

# On a DNA experiment for solving a certain NP-complete problem

E. Stoschek\*      M. Sturm\*      T. Hinze\*

April 6, 1999

## Abstract

The publication of the Adleman experiment in 1994 gave the crucial impetus for many approaches to DNA computing as one possible form of future computing [Adl94]. There is worldwide discussion about chances and limits of this model for computation that focus on implementations and also important theoretical considerations about "Calculating inside the reaction tube". Practical approaches to DNA computing are considered particularly interesting, so to say solutions of NP-problems in the laboratory, from which theoretical models can be derived. A certain NP-problem was solved in the laboratory by an interdisciplinary researching group at the University of Technology Dresden, Germany. We are grateful to H.K. Schackert, M. Hauses, and O.N. Koufaki for their help and support to establish the lab-implementation of the algorithm. Here, we show how to describe the algorithm of the NP-problem in this language and how to implement it in the laboratory. All functions of the language DNA-HASKELL are executed by lab-experiments. We argue which efficient possibilities exist in the molecular biology to solve mathematical problems, especially NP-problems. The theory of complexity is used to evaluate solutions of the laboratory and the model.

## 1 Introduction

An interesting alternative to the application of numbers and processors is the use of biomolecules and laboratory techniques in the computer science [Kar97]. It is well-known that science and economy are very interested in solving a certain class of mathematical problems – the class of NP-problems. The state of the art technology is unable to solve these problems at reasonable time. This led to the search for alternative hardware constructions allowing a high quantity processing to meet exactly these requirements. A variety of approaches to future computing (quantum computing, neural computing, and DNA computing) are focused by numerous scientific studies and are compared with conventional electronic computing.

The development of DNA computing was launched in 1994. After the study of *James Watson's* "Molecular biology" the well-known mathematician Leonard Adleman, an expert in the field of data protection and cryptographics, recognized the relations between DNA and a computer and established the first DNA computer to successfully solve the Hamiltonian Path Problem.

The advantages were obvious; DNA computing is based on a biological parallel computer using DNA operations with enormous memory capacity and operational speed. Adleman's model processes  $1.2 \cdot 10^{18}$  operations per second. This is approximately 1 200 000 times faster than the fastest supercomputer currently available.

---

\*Dresden University of Technology, Department of Computer Science, Institute of Theoretical Computer Science. E-mail: {sturm,th1}@tcs.inf.tu-dresden.de, Fax: +49-351-4638255

A DNA computer is  $10^{10}$  fold more energy efficient than a conventional computer. DNA stores information at a density of  $1\text{Bit}/\text{nm}^3$ , whereas the classic storage density amounts to  $1\text{Bit}/10^{12}\text{nm}^3$  [Kar97].

From the very beginning research in the field of DNA computing followed both aspects, namely the theoretical as well as experimental processing of the problems. However, little is published on dealing with laboratory practice [Das98]. This is due to the complexity of working techniques in a molecularbiological laboratory especially for a theoretical computer scientist.

After Adleman's experiment Lipton formalized the model of parallel computation. He used the 3SAT problem and proved that this problem – like any NP-problem – can be solved in polynomial time. It required a lot of experiments to warrant reproducibility and resistance to errors of DNA operations. These studies focus on PCR (Polymerase Chain Reaction).

Over the last years a variety of DNA models was developed and discussed under different aspects. The objectives were their complexity, their stability, and last not least their practicability. The DNA computer represents a powerful tool to describe, execute and implement parallel processes.

The scientific activities of the authors focus on the solution of a certain NP-problem – the integer knapsack problem. The proposed model is flexible and comprehensive enough to solve any NP-problem.

## 2 DNA-HASKELL

The computer science utilizes the molecularbiological techniques to solve a certain class of mathematical problems. One component of this class is the integer knapsack problem well-known as NP-complete. For a definition see section 4.

