Solving Knapsack Problems in a Sticker Based Model

Pérez–Jiménez, M.J. and Sancho–Caparrini, F.

Dpt. Computer Science and Artificial Intelligence. University of Seville. Spain

Abstract. Our main goal in this paper is to give molecular solutions for two **NP**–complete problems, namely Subset-sum and Knapsack, in a sticker based model for DNA computations. In order to achieve this, we have used a finite set sorting subroutine together with the description of a procedure to formally verify the designed programs through the labeling of test tubes using inductive techniques.

1 Introduction

The *sticker model* was introduced by S. Roweis, E. Winfree et al ([3]) as an abstract model of molecular computing based on DNA with a random access memory and a new form of encoding the information.

The main goal of this work is the resolution, in this model, of two **NP**–complete problems: the *Subset-Sum* problem and the *Knapsack* problem, in its 0/1 bounded and unbounded versions.

The information is represented in the sticker model in a different way from that used in the Adleman-Lipton paradigm. A (n, k, m)-memory strand, with $n \geq k \cdot m$, is n bases in length subdivided into k non-overlapping substrand each m bases long. The substrands should be significantly different from each other. A sticker associated to a (n, k, m)-memory strand is m bases long and complementary to exactly one of the k substrands in the memory strand. If a sticker is annealed to its matching substrand on a memory strand, then the particular substrand is said to be *on*. If no sticker is annealed to a substrand, then the region is said to be *off*. A (n, k, m)-memory complex is a (n, k, m)-memory strand along with its annealed stickers (if any). In a direct way, (n, k, m)-memory complexes represent bit strings of $\{0, 1\}^k$. For this reason, it is usual to identify them either as binary functions ($\sigma : \{1, ..., k\} \longrightarrow \{0, 1\}$, such that $\sigma(i) = 1$ if and only if the i-th substrand is *on*), or as subsets of $\{1, ..., k\}$ by means of the characteristic function.

Within the sticker model, a tube is a finite multiset whose elements are memory complexes (that is, a collection of memory complexes where each one can be repeated). The following operations over tubes of the sticker model are used in this paper:

- *Merge* (T_1, T_2): the memory complexes from the tubes T_1, T_2 are combined to form the multiset union of all strings in the two input tubes. We write $Merge(T_1, T_2) = T_1 \cup T_2$ as well.

- *Separate* (T, i): Given a tube, T, and an integer, i ($1 \leqslant i \leqslant$ number of substrands that form each complex of T), create two new tubes, $+(T, i)$ and $-(T, i)$, where $+(T, i)$ (resp. $-(T, i)$) contain all strings of T having the i–th substrand set to 1 (resp. set to 0). We write $(T_1, T_2) \leftarrow separate(T, i)$ to indicate that $T_1 = +(T, i)$ and $T_2 = -(T, i)$.
- *Set* (T, i): Given a tube, T, and an integer, i ($1 \leqslant i \leqslant$ number of substrands that form each complex of T), this operation produces a new tube where the i–th substrand of each memory complex in T is set to 1. That is, the sticker for that bit is annealed to i-th region on every memory complex in T.
- *Read* (T): Given a nonempty tube T, this operation reads its content. For that, one memory complex must be isolated from T and its annealed stickers, if any, determined.

A (k, l)-library, with $1 \leq k \leq l$, consists of memory complexes with k substrands, the first l substrands are either *on* or *off*, in all possible ways, whereas the last $k - l$ substrands are *off*.

In section 2, the problem of sorting the elements of a finite family of finite sets, according to their cardinality, is studied, and for the first time, a program that is able to solve this problem is described within the sicker model. If it is taken into account that just two molecular operations have been used, namely *separate* and *merge*, the program designed can also be considered as a program within the unrestricted model of Adleman ([1]).

In section 3 we give a filling subroutine within the sticker model to encode the weight of subsets regarding a given positive function that will be used in following sections.

In section 4, we give a molecular solution within the sticker model for the Subset–Sum problem, using the sorting by cardinality program and the filling subroutine. Formal verification of the programs designed in sections 2, 3 and 4 is established through the prior labeling of the distinct tubes which appear in the execution. Then we prove the soundness and completeness of these programs using inductive techniques and analyzing the *history* of every molecule in the initial test–tube along the process.

