Molecular Computing, Bounded Nondeterminism, and Efficient Recursion*

Richard Beigel and Bin Fu
Lehigh University

April 13, 1998

Abstract

The maximum number of strands used is an important measure of a molecular algorithm's complexity. This measure is also called the *volume* used by the algorithm. Every problem that can be solved by an NP Turing machine with $b(n)$ binary nondeterministic choices can be solved by molecular computation in a polynomial number of steps, with four test tubes, in volume $2^{b(n)}$. We identify a large class of recursive algorithms that can be implemented using bounded nondeterminism. This yields improved molecular algorithms for important problems like 3-SAT, independent set, and 3-colorability.

1. A model of molecular computing

Molecular computation was first studied in [1, 20]. The models we define were inspired as well by the work of [3, 28]. A molecular sequence is a string over an alphabet Σ (we can use any alphabet we like, encoding characters of Σ by finite sequences of base pairs). A test tube is a multiset of molecular sequences. We describe the allowable operations below. Where set notation is applied to multisets, multiplicities are respected. In the definitions T_1, T_2, and T_3 denote distinct test tubes, c denotes a character, and i denotes a positive integer.

Separate(T_1, c, i, T_2, T_3)
$T_2 :=$ the multiset of all strings in T_1 whose ith character is c;
$T_3 :=$ the multiset of all strings in T_1 whose ith character is not c;
$T_1 := \emptyset$.

*Research performed at Yale University and at the University of Maryland. Supported in part by the National Science Foundation under grant CCR-8958528, CCR-9415410, and CCR-9700417. The first author was also supported in part by NASA under grant NAG 52895. Address: Dept. of Electrical Engineering and Computer Science, 19 Memorial Dr W Ste 2, Bethlehem PA 18015-3084. Email: {beigel bif3}@eecs.lehigh.edu. Web: http://www.eecs.lehigh.edu/~{beigel bif3}.

Append(T, c)
$T := \{xc : x \in T\}$.

Merge(T_1, T_2, T_3)
$T_3 := T_1 \cup T_2$;
$T_1 := \emptyset$;
$T_2 := \emptyset$.

Others have proposed a variant of operation separate, which we will call Sep. It checks whether a string contains the character c anywhere. If we represent the ith symbol z_i of a string z by the symbol $\langle i, z_i \rangle$ instead, then the standard Sep operation can simulate our Separate operation with no additional overhead. The use of polynomial-size alphabets is standard practice in molecular computing. We prefer the Separate operation for convenience in programming.

The running time for a molecular algorithm is proportional to the number of operations on test tubes. An important complexity measure is the size of the solution space (also called simply *volume*), which is the maximum number of strings in all test tubes at any time, counting multiplicities. Adleman [2] has speculated that molecular computation with a solution space of size 2^{70} (about 0.002 moles) might be possible. Recent papers [3, 23] attempt to optimize solution volume size for particular combinatorial problems.

Problem instances are associated with a parameter n called their size. In complexity theory, n is the length of a suitable encoding of the instance. However, in analysis of algorithms, n is usually a more natural representation-independent parameter, such as the number of vertices in a graph or number of variables in a formula. Although the n's of complexity theory and the n's of analysis of algorithms are usually polynomially related, it can make a phenomenal difference when n appears in the exponent. For that reason we take n to be a problem-dependent but representation-independent notion of size through this paper. We write $|x|$ to denote the *size* of a problem instance x rather than its length, and we usually identify n with $|x|$.

We consider a highly restricted model of $t(n)$-time, $v(n)$-volume molecular computation, which we think has a good chance of eventually being practical. On input x, one test tube T_0 is initialized to hold encodings of the numbers $1, \ldots, v(|x|)$. A sequence of molecular operations $o_1, \ldots, o_{t(|x|)} = f(x)$ is then performed, where f is a conventional polynomial-time computable function (that is, the program is uniform in a weak but appropriate sense). The computation accepts if T_0 is nonempty after the last operation is performed.

Definition 1.

- $\mathrm{MOL}^*(t(n), v(n))$ is the class of languages accepted by such a computation where the running time is $O(t(n))$.
- $\mathrm{MOL}(v(n)) = \bigcup_{k=1}^{\infty} \mathrm{MOL}^*(n^k + k, v(n))$.

Results / Problem	Previously			In This Paper	
	Volume	Limited Model	Reference	Volume	Limited Model
Hamiltonian Path	$n!$	$\surd$	[1]		
SAT	2^n	$\surd$	[20]		
QBF	2^n	$\times\times$	[4, 27]		
3-SAT	1.62^n	$\times$	[23]	1.50^n	$\surd$
3-Colorability	1.89^n	$\surd$	[3]	1.35^n	$\surd$
Independent Set	1.51^n	$\surd$	[3]	1.23^n	$\surd$
$(3,2)$-system				1.39^n	$\surd$

Table 1: Results for particular problems. Under "Limited model" $\surd$ means that the model is similar to ours, $\times$ means that the model allows the Amplify operation, and $\times\times$ means that the model allows Intersection.

We give the most volume-efficient molecular algorithms known for several problems. See Table 1. In this table we compare our results with previous results for some important NP-complete problems.

2. Biological Background for the Operations

DNA is the storage medium for genetic information. It is found in every cellular organism. It is composed of units called nucleotides distinguished by the chemical group attached to them. The four bases are adenine, guanine, cytosine, and thymine, abbreviated as A, G, C and T. Single nucleotides are linked together end-to-end to form DNA strands. Every DNA strand has two distinct ends, one with a free $5'$ PO_4 group and the other with a free $3'$ OH group, called $5'$ and $3'$ ends, respectively. Thus, we can think of the sequence of nucleotides in any strand as having a natural orientation.

Mutual attraction, called hydrogen bonding, exists between As and Ts and between Gs and Cs. Hence, A/T and G/C are called complementary base pairs. For two complementary single strands, hydrogen bonding will only occur if the pair of complementary strands are oriented in an anti-parallel fashion. That is, as one strand of a double helix runs in the $5' \rightarrow 3'$ direction, its complement runs in the $3' \rightarrow 5'$ direction. Two complementary single strands can pair up and twist around each other, forming the famous double helix structure discovered by Watson and Crick. As DNA strands consist of four different symbols A, G, C, T, they can encode conventional strings in the conventional computer.

