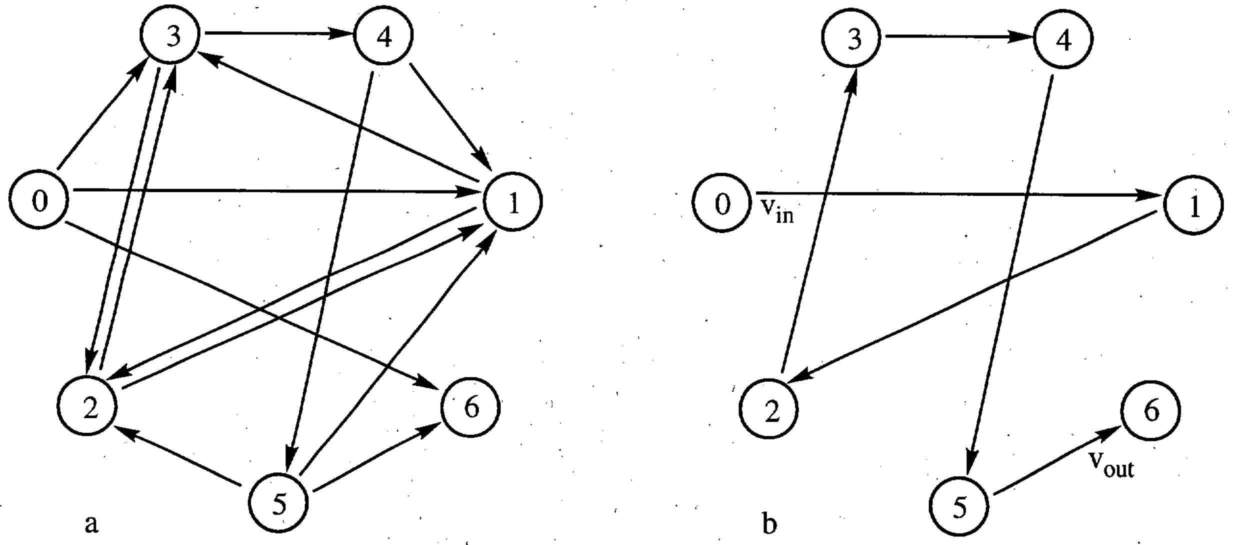
Since these 6 problems form an equivalence class, for any two Π_{3SAT} or Π_{HC}, for instance, $\Pi_{3SAT} \propto \Pi_{HC}$ and $\Pi_{HC} \propto \Pi_{3SAT}$, therefore, the problems are equivalent. Furthermore, if one devises a polynomial-time algorithm that solves HC then it can solve 3SAT, 3DM, VC, CLIQUE, PARTITION or any other ***NP**-complete* problem as well.

3] Adelman's algorithm:

In a seminal paper, Leonard Adelman described an algorithm[2] that uses DNA and the tools of molecular biology to solve an instance of HC. When considering the equivalence properties of the class ***NP**-complete*, the gravity of Adelman's spectacular accomplishment becomes clear, that is, his algorithm or other variants of it could be used to solve any ***NP**-complete* problem. Aldelman's ingenious approach is both simple and intuitive. He uses the massive parallelism accessible through the combinatorial ligation of single-stranded DNA to generate all possible solutions to an instance of HC. He then applies the tools of molecular biology to sort through the myriad DNA strands until a solution to HC is found. Amazingly, Adelman has effectively devised a practical *nondeterministic polynomial-time* algorithm, a program that conducts a *parallel* exhaustive search. Recall that until Adelman's disclosure, there was a consensus among computer scientists that *nondeterministic polynomial-time* algorithms are not practically realizable but are merely abstract tools with which to understand the concept of polynomial-time verifiability.

To understand Adelman's algorithm one must first grasp the intricacies of the Hamiltonian circuit or Hamiltonian path problem (HC). Given a directed graph G=(V,E) (illustrated on the following page) consisting of 7 vertices v_0-v_6 including the designated vertices v_{in} and v_{out}, and 14 paths, the graph is said to have a

Hamiltonian path if and only if there exists a sequence of compatible *one-way* edges (a continuous path) that begins at v_{in} and ends at v_{out}. Intuitively, it is not difficult to see that what makes this problem intractable is the branching of paths at each vertex. While the number of vertices and paths is small the number of combinations of vertices and paths through the graph is high and rises exponentially with the number of vertices and paths. Adelman's nondeterministic algorithm for solving this instance of HC is as follows:



a) Adelman's directed Hamiltonian graph G with v_{in}=0 and v_{out}=6
b) The graph G's unique Hamiltonian path, 0-1, 1-2, 2-3, 3-4, 4-5, 5-6.

- *Step 1:* Generate random paths through the graph.
- *Step 2:* Keep only those paths that begin with v_{in} and end with v_{out}.
- *Step 3:* If the graph has n vertices, then keep only those paths that enter exactly n vertices.
- *Step 4:* Keep only those paths that enter all of the vertices of the graph at least once.
- *Step 5:* If any paths remain say "yes"; otherwise say no.