The solution to this problem based on DNA computing resulted in the development of a model that involves several layers. The lowest level includes the molecularbiological operations that are required for NP-problem solving. A specific selection of a variety of operations solves the integer knapsack problem. The operations process DNA strands and sequences (DeoxyriboNucleic Acid). DNA is

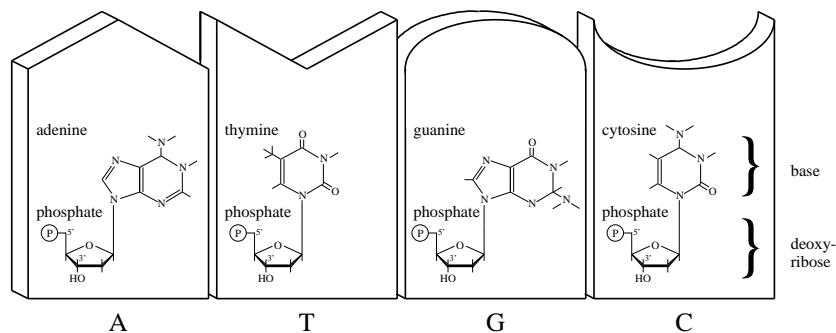


Figure 1: nucleotides with four different bases [Ber92]

the storage medium of genetic information in cell organisms. Phenotypic properties are inherited by propagating genetic information. The carrier of this genetic information is DNA composed of a sequence of nucleotides. There are four different types of nucleotides. Nucleotides vary in the bases adenine, thymine, guanine and cytosine. The corresponding nucleotides are named A, T, G, and C. A base is a

circular molecule consisting of chemical bonds between carbon and nitrogen atoms. Additional molecules are linked to each base and the deoxyribose ring is coupled to phosphate. DNA strands are formed by chemical bonds between the phosphate on a deoxyribose and the deoxyribose of the adjacent nucleotide. In brief, the deoxyribose phosphate units represent the chainlinks of the DNA strand. For each deoxyribose in DNA, one phosphate is joined to the deoxyribose 3'-position and one to the 5'-position. Thus each strand has a definite polarity that can be read bidirectionally (5'-3', 3'-5'). Strands can be handled as single strands and as double

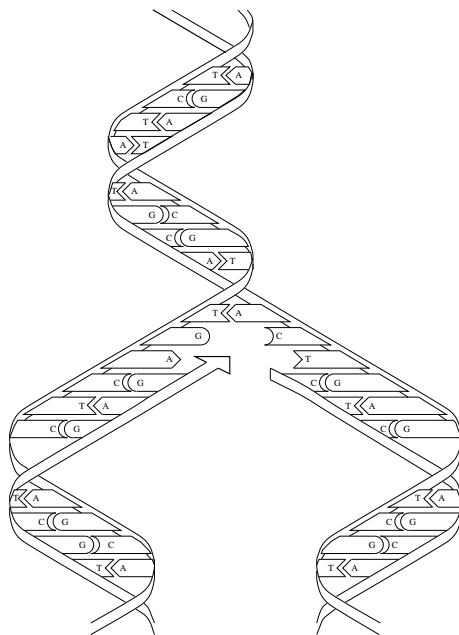


Figure 2: the double helix

strands. The bases of opposite strands fit together precisely in which A pairs with T and G with C through complementary hydrogen bonds as illustrated in Figure 2. The nucleotide sequence of one strand defines the sequence of the opposite strand in the double strand system to form complementary nucleotide sequences. The double strands paired in this way are named complementary. Both strands have opposite directions, one strand from 5' to 3' (sense) and the other one from 3' to 5' (anti-sense). Winding both complementary strands around an axis forms the structure of a double helix. This classical model of DNA was published in 1953 by Watson/Crick and gained great significance in molecular biology, genetic engineering and medicine.

DNA strands can be recombined by exchange of nucleotide sequences. The recombination technology uses a variety of molecularbiological operations that are in part considered in DNA computing. Nearly all these operations can be performed in a reaction tube:

- **Annealing**  
This operation forms under special conditions (temperature, ...) a double strand from at least two antiparallel and complementary single strands.
- **Melting**  
Melting is the opposite of Annealing. A double strand can be divided into its single strands given certain circumstances.