2.1. Encoding

Let $x = x_1 \cdots x_n$ be an n-bit binary string. Assign a unique sequence of $\ell = 2\lceil \log_3 2m \rceil + 1$ bases to each bit position and bit value. Sequence $B_i(0)$ encodes the ith bit of x

as 0. Sequence $B_i(1)$ encodes the ith bit of x as 1. $x = x_1 \cdots x_n$ is represented by the following DNA sequence:

$$B_1(x_1)B_2(x_2)\cdots B_n(x_n).$$

We require that for any $\langle i, a\rangle \neq \langle j, b\rangle$, $B_i(a)$ and $B_j(b)$ have no long common subsequence.

2.2. DNA Manipulations

In the last decade there have been revolutionary advances in the field of biomedical engineering particularly in recombinant DNA and RNA manipulating. Basic principles about recombinant DNA can be found in [37, 38]. In this section we describe some biological operations that are useful for molecular computation. Some of our descriptions are similar to those in [8, 15, 25].

For a string x we denote by $\uparrow x$ the single DNA strand whose sequence is x and whose orientation is $5' \to 3'$, as x is read from left to right. Similarly, $\downarrow x$ denotes the strand complementary to $\uparrow x$. Finally, $\updownarrow x$ denotes the double strand that results when $\uparrow x$ and $\downarrow x$ anneal.

2.2.1. Merge

This is to pour the solution of two test tubes into a third one to achieve union. It can be performed by re-hydrating the tube contents and then combining the fluids together into a new tube by pouring.

2.2.2. Append

This is to elongate every strand in a test tube T by tacking another short strand z onto the end. Suppose $z = D_1D_2\cdots D_k$, where each $D_i \in \{A, T, G, C\}$. A solution containing D_1 is poured into T, and each sequence d reacts with D_1 to form dD_1. After washing the excess D_1 solution away, one could have D_1 from the chain dD_1 coupled with D_2 to form dD_1D_2, and so on. Each character is appended in the $3'$ end (It will be not appended at $5'$-end). Each time only one character is appended. This can be guaranteed by the biological techniques that let the new character be in protected state. The protected state can be changed into unprotected state so that another character can be appended.

2.2.3. Separation by Subsequence

This is to extract all strands containing the sequence $\uparrow x$. We use the method of biotin–avidin purification as described in [1]. We first create many copies of $\uparrow x$'s complementary oligo (a short DNA strand), namely $\downarrow x$. To these oligos we attach biotin molecules, which are in turn anchored to an avidin bead matrix. We then melt the double strands in our test tube and pour them over this matrix. Those strings that contain $\uparrow x$ anneal to the $\downarrow x$ oligos anchored to the matrix. We wash away all

strands that did not anneal, leaving behind only those strands that contain $\uparrow x$, which can then be retrieved from the matrix.

2.2.4. Melting

This is to break double DNA strands into its single-stranded complementary components by heating the solution.

2.2.5. Separation by Length

This is to separate all DNA sequences by their lengths. A technique called gel electrophoresis is used. The molecules are placed at the top of a wet gel, to which an electric field is applied, drawing them to bottom. Larger molecules travel more slowly through. After a period, the molecules spread out into distinct bands according to size.

2.2.6. Amplification (PCR)

This copies DNA strands by using the polymerase chain reaction (PCR). If we have the duplex $\updownarrow xyz$, we first melt it to form $\uparrow xyz$ and $\downarrow xyz$. To this solution we add the primer oligos $\downarrow z$ and $\uparrow x$, which anneal to form the partial duplexes $\uparrow xy\updownarrow z$ and $\updownarrow x\downarrow yz$. DNA polymerase can then elongate the primer to create full duplexes of the form $\updownarrow xyz$. Now we have two copies of our original strand. Thus, if we can guarantee that the primer sequences that we use occur on the ends of every strand, and only on the ends, then we can use PCR to duplicate every strand in the test tube.

2.2.7. Detect

This is to test if there is at least one DNA sequence in the solution of a test tube. It is usually done by amplification (see [30]).

3. Bounded Nondeterminism

NP computation with a limited amount of nondeterminism was introduced in [16, 17, 18] and studied further in [11, 12, 24, 10, 13, 31, 14, 7]. The class NPbits($b(n)$) consists of all languages recognized by an NP machine that make at most $b(n)$ binary nondeterministic choices on each computation path on inputs of size n. (Actually, prior treatments allowed $O(b(n))$ binary choices, but the constant factor turns out to be very important in connection with molecular computation.) We define a refinement of these classes: NPinit($v(n)$) consists of all languages recognized by NP machines that nondeterministically choose a number between 1 and $v(n)$ on inputs of size n and then behave deterministically. Clearly, NPbits($b(n)$) = NPinit($2^{b(n)}$). In order to make this paper self-contained, we give an equivalent definition of NPinit().

Definition 2.

- For $n > 0$, define $\mathrm{pad}_n(x) : \{0,1\}^{\leq n} \to \{0,1\}^{2n}$ by

$$\mathrm{pad}_n(x_1 x_2 \cdots x_k) = 0x_1 0x_2 \cdots 0x_k 1^{2n-2k}.$$

- A family of circuits $\{C_n\}_{n=1}^{\infty}$ is P-uniform if there is a polynomial time computable function f such that $f(0^n) = C_n$ for all n.

- A function f is P-uniform $s(n)$-size computable if there is a P-uniform family of circuits $\{C_n\}_{n=1}^{\infty}$ such that each C_n is of size at most $s(n)$ and

$$|x| \leq n \Rightarrow C_n(\mathrm{pad}_n(x)) = f(x).$$

- $L \in \mathrm{NPinit}(s(n), v(n))$ if there is a P-uniform family of circuits $\{C_n\}_{n=1}^{\infty}$ such that each C_n is of size at most $s(n)$ and

$$x \in L \iff (\exists i)[1 \leq i \leq v(n) \text{ and } C_{|x|}(x, i) = 1].$$

- $\mathrm{NPinit}(v(n)) = \bigcup_{k=1}^{\infty} \mathrm{NPinit}(n^k + k, v(n))$.

4. $\mathrm{NPinit}(v(n)) \subseteq \mathrm{MOL}(v(n))$

In this section we show how to simulate bounded nondeterministic computation via bounded-volume molecular computation. Results of this type appear in [4, 28, 29, 35], but they assume models of molecular computation with more powerful operations, such as Amplify, that may be harder to implement in practice. The following result is due to Boneh et al. [9]. In order to keep our paper self-contained, we give the proof here.