To implement *step 1* of the algorithm, each vertex is represented by a discrete random 20mer oligonucleotide denoted O_i. Each edge O_i-O_j of the graph is created such that it is the 3'-10mer of its lower-order vertex and the 5'-10mer of its higher-order vertex. In the case of i=0 the edge is all O_i and i=6 it is all O_j. The Watson-Crick complement to O_i is denoted $\overline{O_i}$. All possible paths through the graph are created in a single combinatorial ligation step. An example of one of the ligations is illustrated below:

ligation

$O_{2\to3}$ ↓ $O_{3\to4}$

3'-*GTA TAT CCG AGC TAT TCG AG*C TTA AAG CTA GGC TAG GTA C-5'
5'-CG ATA AGC TCG AAT TTC GAT-3'

$\overline{O_3}$

To implement *step 2* of the algorithm, the product of step 1 is amplified by PCR using O_0 and $\overline{O_6}$ as primers, such that only strands beginning at vertex-0 and ending at vertex-6 are amplified.

To implement *step 3* of the algorithm, the product of *step 2* is run on an agarose gel and the 140 bp band excised. In this way only strands that encode 7 vertices are obtained.

To implement *step 4* of the algorithm, the product of *step 3* is purified using a biotin-streptavidin magnetic bead separation kit. First, beads bearing $\overline{O}_1$ are incubated with the *step 3* pool. After melting annealed DNA from the beads, the resulting pool contains sequences that encode vertex-1 at least once. This procedure is repeated with beads containing $\overline{O}_2$, $\overline{O}_3$, $\overline{O}_4$ and $\overline{O}_5$, using the DNA melted off the beads from the previous separation as the DNA-pool for a current separation.

To implement *step 5* of the algorithm, the DNA from *step 4* is amplified by graduated PCR. In this procedure, PCR is conducted using O_0 and $\overline{O}_6$ as primers, then O_0 and $\overline{O}_5$, O_0 and $\overline{O}_4$, O_0 and $\overline{O}_3$, O_0 and $\overline{O}_2$, and finally O_0 and $\overline{O}_1$. In this way, oligonucleotides of 140, 120, 100, 80, 60, and 40 bp are created that reveal the Hamiltonian path.

Concerning the generality of the algorithm, the number of operations increases linearly (not combinatorially) with the number of vertices and the number of oligonucleotides grows linearly with the number of edges (as expected for a polynomial-time algorithm). Concerning the power of this algorithm, current supercomputers execute 10^{12} operations per second. If one considers the ligation of two DNA strands in *step 1* an operation, then at the picomole scale approximately 10^{14} operations are executed. Certainly, much higher operation densities are possible at higher DNA concentrations.

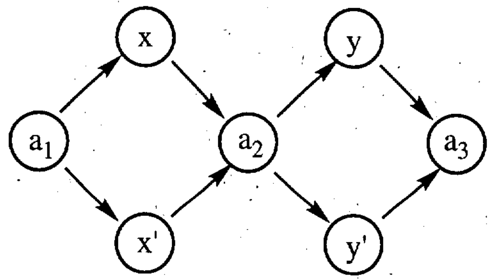
4] Lipton's algorithm:

Adelman was the first researcher to actually construct a molecular computer, but further accomplishments in experimental DNA computing have come from other worker's laboratories, including Ouyang and Bancroft. Lipton modified Adelman's approach, devising an algorithm that uses the massive parallelism of combinatorial DNA ligation to solve satisfaction problems (SAT)[6]. His contribution was to convert the SAT into a graph problem to which Adelman's algorithm could be applied. This process is analogous to the mechanism by which new problems are proven to be ***NP-complete***, that is, polynomial transformation of the new problem, about which little is known, into the old one, which is well-characterized.

The SAT is a variant of 3SAT that asks, given a set of clauses C={c_1, c_2,...c_m} on a finite set U of variables such that $|c_i|=2$ for $1 \le i \le m$, is there a truth assignment for U that satisfies all the clauses in C? The SAT Lipton examined was of the form:

$$F=(x \vee y) \wedge (\bar{x} \vee \bar{y})$$

In this equation there are two clauses, $(x \vee y)$ and $(\bar{x} \vee \bar{y})$. The variables x and y are Boolean and can assume values of 0 (false) and 1 (true). $\vee$ is the logical OR operation, which dictates that $(x \vee y)$=0 only if x=y=0. $\wedge$ is the logical AND operation, which dictates that $(x \wedge y)$=1 only if x=y=1. $\bar{x}$ is the negation of x, which dictates that $\bar{x}=0$ if $x=1$, and, $\bar{x}=1$ if $x=0$. The SAT problem is to assign values to x,y such that F=1 (F is true). Lipton's algorithm employs the same combinatorial DNA ligation step, conceived by Adelman, to simultaneously create all possible solutions to the SAT. The possible values of x and y are translated into a directed graph, illustrated on the following page, in which each vertex and path is encoded by DNA strands as in Adelman's algorithm. The ligation step creates all possible paths through the graph, encoding



$$\begin{tikzcd}
 & x \arrow[dr] & & y \arrow[dr] & \\
a_1 \arrow[ur] \arrow[dr] & & a_2 \arrow[ur] \arrow[dr] & & a_3 \\
 & x' \arrow[ur] & & y' \arrow[ur] &
\end{tikzcd}$$

{x,y}={00,01,10,11}, all possible combinations of x and y, which are contained in test tube-0. Next, a series of magnetic bead-based-extractions and combinations are conducted for $F = (x \vee y) \wedge (\bar{x} \vee \bar{y})$ as follows:

- *Step 1:* Extract x=1 from test tube-0.
- *Step 2:* Extract y=1 from test tube-0.
- *Step 3:* Combine extracts from steps 1 and 2 into test tube-1
- *Step 4:* Extract x=0 from test tube-1
- *Step 5:* Extract y=0 from test tube-1
- *Step 6*: Combine extracts from steps 4 and 5 into test tube-2

Any DNA that remains in test tube-2 encodes the solution to the SAT.
In the general case, given an equation F={$C_1 \wedge C_2 \wedge ... C_n$}, begin with C_1 and conduct an extraction step for each literal [x] ($x_1 \vee x_2 \vee ... x_n$) in C_1. For each x, extract x=1 and for each $\bar{x}$, extract x=0. When the extractions for clause C_1 are complete, combine the extracts in an empty test tube and repeat the extraction sequence for C_2 on the contents of the test tube.
Although not supported by experimental data, Lipton's algorithm is noteworthy because it solves SAT, the hardest problem in NP. Moreover, the SAT is the only problem in NP that has been rigorously proven to be NP-complete.

5] The Condon-Smith algorithm:
Condon, Smith and coworkers[7] recently proposed an algorithm that uses ca. 10^{12} unique oligonucleotides (not spatially arrayed) immobilized on a 1cm^2 glass surface to solve NP-complete problems. Using operations based on *hybridization, polymerase extension, exonuclease degradation* and *ligation*, all oligonucleotides that do not encode solutions to a problem instance are destroyed; any DNA that remains encodes the solution. This is analogous to burning the haystack to find the needle.
The Condon-Smith algorithm proceeds as follows:

- *Step-1:* Parallel synthesis is used to prepare a combinatorial library of 10^{12} discrete oligonucleotides, which is immobilized onto a 1cm^2 activated glass surface via 5'- linkers.
- *Step-2:* A set of complementary DNA strands that satisfies some constraint of the problem is synthesized and *hybridized* to its complements arrayed on the glass surface. This is referred to as the **Mark** operation.

- *Step-3:* If indexed, multiple words are contained on each immobilized oligo, a set of complementary primers is annealed to the complementary, immobilized primers and *polymerase extension* is used to extend the primers to the 3'-ends of the immobilized templates. At the end of steps 2 and 3 all strands that partially or fully satisfy the constraints of the problem are **Marked** as double stranded.
- *Step-4:* All unmarked, single-stranded, immobilized oligonucleotides are destroyed via *exonuclease degradation* using exonuclease-I. This is referred to as the **Destroy-Unmarked** operation.
- *Step-5:* Any immobilized DNA that remains encodes the solution to the problem.

The advantages of the Condon-Smith algorithm include:
- Manipulation of DNA on surfaces is more efficient (minimal losses) and amenable to automation.
- A means is provided to develop the techniques of molecular computation on a small scale to pave the way for future iterations to be conducted on a larger scale, in solution.

The disadvantages of the Condon-Smith algorithm include:
- The scale of computation (10^{12} strands) is small.
- Spatial addressability is lost.
- A read-out operation to determine the solution, once found, is absent.

6] Comparison of the Adelman, Lipton and Condon-Smith algorithms: Putting things in perspective:
Combinatorial chemists are intuitively aware of the relationship between the *size* or *diversity* of a library and the chances of isolating a lead compound from the library. Thus, the power of a library is often judged to be a function of the number of discrete molecules it contains. Similarly, the power of a nondeterministic polynomial-time algorithm is a function of the extent to which it surveys the solution space of a problem instance. Lipton has pointed out that cracking the DES may be the first real-world application of molecular computation[8]. However, combinatorial chemistry, which is a form of exhaustive search that may be rightly described as *applied* molecular computation, should be recognized as the first real-world application of molecular computation.
Considering the manifest similarity between molecular computation and combinatorial chemistry, it is reasonable to surmise that they will follow similar evolutionary paths. For instance, key issues in the evolution of combinatorial chemistry have included the development of efficient encoding and deconvolution strategies, the creation of more effective surface and solution chemistries and the expansion of library size. Correspondingly, key concerns in molecular computation development will likely focus on efficient methods of encoding oligonucleotides, solid-phase versus solution-phase implementations, and expanding the limits of solution-space. In a recent review, Gordon[9] *et al.* organized the objectives of discovery chemistry into three general categories, fine tuning, chemical analoguing, and lead identification, and presented combinatorial approaches to dealing with each category: serial medicinal chemistry, spatially addressable libraries, and encoded/noncoded synthetic libraries, respectively. In progressing from lead identification to fine tuning, the requirement for diversity decreases, hence, in moving from recombinant peptide libraries to encoded/noncoded synthetic libraries to spatially addressable libraries, and finally, to serial medicinal chemistry, the library size decreases from $\geq 10^{12}$ to 10^0. While the encoding/deconvolution strategies and the chemistries of the various combinatorial approaches

differ, the crucial issue of library size is common to all. Thus, library size limitations have dictated the objective (fine tuning, chemical analoguing, or lead identification) to which each method is commonly applied. Library size will unquestionably play a central role in molecular computation as well.