- **Synthesis**  
In particular, this operation gained great significance for problem solving in computer science. A single strand can be synthesized with any base sequence desired.
- **Ligation**  
The enzyme DNA ligase catalyses concatenating of DNA double strands under certain circumstances relying on the structure of the ends of the DNA strands.
- **Extraction**  
This operation causes that all these DNA strands will be extracted from a given double strand set that contain a certain "substrand".
- **Merging**  
This operation allows to merge the contents of several tubes into one tube.
- **Agarose gel electrophoresis**  
DNA strands of different lengths are separated by gel electrophoresis. The negative charged DNA strands are migrating through the agarose matrix in an electric field. The shorter the strands the faster they move. The DNA can be made visible by ethidiumbromide staining, a chemical substance that can be illuminated under UV-light.
- **Phosphorylating/ Dephosphorylating**  
To control the ligation it is possible to insert or to remove a phosphate molecule at the 5'-ends of DNA strands.
- **Blunting**  
This operation fills up sticky ends starting with the 3'-end up to the 5'-end by inserting complementary nucleotides until a blunt end is reached.
- **Polymerase Chain Reaction (PCR)**  
The PCR is a technique to amplify specific DNA sequences using a polymerase to produce newly synthesized copies of a certain DNA sequence. The PCR approach represents an efficient technique to amplify DNA strands without cut and paste mechanism.

The language DNA-HASKELL was conceived on the base of the functional language HASKELL. The decision for a functional language is due to the fact that the abstraction level of a functional program is close to the level of problem specification. This quality allows an easier mathematical handling. Only because of this property correctness proofs, verification, and analysis of program properties are possible. DNA computing as a modern model for computations is convincing only if a formal description of the labwork can be done. DNA-HASKELL with its specific functions represents the second layer of the model.

#### Declaration of DNA data types

```

data Base      = A | T | C | G | *
data Label     = P | B | C | @
data Basepair  = [A,T]| [T,A]| [C,G]| [G,C]| [A,*]| [T,*]| [C,*]| [G,*]|
                [* ,T]| [* ,A]| [* ,G]| [* ,C]
data Labelpair = [@,@]| [@,P]| [@,B]| [@,C]| [P,@]| [B,@]| [C,@]|
                [P,P]| [B,B]| [C,C]| [P,B]| [B,P]| [C,B]| ...
data Strand    = NIL | Cons Basepair Strand
data Dnastrand = Labelpair ++ Strand ++ Labelpair

```

```
data Tube      = NIL | Cons Dnastrand Tube
```

Examples for instances of these data types:

```
[A,*] :: Basepair
[[A,T],[T,A],[A,*]] :: Strand
[[@,@],[A,T],[T,A],[A,*],[@,@]] :: Dnastrand
[[[P,@],[A,T],[T,A],[@,@]],[[@,@],[C,G],[@,@]]] :: Tube
```

Definition of selected DNA operations in DNA-HASKELL<sup>1</sup>

	DNA strand	Notation in DNA-HASKELL	for example
single strands unlabeled	5'-ATCGAT-3'	[[@,@]] ++ [[A,*],[T,*],[C,*],[G,*],[A,*],[T,*]] ++ [[@,@]]	
	3'-TAGCTA-5'	[[@,@]] ++ [[*,T],[*,A],[*,G],[*,C],[*,T],[*,A]] ++ [[@,@]]	
single strands labeled	5'P-ATCGAT-3'B	[[P,@]] ++ [[A,*],[T,*],[C,*],[G,*],[A,*],[T,*]] ++ [[B,@]]	
	3'B-TAGCTA-5'P	[[@,B]] ++ [[*,T],[*,A],[*,G],[*,C],[*,T],[*,A]] ++ [[@,P]]	
double strands unlabeled	5'-ATCGCA-3' 3'-TAGCGT-5'	[[@,@]] ++ [[A,T],[T,A],[C,G],[G,C],[C,G],[A,T]] ++ [[@,@]]	
	5'-TGCGAT-3' 3'-ACGCTA-5'	[[@,@]] ++ [[T,A],[G,C],[C,G],[G,C],[A,T],[T,A]] ++ [[@,@]]	
double strands labeled	5'P-ATCGCA-3' 3'-TAGCGT-5'B	[[P,@]] ++ [[A,T],[T,A],[C,G],[G,C],[C,G],[A,T]] ++ [[@,B]]	
	5'B-TGCGAT-3' 3'-ACGCTA-5'P	[[B,@]] ++ [[T,A],[G,C],[C,G],[G,C],[A,T],[T,A]] ++ [[@,P]]	