Lemma 3. *Let π be a circuit with m gates. Given a tube T_0, a molecular algorithm using only the operations Separate, Append, and Merge, running in time $O(m)$, and using only four test tubes can create tubes T_1 and T_2 such that T_1 contains all strings z from tube T_0 that satisfy $\pi(z) = 1$ and T_2 contains all strings z from tube T_0 that satisfy $\pi(z) = 0$.*

Proof: Let π's input gates be $g_1, \ldots, g_n$ and internal gates be $g_{n+1}, \ldots, g_m$ in topological order; in particular g_m is the output gate. We will use four tubes T_0, T_1, T_2, T_3. For each i, let g_i compute $f_i(g_{j(i)}, g_{k(i)})$ where $j(i) < i$, $k(i) < i$, and f_i is a binary function. We perform the algorithm shown in Figure 1. At completion, T_1 contains all strings that satisfy π and T_2 contains all strings that do not satisfy π. ∎

Theorem 4. $\mathrm{NPinit}(s(n), v(n)) \subseteq \mathrm{MOL}^*(s(n), v(n))$.

for $i := n+1$ to m do
    Separate$(T_0, 0, j(i), T_1, T_2)$
    Separate$(T_1, 0, k(i), T_0, T_3)$
    Append$(T_0, f_i(0,0))$
    Append$(T_3, f_i(0,1))$
    Merge(T_0, T_3, T_1)
    Separate$(T_2, 0, k(i), T_0, T_3)$
    Append$(T_0, f_i(1,0))$
    Append$(T_3, f_i(1,1))$
    Merge(T_0, T_3, T_2)
    Merge(T_1, T_2, T_0)
Separate$(T_0, 1, m, T_1, T_2)$

Figure 1: Simulating a circuit π via molecular computation (cf. [9]). We assume that the ith gate, g_i, computes $f_i(g_{j(i)}, g_{k(i)})$ where $j(i) < i$, $k(i) < i$, and f_i is a binary function. At completion, T_1 contains all strings that satisfy π and T_2 contains all strings that do not satisfy π.

Proof: Let $L \in \text{NPinit}(s(n), v(n))$ via a P-uniform family of circuits $\{C_n\}_{n=1}^{\infty}$ of size $s(n)$ such that each C_n is of size at most $s(n)$, and $x \in L$ iff $C_{|x|}(x,i) = 1$ for some i with $1 \leq i \leq v(|x|)$. For each x, fix circuit $C'_x(i) = C_{|x|}(x,i)$. Applying Lemma 3, we get a molecular algorithm that performs $O(s(|x|))$ operations to determine whether there exists i such that $1 \leq i \leq v(|x|)$ and $C'_x(i) = 1$. Since $\{C_n\}_{n=1}^{\infty}$ is P-uniform, the molecular operations sequences can be also computed in polynomial time. ∎

Corollary 5. $\text{NPinit}(v(n)) \subseteq \text{MOL}(v(n))$.

5. Implementing Recursion with Bounded Nondeterminism

In this section we show how to enumerate search volume using bounded nondeterminism. In many nondeterministic searches, some paths are longer than others, which can be inefficient. However, if we can compute the size of subtrees, then we can balance nondeterministic search trees, which reduces the amount of nondeterminism needed.

Meyer and Paterson [21] formalized the notion of self-reductions. Recursive algorithms for NP problems often take the form of disjunctive-self-reductions [21, 34] (henceforth, simply "dsr"). We will show how to convert $T(n)$-time recursive algorithms of this type to $T(n)$-volume, polynomial-time molecular algorithms.

In an earlier version of this paper [6], we favored tight exposition over tight time bounds. In order to firm up the time bounds in this paper, we have introduced four parameters d, b, s, w to describe d-self-reductions. The reader who is satisfied with arbitrary polynomial-time bounds may safely ignore these parameters, replacing them all by arbitrary polynomials.

Definition 6. A partial order $\prec$ is $(d(n), b(n))$-well-founded if

- $y_k \prec \cdots \prec y_1 \Rightarrow k \leq d(|y_1|)$; and
- $y_k \prec \cdots \prec y_1 \Rightarrow |y_k| \leq b(|y_1|)$

For technical simplicity we will consider only languages L containing the empty string, Λ.

Definition 7.

- A $(d(n), b(n), s(n), w(n))$-dsr for a language L consists of a P-uniform $s(n)$-size computable function $h(x) = \{x_1, \ldots, x_m\}$ and a $(d(n), b(n))$-well-founded partial order $\prec$ on problem instances such that
  - Λ is the only minimal element under $\prec$
  - for all $x \neq \Lambda$, $x \in L \iff h(x) \cap L \neq \emptyset$
  - for all x, $x_i \in h(x) \Rightarrow x_i \prec x$.
  - $||h(y)|| \leq w(|x|)$ for any node y in H_x, where H_x is the least set such that $x \in H_x$ and $(y \in H_x \Rightarrow h(y) \subseteq H_x)$.
- A dsr is a $(d(n), b(n), s(n), w(n))$-dsr for some polynomials $d(n), b(n), s(n), w(n)$.

Definition 8. Let $\langle h, \prec \rangle$ be a $(d(n), b(n), s(n), w(n))$-dsr and let x be a problem instance.

- $T_{h,\prec}(x)$ is the unordered rooted tree that satisfies the following rules: (1) the root is x; (2) for each y, the set of children of y is $h(y)$.
- $|T_{h,\prec}(x)|$ is the number of leaves in $T_{h,\prec}(x)$.

If $\langle h, \prec \rangle$ is a dsr for L, then the corresponding recursive algorithm for L runs in time $|T_{h,\prec}(x)|$. The analysis of such an algorithm usually provides a bound on $|T_{h,\prec}(x)|$ that is suitable for use in constructing a molecular algorithm for L. We formalize this below:

Definition 9.

- A language L is in $\mathrm{REC}^*(d(n), b(n), s(n), w(n), T(x))$ if there is a $(d(n), b(n), s(n), w(n))$-dsr $\langle h, \prec \rangle$ for L such that for all x
  (1) $|T_{h,\prec}(x)| \leq T(x)$, and
  (2) $T(x) \geq \sum_{x_i \in h(x)} T(x_i)$.
- $\mathrm{REC}(T(x)) = \bigcup_{k=1}^{\infty} \mathrm{REC}^*(n^k + k, n^k + k, n^k + k, T(x))$.

Lest conditions (1) and (2) above seem restrictive, we argue that they are quite natural. We consider the typical analysis of a recursive algorithm. One introduces a function T and proves by induction on x that $|T_{h,\prec}(x)| \leq T(x)$, which is (1). The inductive hypothesis is that $|T_{h,\prec}(w)| \leq T(w)$ if $w \prec x$. Inspection of the algorithm yields

$$\begin{aligned} |T_{h,\prec}(x)| &= \sum_{x_i \in h(x)} T_{h,\prec}(x_i) \\ &\leq \sum_{x_i \in h(x)} T(x_i) \qquad \text{by the inductive hypothesis} \end{aligned}$$

The last step in the induction consists of showing that T satisfies $\sum_{x_i \in h(x)} T(x_i) \leq T(x)$, which is (2).