The Hamiltonian Path graph solved using **Adelman's algorithm**[2] contains 7 vertices and 14 paths. The time complexity function of the Hamiltonian path problem is *n*!, where *n* is the number of vertices in the graph. Therefore, Adelman's HP-graph requires at most 7! or 5040 discrete DNA strands to compute a solution. Adelman's algorithm is conducted in solution and therefore is limited by the physical size of an aqueous solution containing the DNA. Lipton's proposed DES attack[8], for example, requires 2^{56} or ca. 10^{17} strands. This amount of DNA could fit into 1L of water and translates to an HP-graph containing 19 vertices. How far could the Hamiltonian path problem be taken with existing combinatorial chemistry technology?

First, consider **VLSIPS technology.**[10] Currently, with 10μm synthesis sites, 10^7 different oligonucleotides can be spatially arrayed on a 4 × 4 cm glass slide. Future versions of the technology may incorporate smaller synthesis sites. If a means of generating 0.5 μm synthesis sites becomes available, then 6.4×10^9 distinct oligonucleotides could be spatially arrayed in the same area. A spatially arrayed library of 6.4×10^9 discrete oligonucleotides could solve an HP-graph containing ≤13 vertices ($13!=6.2 \times 10^9$).

The **Lipton algorithm**[6] is conducted in solution. Though unsupported by experimental data, Lipton's algorithm is limited by the physical size of the DNA pool required. Therefore, the size of the DNA pool required to crack the DES - 10^{17} strands in 1L of water - which translates to a 19-vertex HP-graph, gives a good indication of the inherent limitations.

The **Condon-Smith algorithm**[7] proposes using ca. 10^{12} unique oligonucleotides (not spatially arrayed) on a glass slide to solve NP-complete problems. Using operations based on *hybridization, polymerase extension, exonuclease degradation* and *ligation*, all oligonucleotides that do not encode solutions to a problem instance are destroyed; any DNA that remains encodes the solution. Condon and Smith's library of 10^{12} oligonucleotides could solve an HP-graph containing ≤15 vertices ($15!=1.3 \times 10^{12}$).

Next, consider libraries immobilized on **Polymeric Beads**. Typically 10μm-diameter polystyrene beads, which give rise to 5×10^9 beads/g with a maximum ligand loading of ~20 fmol/bead, are used to create libraries. If a library is created on 1g of polystyrene beads, it will contain $(5 \times 10^9) \times (20 \times 10^{-15}) \times (6 \times 10^{23})$ or 6×10^{19} immobilized oligonucleotides. A library of 6×10^{19} discrete oligonucleotides could solve an HP-graph containing ≤21 vertices ($21!=5.1 \times 10^{19}$). To place this number in perspective, bear in mind that the DES contains 2^{56} possible solutions, which could be searched with ~10^{17} oligonucleotides.

Considering that a plot of variable number versus time complexity function for an exponential-time problem is open ended, one can easily focus on the limits of molecular computation by setting the variable number so high that an unreasonable amount of DNA is required to compute a solution. For instance, Hartmanis[11] recently asserted that if Adelman's HP-graph is scaled to 200 vertices, the amount of DNA required to compute the Hamiltonian path would exceed the mass of the earth. One who engages in such an exercise misses the point. The discoveries of DNA-based computing research will not likely solve *all* exponential-time problems, but instead, will continually redefine the limits of *demonstrable* computation and lead to the development of new paradigms of computation.

7] The Lipton-Connors algorithm:

As Condon and Smith pointed out[7], while the use of immobilized oligonucleotide libraries imposes limitations on the size of NP-complete problem that may be solved, the ease of working on the solid-phase will enable researchers to refine techniques that will be applied to solution-based libraries in future implementations. Curiously, however, Condon and Smith have chosen to work with libraries randomly immobilized on glass slides (10^{12} maximum library size) instead of polystyrene beads (10^{20} maximum library size). They reason that working on the solid phase carries significant advantages in terms of ease of manipulation and automation and that dropping spatial addressability increases the scale at which they can work. This strategy is questionable because it is unreasonable to sacrifice spatial addressability for a small gain in scale (i.e. a 13-vertex HP-graph versus a 15-vertex HP-graph), especially considering that Condon and Smith have not incorporated a read-out strategy into their algorithm. Furthermore, the use of a spatially-addressed library with Condon and Smith's algorithm would give *instant access* to the solution strand after all others are destroyed. It is possible that they based their choice on the fact that a DNA-computer-on-a-chip would be better received by the computer science community, who are accustomed to the idea of photolithographic chip manufacture, than a test tube full of beads.

The Lipton-Connors algorithm is a variant of the Lipton algorithm[6] in which the DNA strands formerly in solution are spatially arrayed on a two-dimensional glass surface. Lipton proposes using hybridization to complementary oligonucleotides immobilized on magnetic beads to extract oligonucleotides encoding potential solutions from the pool containing all possible DNA sequences. This approach is limited by hybridization fidelity, which may introduce error in the last step of the algorithm. Furthermore, using hybridization requires oligonucleotides of at least 15 bases in length for each variable state of the SAT in order to minimize the errors introduced by hybridization infidelity.

The Lipton-Connors algorithm uses a polymerase-mediated single-base extension (APEX) to discriminate between the myriad DNA strands in the pool. Using APEX offers two advantages over hybridization:

- The DNA sequence encoding a variable can be reduced from 15 bases to a minimum of a single base.
- The process of discrimination is enhanced by the fidelity of the polymerase during single-base extension.