In sections 5 and 6, we give molecular solutions, within the sticker model, for Knapsack problem (0/1 bounded and unbounded versions), based in both the sorting by cardinality program given in section 2, and the filling subroutine given in section 3.

All designed programs use a linear number of tubes, and the number of molecular operations is, basically, quadratic.

2 Sorting by Cardinality

Problem: *Let* $A = \{1, ..., p\}$, $B = \{b_1, ..., b_s\} \subseteq A$, $\mathcal{F} = \{D_1, ..., D_t\} \subseteq \mathcal{P}(A)$. *Sort the sets of* $\mathcal{F}$ *according to their relative cardinality to* B *(that is, according to the number of elements of* $B \cap D_i$*).*

Next, we will design a molecular program within the sticker model which solves the above problem.

– The input tube, T_0, will contain memory complexes, σ, based on DNA, encoding each set of the family $\mathcal{F}$. For this, each complex of T_0 will be represented through a boolean function, that is, $T_0 = \{\{\sigma : |\sigma| = p \wedge \exists j\, (\chi_{D_j} = \sigma)\}\}$, where χ_{D_j} is the characteristic function of D_j in A ($\chi_{D_j}(i) = 1$ if $i \in D_j$, and $\chi_{D_j}(i) = 0$ if $i \in A - D_j$).
– The program consists of a main loop FOR with s steps. In the i–th step, $i+1$ tubes, $T_0, T_1, \ldots, T_i$, are generated verifying the condition: $\forall \sigma\ (\sigma \in T_j \longrightarrow |\sigma \cap \{b_1, \ldots, b_i\}| = j)$. In order to achieve this, we design the body of the loop by induction. Once the tubes $T_0, T_1, \ldots, T_i$ corresponding to the i–th step have been built, the tubes of the next step $T_0, T_1, \ldots, T_i, T_{i+1}$ are generated in this way:

$$\begin{cases} T_0 = -(T_0, b_{i+1}) \\ T_j = +(T_{j-1}, b_{i+1}) \cup -(T_j, b_{i+1}) \quad (1 \leqslant j \leqslant i) \\ T_{i+1} = +(T_i, b_{i+1}) \end{cases}$$

The execution of the molecular program can be described starting from a rooted graph that we denominate *labeled merge–binary tree* that is defined by recursion as follows:

– A node with a label is a labeled merge–binary tree of depth 0.
– Let A be a labeled merge–binary tree of depth h. From it, a labeled merge–binary tree, A', of depth $h+1$ is built in this way:
  • Initially, each leaf of A determines two children.
  • The right child of each leaf and the left one of the next leaf give a node of A' whose label is the composition of the labels of this children by a certain fixed binary operation.

In the description that has been carried out so far, the nodes of the merge–binary tree of execution are labeled by means of tubes. The left and right children of a tube, T, of depth h are labeled starting from the separate operation: $(T_{left}, T_{right}) \longleftarrow separate(T, b_{h+1})$. Finally, the binary operation considered is the molecular merge operation applied to the tubes indicated by the labels of the nodes.

These ideas suggest the following molecular program:

```
Input: (T_0, B)
       for i = 1 to s do
           (T_0, T_1') ← separate (T_0, b_i)
           for j = 0 to i − 1 do
               (T_j'', T_{j+1}') ← separate (T_j, b_i)
               T_j ← T_j' ∪ T_j''
           end for
           T_i ← T_i'
       end for
Output: T_0, ..., T_s
```

The procedure described will return $s+1$ tubes and we will note them as: $\texttt{Cardinal\_sort}(T_0, B)[j]$, $(0 \leq j \leq s)$. We have that $|\texttt{Cardinal\_sort}(T_0, B)[j]| = j$.

This molecular program uses $2s$ tubes and the number of molecular operations is $\frac{s\cdot(s+3)}{2}$.

Let us note that the program we have given to solve the sorting problem is valid in a model without random access memory, like the unrestricted model of Adleman. The simplest way to see this is to adapt the input tube, replacing memory complex for single strands of DNA.