- Synthesis

The function `syn :: [Char] -> Tube` creates a DNA single strand from the input base sequence. The tube contains two single strand lists, one in sense and another one with inverted sequence in antisense direction. The heads of the sense list elements are A, T, C or G. All heads of the antisense list elements have \*-entry.

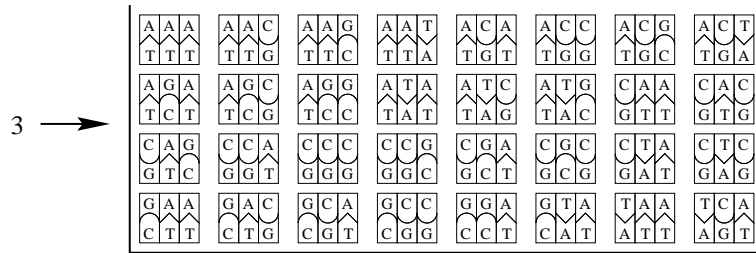


- Initialization

The function `in :: Int -> Tube` is executed as function  
`ini :: Int -> Tube -> Tube`.

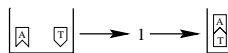
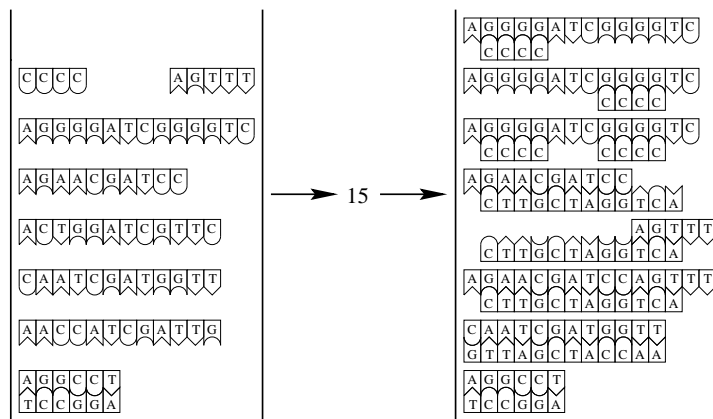
All possible permutations of DNA double strands with the defined length can be generated within the tube. The function creates all list permutations from the base pair lists `[A,T]`, `[T,A]`, `[G,C]` and `[C,G]`. The integer number defines the lengths of the final permuted strand lists. The example uses `k = 3`.

<sup>1</sup>Compilers for HASKELL were developed in Yale, Glasgow and Göteborg and are free in use. DNA-HASKELL will run on the HASKELL interpreter HUGS (HASKELL User's Gopher System from the University of Nottingham).



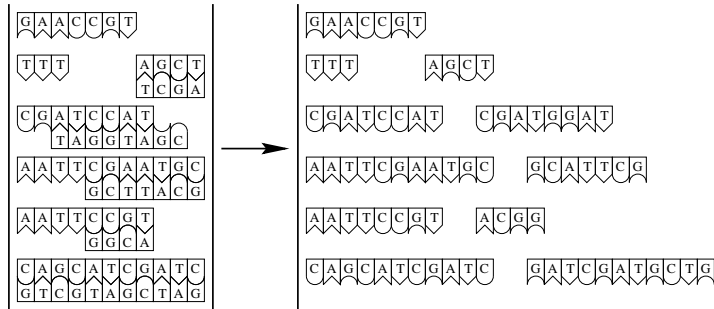
- Annealing

The function `ann :: Tube -> Int -> Tube` simulates the biological operation annealing. Single strands and double strands with sticky ends anneal to each other only when they are complementary. If so, they are forming double strands. All combinations are generated. The simplest form is the annealing of two nucleotides that results in a base pair. The function needs an integer number that limits the length of the new annealed strands.