The function T above depends on problem instances rather than their size because the analysis of the algorithm may depend on two or more parameters. We will need an analogous variant of NPinit.

Definition 10.

- $L \in \text{NPinit}'(s(n), V(x))$ if there is a P-uniform family of circuits $\{C_n\}_{n=1}^{\infty}$ such that each C_n is of size at most $s(n)$ and

$$x \in L \iff (\exists i)[1 \leq i \leq V(x) \text{ and } C_{|x|}(x, i) = 1].$$

- $\text{NPinit}'(V(x)) = \bigcup_{k=1}^{\infty} \text{NPinit}'(n^k + k, V(x))$.

Clearly, if $V(x) \leq v(|x|)$ then $\text{NPinit}'(s(n), V(x)) \subseteq \text{NPinit}(s(n), v(n))$.

Theorem 11. *If $T(x)$ is P-uniform $s'(n)$-size computable, then*

$$\text{REC}^*(d(n), b(n), s(n), w(n), T(x)) \subseteq \text{NPinit}'(t'(n), T(x)),$$

where $t'(n) = O(d(n) \cdot (s(b(n)) + w(n) \cdot s'(b(n))))$.

Proof: Let $L \in \text{REC}^*(d(n), b(n), s(n), w(n), T(x))$ via $\langle h, \prec \rangle$. We will define a function $\text{path}(i, x)$ taking values in $\{0, 1, \Lambda\}$ such that $\text{path}(1, x) \cdots \text{path}(T(x), x)$ is equal to the sequence of values at the leaves of $T_{h,\prec}(x)$ in canonical order. The function $\text{path}(i, x)$ will be computed via tail recursion.

```
function path(i, x)
if x = Λ then return true
else if h(x) = ∅ then return false
else
    {x_1, ..., x_m} := h(x)
    for j := 1 to m do
        if i ≤ T(x_j) then return path(i, x_j)
        else i := i − T(x_j)
```

return Λ

Since h is P-uniform $s(n)$-size computable, there is a P-uniform family of circuits $\{C_{h,n}\}_{n=1}^{\infty}$ such that $C_{h,n}$ is of size at most $s(n)$, and $h(x) = C_{h,n}(\mathrm{pad}_n(x))$ for all x of length at most n. We have a circuit $C_1 = C_{h,b(n)}$ of size at most $s(b(n))$ such that $C_1(\mathrm{pad}_{b(n)}(y)) = h(y)$ for all y with length at most $b(n)$.

Since T is P-uniform $s'(n)$-size computable, there is a P-uniform family of circuits $\{C_{T,n}\}_{n=1}^{\infty}$ such that $C_{T,n}$ is of size $s'(n)$, and $T(x) = C_{T,n}(\mathrm{pad}_n(x))$ for all x of length at most n. So, we have a circuit $C_2 = C_{T,b(n)}$ of size at most $s'(b(n))$ such that $C_2(\mathrm{pad}_{b(n)}(y)) = T(y)$ for all y of length at most $b(n)$.

We are going to construct a circuit to compute path(x, k). $T_{h,\prec}(x)$ is a tree of depth no more than $d(x)$. The circuit, with input $\langle x, k\rangle$, is divided into $d(|x|)$ layers $L_0, L_1, \ldots, L_{d(|x|)}$. Layer L_0 outputs $\langle x_0, k_0\rangle = \langle x, k\rangle$. Suppose we have constructed layer L_i whose output is $\langle x_i, k_i\rangle$. Layer L_{i+1} will consist of three subcircuits, which we call $L_{i+1,1}, L_{i+1,2}$ and $L_{i+1,3}$.

- $L_{i+1,1}$ gets $\langle x_i, k_i\rangle$ from L_i, and computes $h(x_i) = \{y_1, \ldots, y_m\}$.
- $L_{i+1,2}$ gets $\{y_1, \ldots, y_m\}$ from $L_{i+1,1}$, and computes $T(y_1), \ldots, T(y_m)$.
- $L_{i+1,3}$ gets k_i, $\{y_1, \ldots, y_m\}$, and $T(y_1), \ldots, T(y_m)$ from L_i, $L_{i+1,1}$, and $L_{i+1,2}$ respectively. It computes $\langle x_{i+1}, k_{i+1}\rangle = \langle y_t, k_i - (T(y_1) + \cdots + T(y_{t-1}))\rangle$ such that $T(y_1) + \cdots + T(y_{t-1}) < k_i \leq T(y_1) + \cdots + T(y_t)$.

Clearly, $L_{i+1,1}$ has size $O(s(b(|x|)))$, $L_{i+1,2}$ has size $O(w(|x|) \cdot s'(b(|x|)))$, and $L_{i+1,3}$ has size $O(w(|x|) \cdot s'(b(|x|)))$. Therefore, the size of L_i is $O(s(b(|x|)) + w(|x|) \cdot s'(b(|x|)))$. Hence, the function path can be computed by a P-uniform family of circuits with size $O(d(|x|) \cdot (s(b(|x|)) + w(|x|) \cdot s'(b(|x|))))$. ∎

Corollary 12. $\mathrm{REC}(T(x)) \subseteq \mathrm{NPinit}(T(x))$.

6. Two Simple Applications

6.1. Monien and Speckenmeyer's Algorithm for 3-SAT

In this section, we apply our results to the classic 3-SAT algorithm of Monien and Speckenmeyer [22]. This yields a simple $\mathrm{MOL}^*(n \cdot |f|, 1.62^n)$ algorithm for 3-SAT, where n is the number of variables and $|f|$ is the number of clauses in formula f. The 3-SAT problem is to determine whether a set of clauses, each consisting of at most three literals, is satisfiable. We will present the 3-SAT algorithm of Monien and Speckenmeyer. Let $f|_\ell$ denote the formula obtained from f by replacing the literal ℓ by true and the literal $\overline{\ell}$ by false. A k-clause is a disjunction of exactly k literals.

function 3SAT(f)
if f is the empty set of clauses then return true

else if f contains an empty clause then return false
else if some variable v appears only in positive literals then return $3\text{SAT}(f|_v)$
else if some variable v appears only in negative literals then return $3\text{SAT}(f|_{\overline{v}})$
else if f contains a clause C consisting of a single literal ℓ then return $3SAT(f|_\ell)$
else if f contains a clause C consisting of two literals ℓ_1, ℓ_2 then
    return $3\text{SAT}(f|_{\ell_1}) \vee 3\text{SAT}(f|_{\overline{\ell_1}}|_{\ell_2})$
else
    let v be the first variable to appear in f
    return $3\text{SAT}(f|_v) \vee 3\text{SAT}(f|_{\overline{v}})$

The last case in the recursion is ostensibly the worst, yielding two subproblems of size $n-1$, but it only occurs on the first call or immediately after eliminating a single variable, which yields a single subproblem of size $n-1$; unrolling the recursion, we see that the last case gives two subproblems of size $n-2$. The worst case is the second to the last, which yields subproblems of size $n-1$ and $n-2$. Thus the number of leaves in the self-reduction is at most $2f(n)$ where $f(n)$ is given by the recurrence $f(n) = f(n-1) + f(n-2)$; in particular $2f(n) \leq 1.62^n$ for almost all n.