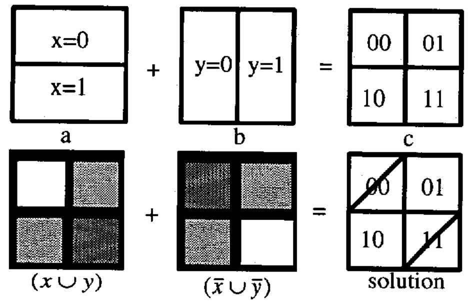
The Lipton-Connors algorithm is best understood by describing how it solves a 2-variable SAT of the form:

$$(x \vee y) \wedge (\bar{x} \vee \bar{y}) = F$$

In this equation there are two clauses, $(x \vee y)$ and $(\bar{x} \vee \bar{y})$. The variables x and y are Boolean and therefore can only assume values of 0 (false) and 1 (true). $\vee$ is the logical OR operation, which dictates that $(x \vee y)$=0 only if x=y=0. $\wedge$ is the logical AND operation, which dictates that $(x \wedge y)$=1 only if x=y=1. $\bar{x}$ is the negation of x, which dictates that $\bar{x} = 0$ if $x = 1$, and, $\bar{x} = 1$ if $x = 0$. The SAT problem is to assign values to x,y such that F=1 (F is true).

- *Step-1:* Each possible value {0,1} of x and y is represented by a discrete oligonucleotide. The value of the variable (x or y) is encoded by the location of its representative oligonucleotide on a two-dimensional glass surface using a binary masking scheme. Possible DNA strands representing x=0, x=1, y=0, y=1 follow:

5'-modifier-C6-Sp-CGC GAG GTC GCA CGG CTC AGA AAA **A**	x=0
5'-modifier-C6-Sp-CGC GAG GTC GCA CGG CTC AGA AAA **T**	x=1
5'-modifier-C6-Sp-CGC GAG GTC GCA CGG CTC AGA AAA **G**	y=0
5'-modifier-C6-Sp-CGC GAG GTC GCA CGG CTC AGA AAA **C**	y=1



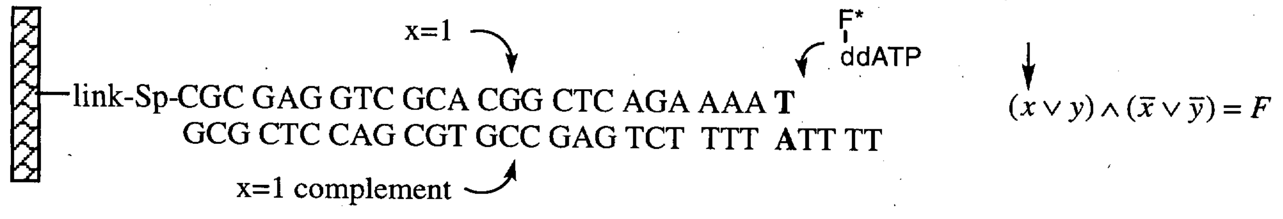
Step-1 Prepare a spatially addressable array of DNA primers (c) encoding all possible solutions to the SAT.

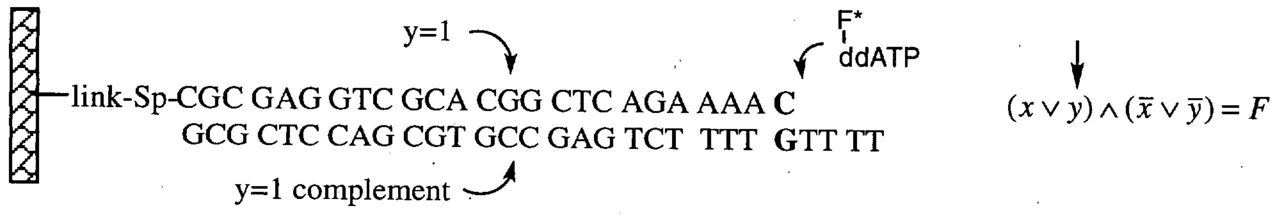
Step-2 For each bracketed expression in the SAT conduct a primer extension reaction with a dye-labeled nucleotide terminator and DNA complements encoding 1 for normal variables (ie:x) and 0 for negations (ie:$\bar{x}$).

Step-3 Dye-free surfaces contain DNA sequences that do not satisfy the SAT and are eliminated. The remaining strands encode the solution to the SAT.

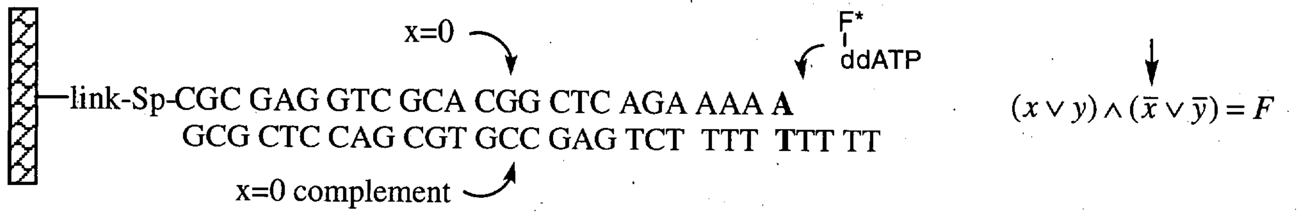
As depicted schematically above, the oligo solution encoding *x=0* is spotted on the upper half of the slide while a solution of the *x=1* oligo is immobilized on the lower half in the normal binary-mask fashion. The slide is then rotated 90° and the process is repeated with the *y=0* and *y=1* oligo solutions. Since spotting will be used instead of photolithographic synthesis, oligo solutions of *x=0* and *y=0* can be mixed together and immobilized in the NW quadrant. Similarly, mixtures of oligonucleotides *x=0+y=1*, *x=1+y=0* and *x=1+y=1* are spotted in the NE, SW and SE quadrants respectively.