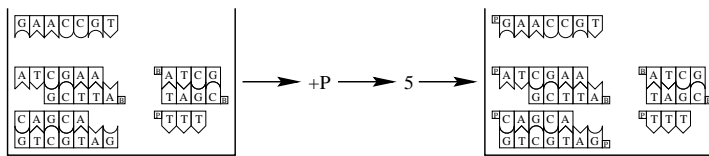
- Melting

The function `mel :: Tube -> Tube` is related to the function annealing because it represents the inversion of the biological annealing effect. Single strands will not be changed by melting. Double strands are separated into their single strand counterparts. After executing this operation the tube will contain only single strands.



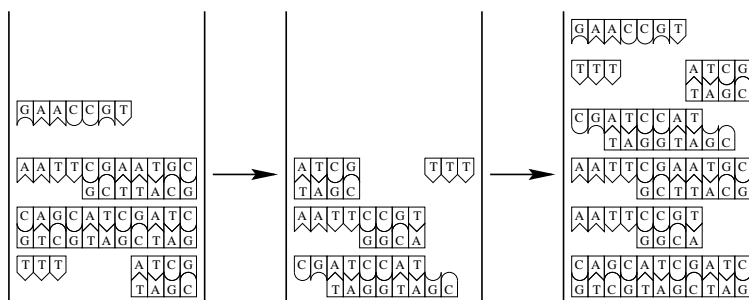
- Labeling

The function `lab :: Tube -> [Char] -> Int -> Tube` sets or removes certain labels on strand ends. The string characterizes the label and the integer number defines which strand end should be labeled. Possible labels are +/- P (Phosphorylating/ Dephosphorylating), +/- B (Biotinylating/ Debiotinylating) und +/- C (set Cy5-label/ remove Cy5-label).



- Union

The function `un :: Tube -> Tube -> Tube` combines the components of either tubes. Identical strands are taken only once.



- Ligation

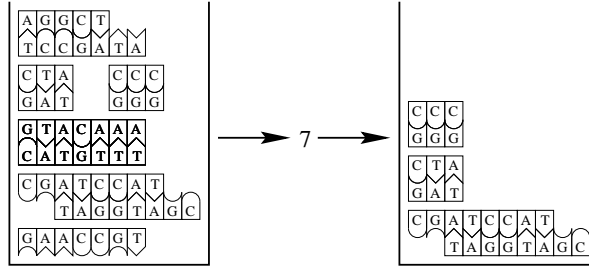
The function `li :: Tube -> Int -> Tube` simulates the biological ligation. All double strands inside the tube can be linked to itself or to another double strand under following conditions: The strands have compatible complementary ends and at least one of the connected strands has to be modified by 5'-phosphorylation. The generation of new concatenated strands will continue until the defined maximum in length is reached.





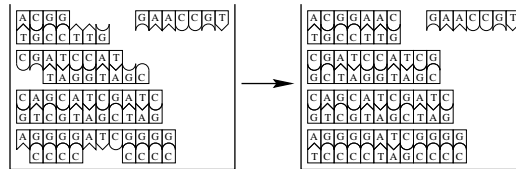
- FilterLength

The function `flt1 :: Tube -> Int -> Tube` filters all strands that vary from the defined length. These strands are collected in the resulting tube.



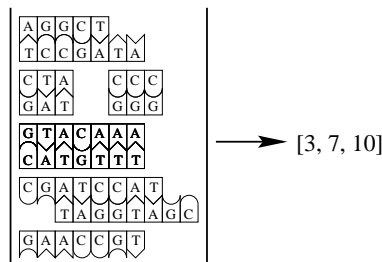
- Blunting

Double strands with protruding ends are converted by the function `blnt :: Tube -> Tube`, at their ends to strands carrying blunt ends. This operation describes the filling in of recessed 3'-ends and the removal of protruding 3'-ends.