We will show the algorithm above is a $(O(n), O(|f|), O(|f|), 2)$-dsr for 3-SAT. The value function h for a formula is the set of subformulas generated by the recursive algorithm. Let $m(x) = n$, where n is the number of variables in the formula x. Define $f_1 \prec f_2$ if $m(f_1) < m(f_2)$. From the analysis above we know if $h(x) = \{x_1, \ldots, x_k\}$, then $k = 1, 2$ and $\{\text{m}(x_1), \ldots, m(x_k)\}$ is either $\{n-1\}$, $\{n-2, n-2\}$ or $\{n-2, n-1\}$. Let $t(n) = 1.62^n$. $2f(n) \leq t(n)$. Hence $t(m(x))$ is an upper bound of the number of leaves of computation tree for the recursive algorithm. It is easy to see that $t(n) \geq t(n-2) + t(n-1) \geq t(n-2) + t(n-2)$. The function h is P-uniform $O(|f|)$-size computable. The partial order is $(O(n), O(|f|))$-well-founded. Hence, $h, \prec$, and $T(x) = t(m(x))$ satisfy the conditions of Definition 8 and Definition 9. By Theorem 4 and Theorem 11, 3-SAT is in $\text{MOL}^*(n \cdot |f|, 1.62^n)$.

Ogihara [23] implements Monien and Speckenmeyer's algorithm via breadth-first search, matching the volume bound above, but taking time $O(n \cdot \max(|f|^2, n))$. He also needs more powerful operations like Polymerization, which creates the complement of each strand in a test tube.

6.1.1. Tarjan's Algorithm

An independent set of nodes in a graph is a set of nodes, no two of which are adjacent. A famous NP-complete problem, the *independent set problem* is the following: Given a graph G and a positive integer k, is there an independent set in G consisting of at least k vertices?

Tarjan [36] gave a simple algorithm to find the largest independent set in a graph. We present Tarjan's algorithm in Figure 2. A simple modification yields a dsr for the independent set problem. We present the corresponding dsr in Figure 3.

In our complexity bounds, n denotes the number of vertices in G, e denotes the number of edges, and $|G|$ denotes $\max(e, n)$. Define $G_1 \prec G_2$ if the number of vertices in G_1 is less than the number of vertices in G_2. We claim that $\langle h, \prec \rangle$ is

function MIS(G)
pick any vertex v in G
if $d(v) \leq 1$ then
    return $\{v\} \cup \text{MIS}(G - v - N(v))$
else
    return $\max(\text{MIS}(G - v), \{v\} \cup \text{MIS}(G - v - N(v)))$

Figure 2: Tarjan's algorithm to find a maximum independent set. Herein, $d(v)$ denotes the degree of v, $N(v)$ denotes the neighbor set of v, and $\max(A, B)$ denotes the larger of the two sets A and B, with ties resolved arbitrarily.

function $h(G, k)$
pick any vertex v in G
if $d(v) \leq 1$ then
    return $\{(G - v - N(v), k - 1)\}$
else
    return $\{(G - v, k),\ (G - v - N(v), k - 1)\}$

Figure 3: A dsr corresponding to Tarjan's algorithm. Herein, $d(v)$ denotes the degree of v and $N(v)$ denotes the neighbor set of v.

a $(O(n), O(|G|), O(|G|), 2)$-dsr with at most $t(n)$ leaves, where $t(n)$ satisfies $t(n) = t(n-1) + t(n-3)$. Let $T(G) = t(m(G))$ where $m(G)$ is the number of vertices in G. Clearly, h, $\prec$, and $T()$ satisfy the conditions of Definition 8. The function h is P-uniform $O(|G|)$-size computable. The partial order $\prec$ is $(O(n), O(|G|))$-well-founded. The recurrence can be solved explicitly to obtain $t(n) < 1.47^n$. Thus, we have shown that the independent set problem is in $\text{MOL}^*(|G| \cdot n, 1.47^n)$, which improves a result of [3].

7. More Complicated Applications

In this section we implement more complicated recursive algorithms for several NP-complete problems. Because our results depend on conventional algorithms from other papers, this section is not self-contained.

In order to streamline the exposition, we will not give tight time bounds for the polynomial-time algorithms in this section. Rather our emphasis will be on minimizing the volume used in polynomial-time algorithms. We begin by giving sufficient conditions for T to be polynomial-time computable.

Definition 13. We say that a partial order $\prec$ on problem instances is *parameterizable* if there are a function m from problem instances to a set M, a partial order $\prec'$ on M, and a polynomial p such that

- $m(x)$ is computable in time polynomial in $|x|$,
- $x \prec y \Rightarrow m(x) \prec' m(y)$, and
- $||\{i : i \in M \text{ and } i \prec' m(x)\}|| \leq p(|x|)$.

In many examples we will take $m(x) = |x|$ and $\prec'$ to be the standard linear order on natural numbers. In other examples, $m(x)$ will be a tuple of parameters (such as the number of 2-clauses and the number of 3-clauses in a Boolean formula); in many (but not all) of these examples we use the partial order $(a_1, \ldots, a_k) \prec' (b_1, \ldots, b_k)$ if $(\forall i)[a_i \leq b_i]$ and $(\exists i)[a_i < b_i]$.

Definition 14. Given h and m, define

- $\mathrm{mh}(x) =$ the multiset $\{m(x_i) : x_i \in h(x)\}$
- $\mathrm{MH}(x) =$ the set $\{\mathrm{mh}(y) : m(y) = m(x)\}$

Definition 15. A dsr $\langle h, \prec \rangle$ is *by cases* if $\prec$ is parameterizable via $\langle m, \prec' \rangle$ in such a way that $\mathrm{MH}(x)$ is computable in time polynomial in $|x|$.