To establish the formal verification of the algorithm program, we will proceed to label the tubes obtained along the execution so that we can individualize them in any moment of the running.

```
Input: T_0
       T_{0,0} ← T_0;  T_{0,-1} ← ∅;  T_{0,1} ← ∅
       for i = 1 to s do
           T_{i,-1} ← ∅;  T_{i,i+1} ← ∅
           for j = 0 to i do
               T_{i,j} ← +(T_{i-1,j-1}, b_i) ∪ −(T_{i-1,j}, b_i)
           end for
       end for
Output: T_{s,0}, ..., T_{s,s}
```

By means of convenience, we assume that $T_{0,-1} = T_{0,1} = \emptyset$, and we will note $B_j = \{b_1, ..., b_j\}$, and, by definition, we will take $B_0 = \emptyset$.

Proposition 1. $\forall\, i\ (1 \leq i \leq s \rightarrow \forall\, j \leq i\ \forall\, \sigma\ (\sigma \in T_{i,j} \rightarrow |\sigma \cap B_i| = j))$.

Proof. By induction on i. Let us see that $\forall\, j \leq 1\ \forall\, \sigma \in T_{1,j}\ (|\sigma \cap B_1| = j)$.

- Let $\sigma \in T_{1,0} = +(T_{0,-1}, b_1) \cup -(T_{0,0}, b_1)$. Since $T_{0,-1} = \emptyset$ and $T_{0,0} = T_0$, it results that $\sigma \in -(T_0, b_1)$, that is, $b_1 \notin \sigma$. Then, $|\sigma \cap B_1| = 0$.
- Let $\sigma \in T_{1,1} = +(T_{0,0}, b_1) \cup -(T_{0,1}, b_1)$. Since $T_{0,1} = \emptyset$, it results that $\sigma \in T_{0,0} = T_0$ and $b_1 \in \sigma$, then $|\sigma \cap B_1| = 1$

Let i $(1 \leq i < s)$ be such that $\forall\, j \leq i\ \forall\, \sigma \in T_{i,j}\ (|\sigma \cap B_i| = j)$. Let us see that the result verifies for $i + 1$. For it, we now proceed by induction on j: $\forall\, j \leq i + 1\ \forall\, \sigma \in T_{i+1,j}\ (|\sigma \cap B_{i+1}| = j)$.

- Let $\sigma \in T_{i+1,0} = +(T_{i,-1}, b_{i+1}) \cup -(T_{i,0}, b_{i+1})$. Since $T_{i,-1} = \emptyset$, it results that $\sigma \in T_{i,0}$ and $b_{i+1} \notin \sigma$, by induction hypothesis we deduce that $|\sigma \cap B_i| = 0$. So $|\sigma \cap B_{i+1}| = 0$, since $b_{i+1} \notin \sigma$.
- Let $j > 0$ and $\sigma \in T_{i+1,j} = +(T_{i,j-1}, b_{i+1}) \cup -(T_{i,j}, b_{i+1})$. Then
  - If $\sigma \in T_{i,j-1}$ and $b_{i+1} \in \sigma$, by induction hypothesis we have that $|\sigma \cap B_i| = j - 1$. Since $b_{i+1} \in \sigma$, we conclude that $|\sigma \cap B_{i+1}| = j - 1 + 1 = j$.
  - If $\sigma \in T_{i,j}$ and $b_{i+1} \notin \sigma$, by induction hypothesis we have that $|\sigma \cap B_i| = j$. As $b_{i+1} \notin \sigma$, we have $|\sigma \cap B_{i+1}| = j$. □

Proposition 2. $\forall\, \sigma \in T_0\ \forall\, i\ (0 \leq i \leq s \rightarrow \sigma \in T_{i,|\sigma \cap B_i|})$.

Proof. By induction on i. For $i = 0$, the result is trivial.
Assume the result holds for i $(0 \leq i < s)$; we will prove it for $i + 1$.

– If $b_{i+1} \in \sigma$, we have $|\sigma \cap B_{i+1}| = 1 + |\sigma \cap B_i|$. By induction hypothesis, $\sigma \in T_{i,|\sigma\cap B_i|}$, then $\sigma \in +(T_{i,|\sigma\cap B_i|}, b_{i+1}) \subseteq T_{i+1,|\sigma\cap B_i|+1}$.
– If $b_{i+1} \notin \sigma$, then $|\sigma\cap B_{i+1}| = |\sigma\cap B_i|$. By induction hypothesis, $\sigma \in T_{i,|\sigma\cap B_i|}$, then $\sigma \in -(T_{i,|\sigma\cap B_i|}, b_{i+1}) \subseteq T_{i+1,|\sigma\cap B_i|} = T_{i+1,|\sigma\cap B_{i+1}|}$. □

¿From the preceding propositions it may be concluded, respectively, soundness (every molecule of the output tube provides a correct solution associated to that tube) and completeness (every molecule of the input tube appears in the corresponding output tube, according to its cardinality) of the designed program.