- Electrophoresis

The electrophoresis is formalized by the typing `elt :: Tube -> [Int]`. It sorts all strands of one tube according to length. The function returns a list with all appropriate lengths.



## 3 The complexity of DNA-HASKELL

### 3.1 Objective

This section evaluates DNA-HASKELL operations from the complexity theoretical point of view. The time complexity will be determined. The following model implementations are compared:

- implementation on an SISD architecture (HASKELL-program using a Von-Neumann-computer) vs.
- implementation on a massive parallel SIMD architecture (execution of the operations in the laboratory).

Furthermore, we point out the real time that is required for executing DNA-HASKELL operations in the laboratory.

### 3.2 Problem sizes

$n$ : number of base pairs in the tube  
 $n$  is the key parameter of problem size. All other quantities used are auxiliary parameters.

$p$ : number of DNA strands in the tube  
Single strands and double strands are counted.

$l$ : parameter of length  
 $l$  is a natural number that is required as an additional input parameter in some operations.

It is valid:  $p \sim n$ . That means  $p$  and  $n$  only vary in a constant factor in each tube:  $p \cdot c = n$ . Thus, a DNA strand has an average length of  $c = \frac{n}{p}$  base pairs.

### 3.3 Implementation of DNA-HASKELL on a SISD architecture (simulation using HASKELL)

#### 3.3.1 Time steps

- one recursive function call
- one comparison to base pairs
- one comparison to label pairs
- replace one base pair by another base pair
- replace one label pair by another label pair
- one addition of natural numbers

#### 3.3.2 Useful operations for help

**Length determination** of one DNA strand

The whole strand has to be examined once requiring  $O(\frac{n}{p}) = O(c) = O(1)$  additions.

$\rightarrow O(1)$

**Match:** Is one given DNA strand a part of another one?

Let  $c_1$  base pairs the length of one strand and  $c_2$  base pairs the length of the other one ( $c_1, c_2$  constant). Thus,  $c_1 \cdot c_2$  comparisons to base pairs are required in the worst case.  $O(c_1 \cdot c_2) = O(1)$ .

→  $O(1)$

**Concatenation test:** Are two given DNA strands concatenable?

Let  $c_1$  base pairs the length of one strand and  $c_2$  base pairs the length of the other one ( $c_1, c_2$  constant). Thus,  $c_1 \cdot c_2$  comparisons to base pairs and one comparison to label pairs are required in the worst case.  $O(c_1 \cdot c_2 + 1) = O(1)$ .

→  $O(1)$

**Element test:** Does the tube contain a certain DNA strand?

The match operation has to be executed  $p$  times. That means:  $p \cdot O(1)$ . It follows:

→  $O(n)$

**Reverse:** Generating of the reverse DNA strand corresponding to each DNA strand in the tube

To execute this function all  $p$  DNA strands in the tube have to be examined once. That means  $O(p)$ . The consequence is:

→  $O(n)$

**Member test:** remove all identical DNA strands that appear more than once in the tube (including all reverse identical strands)

Each strand and its reverse counterpart must be compared with all other strands and their reverse counterparts. This function requires  $(2p) \cdot (2p-1)$  comparisons between DNA strands and results in the complexity:

→  $O(n^2)$

### 3.3.3 Synthesis

Two single strands (one and its reverse counterpart) have to be synthesized and put into an empty tube. According to the definition of problem size  $O(n)$  recursive function calls "Go to the next base pair" are necessary.

→  $O(n)$

### 3.3.4 Union

The union operation copies all  $p$  DNA strands of both input tubes into the resulting tube in  $p \cdot O(1)$  time steps. Subsequently, the member test is executed to ensure that no duplicates of DNA strands remain in the resulting tube. The member test with complexity  $O(n^2)$  determines the time complexity of union.