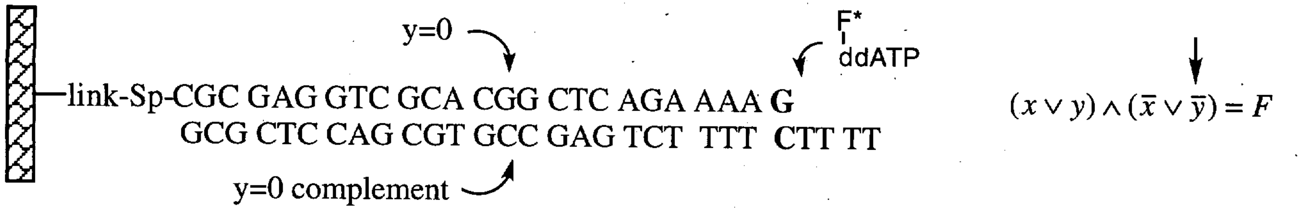
- *Step-2:* For each bracketed expression (clause) in the SAT equation, a primer extension reaction is conducted with fluorescein-labeled ddATP using DNA complements encoding 1 for normal variables (i.e. the complement of x=1 for x) and 0 for negations (i.e. the complement of x=0 for $\bar{x}$). Thus, for the first clause of the equation, the following two primer extension reactions would be concurrently executed:


x=1
F*
ddATP
link-Sp-CGC GAG GTC GCA CGG CTC AGA AAA **T**
GCG CTC CAG CGT GCC GAG TCT TTT **A**TT TT
x=1 complement
$(x \vee y) \wedge (\bar{x} \vee \bar{y}) = F$


y=1
F*
ddATP
link-Sp-CGC GAG GTC GCA CGG CTC AGA AAA **C**
GCG CTC CAG CGT GCC GAG TCT TTT **G**TT TT
y=1 complement
$(x \vee y) \wedge (\bar{x} \vee \bar{y}) = F$

This procedure will produce a slide in which all regions that satisfy clause-1 (x∨y) are colored (see diagram). Both primer and template oligonucleotides depicted above have two distinct regions, a 20-base constant region and a 5-base variable region. Taking the primers as an example, the 20-base region extending from the spacer (Sp) to the five As is constant for all the immobilized primers and serves to insure that efficient hybridization occurs between the templates and primers under the conditions of the extension. The 5-base region appended to the constant sequence is a variable segment that partially encodes the value of its variable. The variable region only partially encodes the variable value because the balance of this information is encoded in the primer's location on the slide. The variable region is located at the 3'-end because this is where the polymerase is most sensitive to mismatches. For the second clause of the equation, the following two primer extensions would be concurrently executed:


x=0
F*
ddATP
link-Sp-CGC GAG GTC GCA CGG CTC AGA AAA A
GCG CTC CAG CGT GCC GAG TCT TTT TTT TT
x=0 complement
$(x \vee y) \wedge (\bar{x} \vee \bar{y}) = F$


y=0
F*
ddATP
link-Sp-CGC GAG GTC GCA CGG CTC AGA AAA G
GCG CTC CAG CGT GCC GAG TCT TTT CTT TT
y=0 complement
$(x \vee y) \wedge (\bar{x} \vee \bar{y}) = F$

This procedure will produce a slide in which all regions that satisfy clause-2 $(\bar{x} \vee \bar{y})$ are colored.

- *Step-3:* The dye-free regions from each round of primer extension are eliminated, leaving the solution(s) to the SAT.

The Lipton-Connors algorithm shares some of the same advantages and disadvantages that apply to the Condon-Smith algorithm, namely:

Advantages:

- *Both algorithms:* Manipulation of DNA on surfaces is more efficient (minimizing losses) and amenable to automation.
- *Both algorithms:* Provide a means to develop the techniques of molecular computation on a small scale to pave the way for future iterations to be conducted on a larger scale, in solution.
- *The Lipton-Connors algorithm:* Spatial addressability offers *instant access* to the solution.

Disadvantages:

- *Both algorithms:* The maximum scale of computation is small:
  Condon-Smith=10^{12} strands (≤15 vertex-HP-graph).
  Lipton-Connors=10^{9} strands (≤13 vertex-HP-graph).
- *The Condon-Smith algorithm:* Spatial addressability is lost. There is no read-out operation to determine the solution, once found.

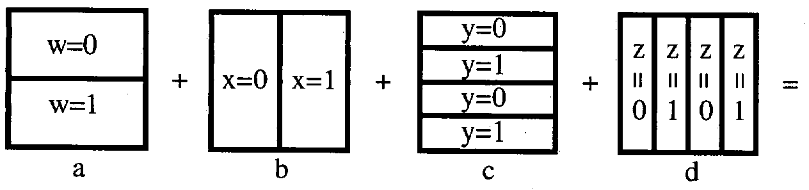
Clearly, neither algorithm threatens the world's cryptographic systems. However, they do offer insight into novel paradigms for computation that might evolve into more powerful approaches in future iterations.