Corollary 1. *(Soundness)* $\forall\, j\ \forall\, \sigma\ (0 \leq j \leq s \wedge \sigma \in T_{s,j} \rightarrow |\sigma \cap B| = j)$.

Corollary 2. *(Completeness) If* $\sigma \in T_0$ *and* $|\sigma \cap B| = j$, *then* $\sigma \in T_{s,j}$.

As cases of particular interest, that we will use in other molecular programs, we get the following:

– `Cardinal_sort(`T_0`)`, when $B = A$.
– `Cardinal_sort(`T_0, l, k`)`, when $B = \{l, l+1, ..., k\}$.

3 A filling subroutine

In this section we show a molecular program that will be used as auxiliary subroutine to solve the Subset–Sum problem and the Knapsack problem in the following sections.

Let $A = \{1, ..., p\}$, $r \in \mathbb{N}$ and $f : A \longrightarrow \mathbb{N}$ a function. If $B \subseteq A$, we note $f(B) = \sum_{i\in B} f(i)$. For convenience we define $f(0) = 0$. Let $q_f = f(A)$, $A_i = \{0, \ldots, i\}$ $(0 \leq i \leq p)$ and T_0 a multiset of (n, k, m)-memory complexes, σ, with $k \geq p + r + q_f$.

As it was seen in the previous section, each $\sigma \in T_0$ encodes a subset $B_\sigma \subseteq A$ characterized by the condition $B_\sigma = \{i :\ 1 \leq i \leq p\ \wedge\ \sigma(i) = 1\}$, and reciprocally, each subset, $B \subseteq A$, can be encoded by a molecule $\sigma_B \in T_0$, characterized by the condition: $\sigma_B(i) = 1$ if and only if $i \in B$.

If $\sigma \in T_0$, we can suppose that it is formed by the following *zones*:

$(A\sigma) = \sigma(1) \ldots \sigma(p)$, $\quad (F\sigma) = \sigma(p + r + 1) \ldots \sigma(p + r + q_f)$
$(L\sigma) = \sigma(p + 1) \ldots \sigma(p + r)$, $(R\sigma) = \sigma(p + r + q_f + 1) \ldots$

The subroutine works over T_0, and it modifies their elements making that the molecules of the output tube store in $(F\sigma)$ the weight, regarding f, of the subset of A encoded in $(A\sigma)$ (zones $(R\sigma)$ and $(L\sigma)$ have no effect in the process, but they will be useful for a general use). That is:

$$\sum_{i=1}^{p} \sigma(i) f(i) = \sum_{j=p+r+1}^{p+r+q_f} \sigma(j)$$

The designed program will be noted `Parallel_Fill(`T_0, f, p, r`)`:

```
Input: (T_0, f, p, r)
       for i = 1 to p do
           (T^+, T^-) ← separate(T_0, i)
           for j = 1 to f(i) do
               T^+ ← set(T^+, p + r + f(A_{i-1}) + j)
           end for
           T_0 ← merge (T^+, T^-)
       end for
Output: T_0
```

To establish the formal verification of the algorithm, we will proceed to label the tubes obtained along the execution.

```
Input: (T_0, f, p, r)
       for i = 1 to p do
           (T^+_{i,0}, T^-_i) ← separate(T_{i-1}, i)
           for j = 1 to f(i) do
               T^+_{i,j} ← set(T^+_{i,j-1}, p + r + f(A_{i-1}) + j)
           end for
           T_i ← merge (T^+_{i,f(i)}, T^-_i)
       end for
Output: T_p
```

For each i $(1 \leq i \leq p)$ we consider the following *regions*:

$$R_i = \{p + r + f(A_{i-1}) + 1, ..., p + r + f(A_i)\}$$

Definition 1. *For each $\sigma \in T_0$ and each k $(1 \leq k \leq p)$, we will note σ^k the molecule obtained from σ after the execution of the k–th step in the main loop of the program.*

That is, the molecules σ^k provide the *history* of the molecule σ of the input tube, while the program is running. Keeping in mind the syntactic structure of the program, it is straightforward to prove the following results:

Lemma 1.