→  $O(n^2)$

### 3.3.5 Labeling

All  $p$  DNA strands in the input tube are examined exactly once during labeling. The steps "comparison on a label pair" and "exchange of a label pair" must be executed  $p$  times. This results in:

→  $O(n)$













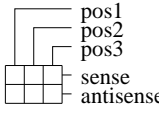








### 3.3.6 Annealing

The input tube contains  $p$  DNA strands. Strands with a maximum length of  $l$  base pairs can be produced by annealing. In principle, any of the  $p$  strands can anneal

in sense and antisense direction beginning at each of the  $l$  positions. There are  $2^{2pl}$  possibilities for annealing that have to be checked. These possibilities can be listed in a binary table. The test whether or not two strands are able to anneal requires  $O(1)$  time steps. The complexity  $O(2^{2pl})$  results in  $2^{2pl}$  possibilities. That means:  
 $\rightarrow O(2^{l \cdot n})$

example:

input tube:   $p = 2$   
 $l = 3$

2*p*l places											possibility to anneal	
 pos1	 pos2	 pos3	 pos1	 pos2	 pos3	 pos1	 pos2	 pos3	 pos1	 pos2	 pos3	
0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	1	
⋮												
0	0	0	1	0	0	1	0	0	0	0	0	
⋮												
0	0	0	1	0	1	0	0	0	0	0	0	
⋮												
0	1	0	1	0	0	0	0	0	0	0	0	
⋮												
0	1	0	1	1	0	0	0	0	0	0	0	
⋮												
1	0	0	1	0	0	0	0	0	0	0	0	
⋮												
1	1	1	1	1	1	1	1	1	1	1	1	

### 3.3.7 Ligation

The time complexity of ligation is determined by the number of concatenation tests. The simulation of ligation runs at several stages. At first, all  $p$  DNA strands of the input tube and their reverse counterparts are checked for their concatenability and joint if possible:

The  $2p$  strands of the input tube including their reverse counterparts and the maximum  $(2p)^2$  new strands of the first stage form the set of DNA strands for the next stage. The ligation is terminated when all concatenation tests are negative or only DNA strands with more than  $l$  base pairs are generated. The worst case



consists of  $(l - 1)$  stages if there is a 1bp strand in the tube. One can describe the number of concatenation tests by  $\sum_{i=2}^l (2p)^i$ . It is valid:

$$\sum_{i=2}^l (2p)^i = \frac{(2p)^{l+1} - 1}{2p - 1} - 2p - 1$$

The ligation has the time complexity:  
 $\rightarrow O(n^l)$

### 3.3.8 Cut

Finding the restriction site and its cutting position from the database of restriction enzymes needs  $O(1)$  time steps using a hash table for instance. After that, the input tube is matched  $n$  times for the restriction site. Possible cleavages are executed in  $O(n)$ . Finally, the duplicated DNA strands are removed using the member test. Its time complexity  $O(n^2)$  defines the cleavage operation.  
 $\rightarrow O(n^2)$

### 3.3.9 CuttingOut

To execute the excise operation the lengths of all  $p$  DNA strands in the input tube must be estimated. This requires  $p \cdot O(1)$  time steps. During this process all strands with the given length are copied into the resulting tube. CuttingOut has the time complexity:  
 $\rightarrow O(n)$

## 3.4 Implementation of DNA-HASKELL on a massive parallel SIMD architecture

A molecularbiological laboratory is considered as massive parallel SIMD architecture ("biohardware"). The described operations run there as biochemical reactions. An execution protocol exists for each operation. Every operation represents one lab step with the corresponding time complexity  $O(1)$ . Furthermore, the complexity of lab operations is characterized by the number of involved DNA strands and the reaction volume.

## 3.5 Comparison on the implementations

operation	time complexity in SISD simulation	time complexity in SIMD model (lab steps)	real time required in the laboratory
Synthesis	$O(n)$	$O(1)$	depends on the lengths of the strands
Annealing	$O(2^{l \cdot n})$	$O(1)$	4h
Union	$O(n^2)$	$O(1)$	0,1h
Labeling	$O(n)$	$O(1)$	2h
Cut	$O(n^2)$	$O(1)$	3h
CuttingOut	$O(n)$	$O(1)$	3h
Ligation	$O(n^l)$	$O(1)$	12h

The table shows that annealing and ligation in the laboratory are particularly more time efficient as opposed to the SISD version. These operations consume exponential time in simulation.