Lemma 16. *Let $\langle h, \prec \rangle$ be a dsr by cases with parameter function $m()$. Let T_0 be the least function T such that*

(1) $|T_{h,\prec}(x)| \leq T(x)$

(2) $T(x) \geq \sum_{x_i \in h(x)} T(x_i)$

(3) $T(x)$ *is a function of* $m(x)$

Then T_0 exists and $T_0(x)$ is computable in time polynomial in $|x|$.

Proof: Let $\langle h, \prec \rangle$ have a parameterization $\langle m, \prec' \rangle$, where $m(x)$ and $\mathrm{MH}(x)$ are computable in time polynomial in $|x|$. Define a partial function t from M to natural numbers recursively:

$$t(\mu) = \begin{cases} 1 & \text{if } \mu \text{ is a minimal element under } \prec' \\ \max_{m(y)=\mu} \sum_{y_i \in h(y)} t(m(y_i)) & \text{otherwise} \end{cases}$$

If μ is in the range of m, then $t(\mu)$ is defined because $\mathrm{MH}(x)$ is a finite set for every x. Now it is easy to see that $t \circ m$ is the least function satisfying (1,2,3).

By Definition 13, $|\{i : i \prec' m(x)\}| \leq p(|x|)$. If we compute $t(m(x))$ by the obvious recursion, at most $p(|x|)$ different subproblems will arise. If we use a table to avoid recomputation, the recursion will run in polynomial time. ∎

7.1. 3-SAT

Applying our methods to the algorithm of Monien and Speckenmeyer [22] yields a simple MOL(1.62^n) algorithm for 3-SAT. Monien and Speckenmeyer's result has been improved several times [32, 19, 33]. Each of those papers gives a dsr, so our methods are applicable to all of them.

Schiermeyer [33] claims a 1.497^n time algorithm for 3-SAT. His algorithm is a dsr for 3-SAT. We will prove that 3SAT $\in$ REC($T(F)$), where the function $T(F) \leq 1.497^n$ and will be defined below. We follow [33] to define F_3 and F_3'. For a formula F with n variables, let p be the maximum number of 1-clauses and 2-clauses (with preference of 1-clauses) such that no variable occurs more than twice. Let q be the number of remaining 2-clauses and define $m = p + \min(2, q)$. Let $F_3(n) = c\beta^n \cdot \frac{\beta}{\alpha^4}$ and $F_3'(n) = c\beta\alpha^{-m}$, where $\beta = 1.4963, \alpha = 1.04855$ and c is a sufficiently large constant.

$$T(F) = \begin{cases} F_3(n) & \text{if } F \text{ has no 1-clauses or 2-clauses} \\ F_3'(n) & \text{otherwise} \end{cases}$$

Schiermeyer states that $F_3'(n) \geq |T_{h,\prec}(F)|$ if F has at least one 1-clause or 2 clause, and that $F_3(n) \geq |T_{h,\prec}(F)|$ for all F. Hence, $T(F) \geq |T_{h,\prec}|$, since $|T_{h,\prec}| \leq$ the number of recursive calls. The inequalities that Schiermeyer gives in the proofs of his Lemma 4.3 and Lemma 4.4 imply that our $T(F)$ satisfies the conditions of Definition 9. Hence, 3SAT $\in$ REC($T(F)$) $\subseteq$ MOL(1.497^n).

7.2. 3-Coloring and $(3, 2)$-System

Beigel and Eppstein [5] give algorithms for $(3, 2)$-system and 3-coloring. In the (a, b)-system problem, we are given a collection of n vertices, each of which can be given one of a different colors. However certain color combinations are disallowed: we are also given a set of constraints, each of which forbids one coloring of some b-tuple of variables. $(3, 2)$-system generalizes 3-coloring, 3-SAT and 3-edge-coloring.

7.2.1. $(3, 2)$-System Algorithm

The size of a $(3, 2)$ system is the number of variables in it. Beigel and Eppstein's [5] $(3, 2)$-system algorithm can be sketched as follows:

function 32SYS(F)
if $|F| \leq 5$ then return brute-force(F)
else
    $\langle F_1, \ldots, F_k \rangle = h_{32}(F)$
    return $\bigvee_{i=1}^{m}$ 32SYS(F_i)

In the algorithm above, brute-force(F) means "use the brute force method to solve the $(3, 2)$-system F;" $k \leq 3$; h_{32} is polynomial-time computable; and $|F_i| < |F|$, where $|F|$ is the number of vertices in F. Let $h = h_{32}$ and let $\prec$ be the standard linear ordering on the natural numbers. Then $\langle h, \prec \rangle$ is a dsr for $(3, 2)$-system. Define $m(F) = |F|$.

In case 1, $\mathrm{mh}(F) = \{n-(4+i), n-1\}$, where $i \geq 0$.
In cases 2a, 2c, and 3, $\mathrm{mh}(F) = \{n-(3+i), n-2\}$, where $i \geq 0$.
In cases 2b, 2d, 6, 8c and 9, $\mathrm{mh}(F) = \{n'\}$, where $n' < n$.
In case 4, $\mathrm{mh}(F) = \{n-5, n-3, n-3\}$.
In case 5, $\mathrm{mh}(F) = \{n-4, n-4\}$.
In case 7, $\mathrm{mh}(F) = \{n-3, n-3\}$.

$\mathrm{MH}(F)$ is polynomial-time computable by the case analysis above. Let $t(n) = 1.38028^n$. It is easy to see that for every input x with $\{n_1, \ldots, n_k\} = \mathrm{mh}(x)$ and $m(F) = n$, $t(n) \geq t(n_1) + \cdots + t(n_k)$. Define $T(F) = t(m(F))$. By Lemma 16, there is a polynomial-time computable function T_0 such that $T_0(F) \leq T(F)$ and T_0, h, and $\prec$ satisfy the conditions of Definition 9. Thus, $(3,2)$-system is in $\mathrm{REC}(T_0(F))$. So,

$$\begin{aligned}
(3,2)\text{-system} &\in \mathrm{NPinit}'(T_0(F)) && \text{by Theorem 11} \\
&\subseteq \mathrm{NPinit}'(T(F)) && \text{because } T_0(F) \leq T(F) \\
&= \mathrm{NPinit}(t(n)) && \text{because } T(F) = t(|F|) \\
&= \mathrm{NPinit}(1.38028^n) && \text{because } t(n) = 138028^n \\
&\subseteq \mathrm{MOL}(1.38028^n) && \text{by Corollary 5.}
\end{aligned}$$