The following diagram illustrates how the Lipton-Connors algorithm can be applied to solve larger SAT problems:

Problem: $(w \cup x) \cap (\overline{w} \cup \overline{x}) \cap (\overline{w} \cup x) \cap (\overline{x} \cup \overline{y}) \cap (y \cup \overline{z}) = 1$

Solution:

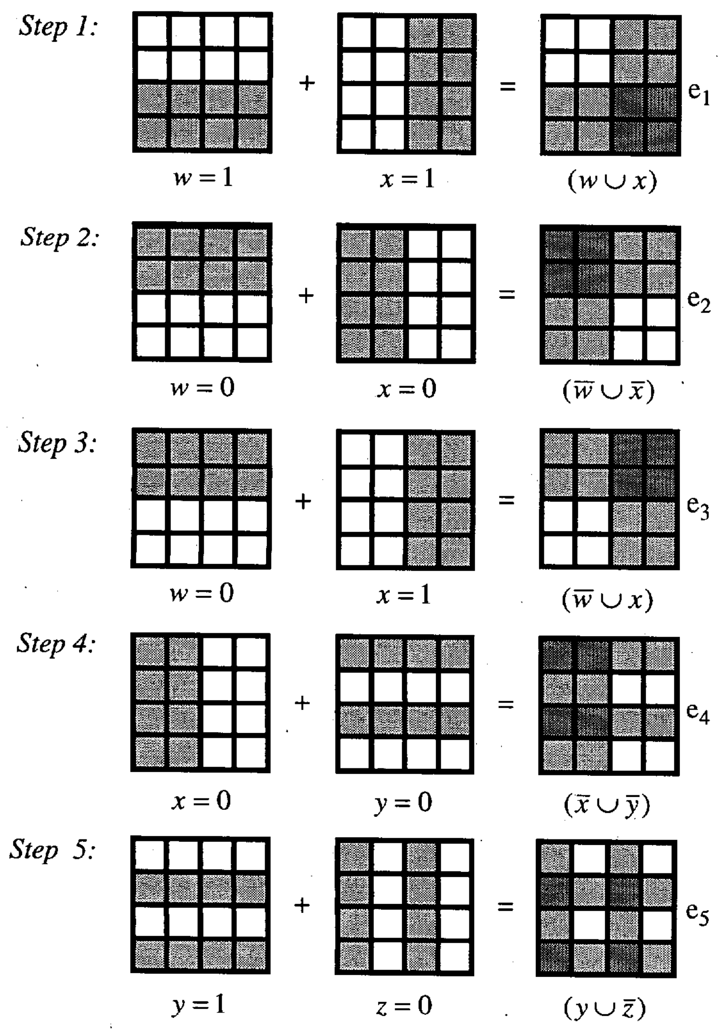
1) Let the possible values (0,1) for each of w,x,y and z be represented by a discrete oligonucleotide. Attach the oligonucleotides to a glass slide as depicted below. Five identical matrices (e), which could be arrayed on the same slide, will be required to solve the equation.



0000	0001	0100	0101
0010	0011	0110	0111
1000	1001	1100	1101
1010	1011	1110	1111

e

2) Each matrix (e_i) will solve one of the bracketed clauses. For each variable (ie:w) in a bracketed expression, treat the appropriate matrix with the DNA complement encoding 1 (ie: w=1 complement). For each negation (ie:$\overline{w}$), treat with the complement encoding 0 (ie: w=0 complement).The expected results of each of the five primer extension reactions are illustrated below:



Surfaces of the slide which are dye-free contain DNA sequences which do not satisfy the equation. These "null" sequences are removed from contention.

For example, the results of steps one and two indicate that the four DNA sequences in the upper left quadrant and the four sequences in the lower right quadrant do not satisfy the equation.

Similarly, steps three to five eliminate the DNA sequences in the lower left quadrant and three of the four squares in the upper right quadrant. Striking all the null sequences yields the solution to the equation (w,x,y,z)=(0,1,0,0) as depicted below:

~~0000~~	~~0001~~	0100	~~0101~~
~~0010~~	~~0011~~	~~0110~~	~~0111~~
~~1000~~	~~1001~~	~~1100~~	~~1101~~
~~1010~~	~~1011~~	~~1110~~	~~1111~~

The 4-bit SAT depicted above contains four binary variables: w, x, y, and z. Therefore, eight 5'-modified oligonucleotides similar in structure to those used in the 2-bit computation are required:

5'- modifier-C6-S18-CG CGA GGT CGC ACG GCT CAG AAA **AA**	$w = 0$
5'- modifier-C6-S18-CG CGA GGT CGC ACG GCT CAG AAA **AT**	$w = 1$
5'- modifier-C6-S18-CG CGA GGT CGC ACG GCT CAG AAA **AG**	$x = 0$
5'- modifier-C6-S18-CG CGA GGT CGC ACG GCT CAG AAA **AC**	$x = 1$
5'- modifier-C6-S18-CG CGA GGT CGC ACG GCT CAG AAA **TA**	$y = 0$
5'- modifier-C6-S18-CG CGA GGT CGC ACG GCT CAG AAA **TT**	$y = 1$
5'- modifier-C6-S18-CG CGA GGT CGC ACG GCT CAG AAA **TG**	$z = 0$
5'- modifier-C6-S18-CG CGA GGT CGC ACG GCT CAG AAA **TC**	$z = 1$