1. *The initial zone of the molecule (encoding the subset of A) does not change along the execution of the program; that is,*

$$\forall\, \sigma \in T_0\ \forall\, k\ (1 \leq k \leq p \rightarrow (A\sigma) = (A\sigma^k)) \tag{1}$$

2. *The molecules that are obtained in the k-th step of the main loop are stored in the tube T_k; that is,*

$$\forall\, \sigma \in T_0\ \forall\, k\ (1 \leq k \leq p \rightarrow \sigma^k \in T_k) \tag{2}$$

3. *Every molecule of the k-th tube comes from some molecule in the initial tube; that is,*

$$\forall\, k\ (1 \leq k \leq p \rightarrow \forall\, \tau \in T_k\ \exists\, \sigma \in T_0\ (\sigma^k = \tau)) \tag{3}$$

4. *The execution of a step of the main loop does not modify the regions corresponding to previous steps; that is,*

$$\forall\, \sigma \in T_0\ \forall\, i\ \forall\, k\ (1 \leq i \leq k \leq p \rightarrow \sigma^i_{|R_i} = \sigma^k_{|R_i}) \tag{4}$$

5. *After the execution of the i-th step of the main loop, the region R_i of σ has been modified to agree with the value of $\sigma(i)$; that is,*

$$\forall\, \sigma \in T_0\ \forall\, i\ \forall\, k\ (1 \leq i \leq k \leq p \rightarrow \sigma^k_{|R_i} \equiv \sigma(i)) \tag{5}$$

6. *The execution of a step of the main loop does not modify the zones $(L\sigma)$ and $(R\sigma)$; that is,*

$$\forall\, \sigma \in T_0\ \forall\, k\ (1 \leq k \leq p \rightarrow (L\sigma) = (L\sigma^k) \wedge (R\sigma) = (R\sigma^k)) \tag{6}$$

The following result assures us that the main loop modifies the regions R_i of the molecules to encode the partial weight of the set represented by each one of them.

Proposition 3. *Let $B \subseteq A$ such that $\sigma_B \in T_0$, then for each k $(1 \leq k \leq p)$ we have that:*

$$f(B \cap \{1, ..., k\}) = \sum_{j=p+r+1}^{p+r+f(A_k)} \sigma^k_B(j)$$

Proof. ¿From (1) it follows that

$$f(B \cap \{1, ..., k\}) = \sum_{i=1}^{k} f(i) \cdot \sigma_B(i) = \sum_{i=1}^{k} f(i) \cdot \sigma^k_B(i)$$

On the other hand, (1) and (5) assures that

$$f(i) \cdot \sigma^k_B(i) = \textstyle\sum_{j \in R_i} \sigma^k_B(j))\ (1 \leq i \leq k)$$ □

Corollary 3. *For each $B \subseteq A$ such that $\sigma_B \in T_0$ there exists $\tau \in T_p$ such that $f(B) = \sum_{i=p+r+1}^{p+r+q_f} \tau(i)$.*

Proof. Given $B \subseteq A$, let us consider the associated molecule $\sigma_B \in T_0$. It suffices to consider $\tau = \sigma^p_B$, since $f(B) = f(B \cap \{1, ..., p\}) = \sum_{j=p+r+1}^{p+r+q_f} \sigma^p_B(j)$. □

4 Subset-Sum problem

Problem: *Let $A = \{1, ..., p\}$ and $w : A \longrightarrow \mathbb{N}$ a weight function. Let $k \in \mathbb{N}$ be such that $k \leq w(A) = q_w$. Determine whether there exists a subset $B \subseteq A$ such that the sum of the weights of the elements in B is, exactly, k.*

Next we will design a molecular program within the sticker model that solves the Subset–Sum problem. The input tube, T_0, will be a $(p + q_w, p)$-library. In a first stage (filling), each molecule, σ, of the input tube is filled in order to obtain in their last q components the weight of the subset that it encodes; the molecules of the resulting tube of the previous stage are ordered according to their cardinality. Finally, the k-th tube is read: it contains the molecules from the input tube encoding subsets of A of weight k, if any.