7.2.2. 3-Coloring Algorithm

There are two parts to Beigel and Eppstein's algorithm. The first part runs in polynomial-time and finds an independent set S with a lot of neighbors. Let $\Gamma(S)$ denote the set of vertices in G that are not in S but are adjacent to an element of S. The second part 3-colors S in all possible ways. Each of these $3^{|S|}$ partially-colored graphs is transformed into an equivalent $(3,2)$-system with $n - |S| - |\Gamma(S)|$ variables, which is solved by calling 32SYS. Their algorithm runs in time $3^{|S|}1.38028^{n-|S|-|\Gamma(S)|}$, which is less than 1.345^n for sufficiently large n. Thus we have the following $\mathrm{NPinit}(1.345^n)$ algorithm:

> choose a natural number $m < 1.345^n$
> construct Beigel and Eppstein's set S
> let $c = m \bmod 3^{|S|}$
> color S with the cth 3-coloring in the lexicographical ordering
> form the corresponding $(3,2)$-system F
> let $b = \lfloor m/3^{|S|} \rfloor$
> run $\mathrm{32SYS}(F)$ using the nondeterministic choices dictated by b

Therefore 3-coloring is in $\mathrm{MOL}(1.345^n)$.

function MAXIND(F)
$\langle G, S, a\rangle := F$
if $|G| \leq 5$ then
    return brute-force(F)
else
    $\langle F_1, \ldots, F_m\rangle := h_0(F)$
    $\langle b_1, \ldots, b_m\rangle := g(F)$
    return $\max(b_1 + \text{MAXIND}(F_1), \ldots, b_k + \text{MAXIND}(F_m))$

Figure 4: A sketch of Robson's purely recursive DTIME (1.229^n) algorithm for independent set.

7.3. Independent Set

7.3.1. Robson's Algorithm

The best published purely recursive algorithm for the independent set problem is due to Robson [26] and runs in time 1.229^n for sufficiently large n.[1] In this section, we construct a dsr with 1.229^n leaves based on that algorithm. Thus we obtain a MOL(1.229^n) algorithm for the independent set problem. In this section, $|G|$ will represent the number of nodes n in graph G (as opposed to $\max(n, e)$).

The input to Robson's algorithms is a triple (G, S, a) with the following properties: (1) $a \in \{0, 1, 2\}$; (2) if $a = 1$ then $|S| = 2$, and (3) if $a = 0$ then $S = \emptyset$. The output is the size of the largest independent set I containing at least a elements of S. Robson's algorithm is sketched in Figure 4.

In Robson's algorithm, h_0 produces a series of new problems and g produces a tuple of integers. Without loss of generality, we may assume that the each of the graphs in $F_1, \ldots, F_m$ has fewer nodes than the graph in F (otherwise, we unroll the recursion as we did for 3SAT).

From Robson's paper [26], we know that h_0 and g are polynomial time computable, and that MAXIND$(G, \emptyset, 0)$ returns the size of G's largest independent set. Define

$$\langle G, S, a\rangle \prec_1 \langle G', S', a'\rangle \iff |G| < |G'|.$$

Let $T'_{h_0,\prec_1}(F)$ denote the computation tree of the algorithm MAXIND on input F.

Define $T'(F)$ as follows (t, t^1 and t^2 are time upper bounds given by Robson [26]):

$$T'(F) = \begin{cases} t(G, n) & \text{if } F = \langle G, S, 0\rangle \\ t^1(G, |G|, S, |S|) & \text{if } F = \langle G, S, 1\rangle \\ t^2(G, |G|, S, |S|) & \text{if } F = \langle G, S, 2\rangle \end{cases}$$

For each graph G with n nodes, $T'(\langle G, S, a\rangle) \leq 1.229^n$ by the definitions of t, t^1 and t^2 in [26]. From the proof of Theorem 4 and the algorithms for ms_1 and ms_2 in [26],

[1] Although Robson has a faster dynamic programming algorithm for independent set, we are not hopeful of adapting it to molecular computing. Molecular computing may motivate the search for more efficient recursive algorithms that do not use dynamic programming.

function IND(F, k)
$\langle G, S, a\rangle := F$
if $|G| \leq 5$ then
    return brute-force(F)
else
    $\langle F_1, \ldots, F_m\rangle := h_0(F)$
    $\langle b_1, \ldots, b_m\rangle := g(F)$
    if $k \leq b_i$ then
        return true
    else
        return $\bigvee_{i=1}^{m}$ IND$(F_i, k - b_i)$

Figure 5: A decision procedure corresponding to Robson's MAXIND.

we know that

$$T'(F) \geq \max\Big(|T'_{h_0,\prec}(F)|, T'(F_1) + \cdots + T'(F_m)\Big).$$

It is easy to transform the algorithm MAXIND into a dsr for a generalization of the independent set problem.

As in Section 6.1.1, it is easy to transform the function MAXIND into a decision procedure IND. To be precise, we will have

$$\text{IND } (\langle\langle G, S, a\rangle, k\rangle) = \text{ true } \iff \text{ MAXIND } (G, S, a) \geq k.$$

The function IND is given in Figure 5. Therefore, a graph G has an independent set consisting of k vertices if and only if IND$(\langle G, \emptyset, 0\rangle, k) =$ true. Define

$$\langle\langle G, S, a\rangle, k\rangle \prec \langle\langle G', S', a'\rangle, k'\rangle \iff \langle G, S, a\rangle \prec_1 \langle G', S', a'\rangle.$$

If $k > b_i$ for $i = 1, \ldots, m$, then let

- $\langle F_1, \ldots, F_m\rangle = h_0(F)$,
- $\langle b_1, \ldots, b_m\rangle = g(F)$,
- $h(F, k) = \langle\langle F_1, k - b_1\rangle, \ldots, \langle F_m, k - b_m\rangle\rangle$.

If $k \leq b_i$ for some i, the recursive algorithm returns "true" and h is undefined.

The pair $\langle h, \prec\rangle$ is a dsr for the independent set problem. It is easy to see that

$$T'_{h_0,\prec_1}(F) \geq T_{h,\prec}(F, k).$$

Define $T(F, k) = T'(F)$. We have

$$T(F, k) = T'(F) \geq |T_{h_0,\prec_1}| \geq |T_{h,\prec}|.$$

Because $T'(F) \geq T'(F_1) + \cdots + T'(F_m)$ for $\langle F_1, \ldots, F_m\rangle = h(F)$, we have

$$T(F, k) \geq T(F_1, k - b_1) + \cdots + T(F_m, k - b_m).$$

Hence the independent set problem belongs to REC$(T'(F, k)) \subseteq$ NPinit$(T'(F, k))$, so the independent set problem is in MOL(1.229^n).