These oligonucleotides will act as immobilized primers. Eight template oligos complementary to the oligos listed above will also be required:

5'-TTT **TTT** TTT CTG AGC CGT GCG ACC TCG CG-3'	$\overline{w} = 0$
5'-TTT **TAT** TTT CTG AGC CGT GCG ACC TCG CG-3'	$\overline{w} = 1$
5'-TTT **TCT** TTT CTG AGC CGT GCG ACC TCG CG-3'	$\overline{x} = 0$
5'-TTT **TGT** TTT CTG AGC CGT GCG ACC TCG CG-3'	$\overline{x} = 1$
5'-TTT **TTA** TTT CTG AGC CGT GCG ACC TCG CG-3'	$\overline{y} = 0$
5'-TTT **TAA** TTT CTG AGC CGT GCG ACC TCG CG-3'	$\overline{y} = 1$
5'-TTT **TCA** TTT CTG AGC CGT GCG ACC TCG CG-3'	$\overline{z} = 0$
5'-TTT **TGA** TTT CTG AGC CGT GCG ACC TCG CG-3'	$\overline{z} = 1$

While sixteen oligos may seem like a large number to solve such a problem, what is important is how the number grows as the problem size increases. To state the relationships explicitly, as the number of variables (n) in the SAT increases, the number of oligos needed increases by 4n while the number of possible solutions to the SAT increases by 2^n.

References:

1] National Bureau of Standards, “Data Encryption Standard,” US Department of Commerce, FIPS, pub. 46, January, 1977.

2] Adelman, L. M. “Molecular computation of solutions to combinatorial problems.” *Science* **1994**, *266*, 1021.

3] Garey, M. R.; Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness* (Freeman, San Francisco, 1979).

4] Cook, S. “The Complexity of Theorem Proving Procedures.” *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing.* (Association of Computing Machinery, New York, 1971).

5] Karp, R.M. “Reducability among combinatorial problems” in Miller, R.E.; Thatcher, J.W. *Complexity in Computer Computations*, Plenum Press, New York 1972. p.85-103.

6] Lipton, R. J. “DNA solution of hard computational problems.” *Science* **1995**, *268*, 542.

7] Cai, W.; Condon, A.E.; Corn, R.M.; Glaser, E.; Fei, Z.; Frutos, T.; Guo, Z.; Lagally, M.G.; Liu, Q.; Smith, L.M.; Thiel, A. “The Power of Surface-Based Computation.” in *Proceedings of the Second Annual Meeting on DNA-Based Computers*, Princeton, NJ, June 1996 (American Mathematical Society, Providence, RI).

8] Boneh, D.; Dunworth, C.; Lipton, R. J. “Breaking DES using a molecular computer.” in *Proceedings of the Second Annual Meeting on DNA-Based Computers*, Princeton, NJ, June 1996 (American Mathematical Society, Providence, RI).

9] Gallop, M.A.; Barrett, R.W.; Dower, W.J.; Fodor, S.P.A.; Gordon, E.M. "Applications of combinatorial technologies to drug discovery. 1. Background and peptide combinatorial libraries." *J. Med. Chem.* **1994**, *37*, 1233.

10] a) Fodor, S. P. A.; Read, J. L.; Pirrung, M. C.; Stryer, L.; Liu, A. T.; Solas, D. “Light-directed spatially-addressable parallel chemical synthesis.” *Science* **1991**, *251*, 767. b) Pease, A. C.; Solas, D.; Sullivan, E. J.; Cronin, M. T.; Holms, C. P.; Fodor, S. P. A. “Light-generated oligonucleotide arrays for rapid DNA sequence analysis.” *Proc. Natl. Acad. Sci. U.S.A.* 1994, *91*, 5022. c) Broude, N.E.; Sano, T.; Smith, C.L.; Cantor, C.R. “Enhanced DNA sequencing by hybridization.” *Proc. Natl. Acad. Sci. U.S.A.* **1994**, **91**, 3072. d) Schumaker, J. M.; Metspalu, A.; Caskey, C. T. “Mutation detection by solid-phase primer extension.” *Hum. Mutat.* **1996**, *7*, 346.

11] Hartmanis, J. “On the weight of computations.” *Bull. Eur. Assoc. Theor. Comp. Sci.* **1995**, *55*, 136.

12] Frutos, A.G.; Liu, Q.; Thiel, A.J.; Sanner, A.M.W.; Condon, A.E.; Smith, L.M.; Corn, R.M. "Demonstration of a word design strategy for DNA computing on surfaces." *Nucleic Acids Res.* **1997**, *25*. 4748.

13] Stimpson, D.E.; Hoijer, J.V.; Hsein, W.T.; Jou, C.; Gordon, J.; Theriault, T.; Gumble, R.; Baldeschwieler, J.D. " Real-time detection of DNA hybridization and melting on oligonucleotide arrays by using optical wave guides." *Proc. Natl. Acad. Sci. USA* **1995**, *92*, 6379.