These ideas suggest the design of the following molecular program:

`Subset_Sum`(p, w, k)

$q_w \leftarrow \sum_{i=1}^{p} w(i)$

$T_0 \leftarrow (p + q_w, p)$-`library`

$T_1 \leftarrow$ `Parallel_Fill`$(T_0, w, p, 0)$

$T_{out} \leftarrow$ `Cardinal_sort`$(T_1, p + 1, p + q_w)[k]$

`Read`(T_{out})

The number of used tubes (including the subroutines) is $4 + 2q$ and the number of molecular operations is $2p + q + 1 + \frac{q \cdot (q+3)}{2}$.

The following result shows the soundness of the molecular program; that is, every molecule in the output tube encodes a correct solution for the Subset–Sum problem.

Theorem 1. *(Soundness) If $T_{out} \neq \emptyset$, then there exists $B \subseteq A$ such that $w(B) = k$.*

Proof. Taking $\tau \in T_{out}$, and applying (3) and proposition 3, we obtain $\sigma \in T_0$ verifying the result for B_σ. □

Next theorem proves the completeness of the given molecular program; that is, every molecule in the input tube encoding a correct solution of the Subset–Sum problem, is in the output tube.

Theorem 2. *(Completeness) Let $\sigma \in T_0$ be such that $w(B_\sigma) = k$. Then $T_{out} \neq \emptyset$*

Proof. Let $\sigma \in T_0$ be such that $w(B_\sigma) = k$, from corollary 3, after the execution of the filling subroutine, we have a molecule $\tau = \sigma^p \in T_1$ such that $w(B_\sigma) = \sum_{i=p+1}^{p+q_w} \tau(i)$. Then $\tau \in T_{out}$. □

Note. The above program does not only solve the problem of decision, but rather, it returns all the solutions to the problem. If we only wanted to solve the decision problem, once the filling stage has been executed, we could use some appropriate restriction enzymes which remove from each molecule, σ, in the tube T_p, the initial region $(A\sigma)$, and then apply an appropriate procedure `Cardinal_sort`, so that we may work with molecules of smaller length in the final stage.

5 0/1 Bounded Knapsack Problem

Problem: *Let $A = \{1, ..., p\}$ be a non empty finite set, $w : A \longrightarrow \mathbb{N}$ a weight function, and $\rho : A \longrightarrow \mathbb{N}$ a function of values. Let $k, k' \in \mathbb{N}$ be such that $k \leq w(A) = q_w$ and $k' \leq \rho(A) = q_\rho$. Determine whether there exists a subset $B \subseteq A$ such that $w(B) \leq k$ and $\rho(B) \geq k'$.*

Next we will design a molecular program in the sticker model that solves the problem 0/1 bounded Knapsack problem: we begin with a $(p + q_w + q_\rho, p)$-library. In a first stage (filling), we proceed as in the previous program: each molecule of the initial tube is filled appropriately so that it encodes the *weight* of the associate subset; next the molecules, σ, of the resulting tube are ordered

according to the cardinal of $w(A\sigma)$, being obtained some tubes, $T_0, ..., T_{q_w}$, such that $\forall\sigma\ (\sigma \in T_j \Rightarrow |w(A\sigma)| = j)$. With the tube $T_0 \cup ... \cup T_k$ a second stage of filling is carried out, regarding the function of *values*. Then, the molecules, σ, from the resulting tube are ordered according to the cardinal of $\rho(A\sigma)$, being obtained, again, some tubes, $T_0, ..., T_{q_\rho}$, such that $\forall\sigma\ (\sigma \in T_j \Rightarrow |\rho(A\sigma)| = j)$. Finally, the tube $T_{k'} \cup ... \cup T_{q_\rho}$, containing the encoded solutions to the problem, is read.

These ideas suggest the following molecular program:

```
Knapsack(p, w, ρ, k, k')
    q_w ← Σ_{i=1}^{p} w(i);  q_ρ ← Σ_{i=1}^{p} ρ(i);  T_0 ← (p + q_w + q_ρ, p)-library
    T_0 ← Parallel_Fill(T_0, w, p, 0)
    Cardinal_sort(T_0, p + 1, p + q_w)
    T_1 ← ∅
    for i = 1 to k do
        T_1 ← merge (T_1, Cardinal_sort(T_0, p + 1, p + q_w)[i])
    end for
    T_0 ← Parallel_Fill(T_1, ρ, p, q_w)
    Cardinal_sort(T_0, p + q_w + 1, p + q_w + q_ρ)
    T_1 ← ∅
    for i = k' to q_ρ do
        T_1 ← merge (T_1, Cardinal_sort(T_0, p + q_w + 1, p + q_w + q_ρ)[i])
    end for
    Read(T_1)
```

The number of tubes used by the program (again, including the subroutines) is $5 + 2 \cdot \max\{q_w, q_\rho\}$, and the number of molecular operations carried out in the execution is $4p + k - k' + \frac{q_w \cdot (q_w+5) + q_\rho \cdot (q_\rho+7)}{2} + 1$.