8. Acknowledgments

We are grateful to William Gasarch for his patience in reading this paper as well as his suggestions in improving the presentation. We are also grateful to Tirza Hirst for helpful discussions and to Ingo Schiermeyer for sharing with us a preliminary draft of [33]. Special thanks go to Mitsunori Ogihara and unknown referees, whose comments greatly improved our presentation.

References

[1] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, Nov. 1994.

[2] L. Adleman. On constructing a molecular computer. In *Proceedings of the 1st DIMACS Workshop on DNA Based Computers (1995)*, pages 1–21. American Mathematical Society, 1996. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 27.

[3] E. Bach, A. Condon, E. Glaser, and C. Tanguay. DNA models and algorithms for NP-complete problems. In *Proceedings of the 11th Annual Conference on Structure in Complexity Theory*, pages 290–299, 1996.

[4] D. Beaver. A universal molecular computer. In *Proceedings of the 1st DIMACS Workshop on DNA Based Computers (1995)*. American Mathematical Society, 1996. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 27.

[5] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3446^n)$: a no-MIS algorithm. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 444–452, 1995.

[6] R. Beigel and B. Fu. Molecular computing, bounded nondeterminism, and efficient recursion. In *Proceedings of the 24th International Colloquium on Automata, Languages, and Programming*, pages 816–826, 1997. LNCS 1256.

[7] R. Beigel and J. Goldsmith. Downward separation fails catastrophically for limited nondeterminism classes. In *Proceedings of the 9th Annual Conference on Structure in Complexity Theory*, pages 134–138, 1994.

[8] D. Boneh, C. Dunworth, and R. J. Lipton. Breaking DES using a molecular computer. In *Proceedings of the 1st DIMACS Workshop on DNA Based Computers (1995)*. American Mathematical Society, 1996. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 27.

[9] D. Boneh, C. Dunworth, R. J. Lipton, and J. Sgall. On the computational power of DNA. *Discrete Applied Math.*, 71:79–94, 1996.

[10] J. F. Buss and J. Goldsmith. Nondeterminism within P. *SICOMP*, 22:560–572, 1993.

[11] J. D. C. Àlvarez and J. Torán. Complexity classes with complete problems between P and NP-complete. In *Foundations of Computation Theory*, pages 13–24. Springer-Verlag, 1989. LNCS 380.

[12] J. Díaz and J. Torán. Classes of bounded nondeterminism. *MST*, 23:21–32, 1990.

[13] J. Goldsmith, M. Levy, and M. Mundhenk. Limited nondeterminism. *SIGACT News*, pages 20–29, June 1996.

[14] L. Hemachandra and S. Jha. Defying upward and downward separation. *Inf. & Comp.*, 121:1–13, 1995.

[15] L. Kari. DNA computing: Arrival of biological mathematics. *he Mathematical Intelligencer*, 19(2):9–22, 1997.

[16] C. M. R. Kintala. *Computations with a restricted number of nondeterministic steps*. PhD thesis, Pennsylvania State University, University Park, PA, 1977.

[17] C. M. R. Kintala and P. C. Fischer. Computations with a restricted number of nondeterministic steps. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pages 178–185, 1977.

[18] C. M. R. Kintala and P. C. Fischer. Refining nondeterminism in relativized polynomial-time bounded computations. *SICOMP*, 9(1):46–53, Feb. 1980.

[19] O. Kullmann. A systematical approach to 3-SAT-decision, yielding 3-SAT-decision in less than 1.5045^n steps. Manuscript available from `kullmann@mi.informatik.uni-frankfurt.de`, 1995.

[20] R. Lipton. Using DNA to solve NP-complete problems. *Science*, 268:542–545, Apr. 1995.

[21] A. Meyer and M. Paterson. With what frequency are apparently intractable problems difficult? Technical Report MIT/LCS/TM-126, M. I. T., 1979.

[22] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Math.*, 10:287–295, 1985.

[23] M. Ogihara. Breadth first search 3SAT algorithms for DNA computers. Technical Report 629, University of Rochester, July 1996.

[24] C. H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the V–C dimension. In *Proceedings of the 8th Annual Conference on Structure in Complexity Theory*, pages 12–18, 1993.

[25] J. Reif. Parallel molecular computation: Models and simulations. In *Proceedings he Seventh Annual ACM Symposium on Parallel Algorithms and Archi tectures*, pages 213–223, 1995.

[26] J. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7:425–440, 1986.

[27] D. Roos and K. Wagner. On the power of bio-computers. Technical report, University of Wurzburg, Feb. 1995. ftp://haegar.informatik.uni-wuerzburg.de/pub/TRs/ro-wa95.ps.gz.

[28] D. Roos and K. Wagner. On the power of dna-computing. *Information and Computation*, 131:95–109, 1996.

[29] P. Rothemund. A DNA and restriction enzyme implementation of Turing machines. In *Proceedings of the 1st DIMACS Workshop on DNA Based Computers (1995)*, pages 75–119. American Mathematical Society, 1996. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 27.

[30] J. Sambrook, E. Fritsch, and T. Maniatis. *Molecular Cloning.* Cold Spring Harbor, NY, 1989.

[31] L. Sanchis. Constructing language instances based on partial information. *International Jour. Found. Comp. Sci.*, 5(2):209–229, 1994.

[32] I. Schiermeyer. Solving 3-satisfiability in less than 1.579^n steps. In *6th Worksh. Computer Science Logic*, pages 379–394. Springer-Verlag, 1993.

[33] I. Schiermeyer. Pure literal lookahead: An $O(1,497^n)$ 3-satisfiability algorithm. Manuscript, August 14, 1996.

[34] C. P. Schnorr. Optimal algorithms for self-reducible problems. In *Proceedings of the 3rd International Colloquium on Automata, Languages, and Programming*, pages 322–337, 1976.

[35] W. Smith and A. Schweitzer. DNA computers in vitro and vivo. Technical report, NEC, 1995.

[36] R. Tarjan. Finding a maximum clique. Technical Report 72-123, Cornell University, 1972.

[37] J. Watson, M. Gilman, J. Witkowski, and M. Zoller. *Recombinant DNA (2nd edition).* Scientific American Books, W.H.Freeman and Co., 1992.

[38] J. Watson, N. Hoplins, J. Roberts, and et. al. *Molecular Biology of the Gene.* Benjamin/Cummmings Menlo Park CA, 1987.