The formal verification of this program is similar to the one for the Subset–Sum problem, we omit its proof since it does not show any new ideas for the verification methods for the sticker model.

6 0/1 Unbounded Knapsack Problem

Problem: *Under the same conditions as the Knapsack problem, determine a subset $B \subseteq A$ such that $\rho(B) = \max\{\rho(C) :\ C \subseteq A \wedge w(C) \leq k\}$.*

A molecular solution is obtained from the solution of the bounded problem, changing the final output stage: after the sorting of the tubes regarding the function of values, we will choose the non empty tube with bigger index.

```
Unbounded_Knapsack(p, w, ρ, k)
    q_w ← Σ_{i=1}^{p} w(i);  q_ρ ← Σ_{i=1}^{p} ρ(i);  T_0 ← (p + q_w + q_ρ, p)-library
    T_0 ← Parallel_Fill(T_0, w, p, 0)
    Cardinal_sort(T_0, p + 1, p + q_w)
    T_1 ← ∅
    for i = 0 to k do
        T_1 ← merge(T_1, Cardinal_sort(T_0, p + 1, p + q_w)[i])
    end for
```

$T_0 \leftarrow$ `Parallel_Fill`(T_1, ρ, p, q_w)
$i \leftarrow q_\rho$; $t \leftarrow 0$
`Cardinal_sort`$(T_0, p + q_w + 1, p + q_w + q_\rho)$
`while` $i \geq 1 \ \wedge \ t = 0$ `do`
    $T' \leftarrow$ `Cardinal_sort`$(T_0, p + q_w + 1, p + q_w + q_\rho)[i]$
    `if` $T' \neq \emptyset$ `then: Read`(T'); $t \leftarrow 1$
    `else:` $i \leftarrow i - 1$
`end while`

The number of tubes used by the program is $5 + 2 \cdot \max\{q_w, q_\rho\}$, and the number of molecular operations carried out in the execution of the program, $4p + k - k' + \frac{q_w \cdot (q_w+5) + q_\rho \cdot (q_\rho+9)}{2}$, is easily obtained from the bounded case.

7 Conclusions

In this work, a molecular program that allows us to sort a finite family of finite sets, according to their cardinality, has been presented within the sticker model. In order to achieve this, we have used a new technique which is different from the one used by S. Roweis et al ([3]) to solve the Minimal Set Cover problem.

Also, we have designed original molecular programs in the aforementioned model that solve the Subset–Sum problem and the Knapsack problem (0/1 bounded and unbounded versions), which are NP–complete problems. As a subroutine, we present a molecular program (`Parallel_Fill`) within the sticker model that performs the computation of the weight of subsets regarding a given function. The molecular solutions presented of Subset Sum and Knapsack problems use a linear number of tubes, and the number of molecular operations is, basically, quadratic; nevertheless, the volume of required DNA (space complexity) to perform the computations may vary by exponential factors.

We present formal verification of the designed programs, proving soundness and completeness of these programs through the labeling of tubes and using techniques of induction.

The study of the formal aspects of molecular programs opens up a new research field for their automatic processing by means of theorem provers (ACL2, PVS, ...). The production of prototypes which are able to be executed regarding molecular computational models within the framework of theorem provers, will allow to automate both the soundness and completeness of molecular programs, which have been designed within these models.

References

1. Adleman, L. On constructing a molecular computer, in *DNA based computers*, R.J. Lipton and E.B. Baum, eds., American Mathematical Society, 1996, 1–22.
2. Garey M.R.; Johnson D.S. *Computers and intractability*, W.H. Freeman and Company, New York, 1979.
3. Roweis, S.; Winfree, E.; Burgoyne, R.; Chelyapov, N.; Goodman, M; Rothemund, P. and Adleman, L. A Sticker–Based Model for DNA Computation, *J. Comp. Biol.* 5, 615–629, 1998