## 4 The solution of an integer knapsack problem with DNA-HASKELL

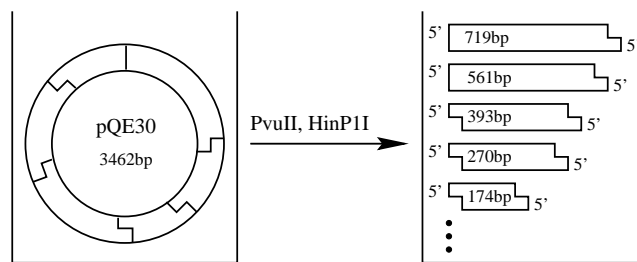
The integer knapsack problem belongs to the class of NP complete problems. There are  $n$  objects with the weights  $a_i \in \mathbb{N}$ ,  $1 \leq i \leq n$  and a number  $b \in \mathbb{N}$ . The integer knapsack problem is based on the decision whether or not a subset  $I \subseteq \{1, 2, \dots, n\}$  exists under the condition  $\sum_{i \in I} a_i = b$ . The parameter  $n$  represents the problem size.

$n$  objects allow  $2^n$  possibilities to pack the knapsack.

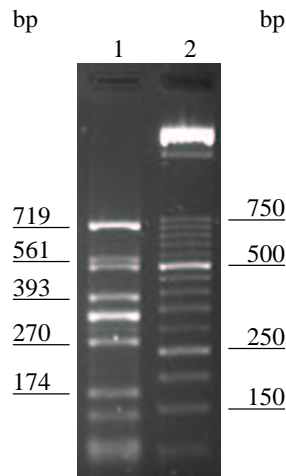
For example, to implement in DNA-HASKELL we use the integer knapsack problem with two objects, their weights  $a_1 = 719$ ,  $a_2 = 393$ , and  $b = 1112$ .

The object weights are encoded by DNA double strands with lengths according to the weights. The plasmid pQE30 forms the basic material. It is a ring-shaped double stranded DNA and can be isolated from bacteria in the laboratory. pQE consists of 3462 base pairs (bp) with well-known sequence.

For extracting DNA fragments with 719bp and 393bp from the plasmid it is cleaved by the restriction enzymes PvuII and HinP1I. PvuII is a blunt end cutter. Its restriction site appears only once in pQE30. The restriction site of HinP1I – a sticky end cutter – exists on several different positions in pQE30. A subset of DNA is not processed:

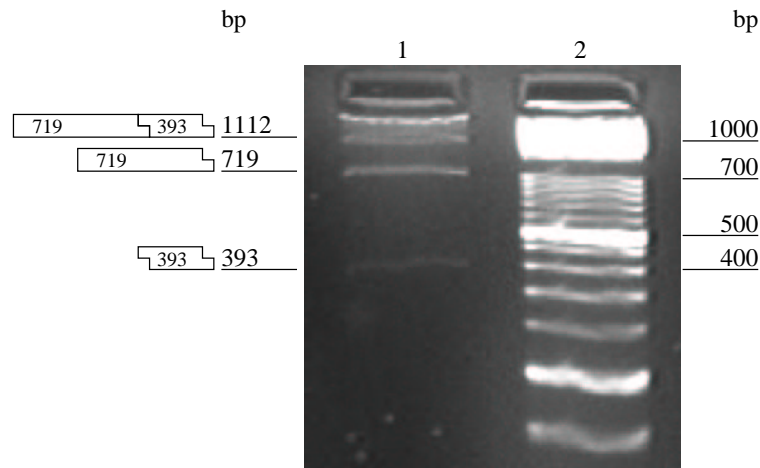


To further control the ligation reactions 5' ends of the DNA fragments must be dephosphorylated. This step is executed immediately after cleaving. The strands are separated by length through an agarose gel electrophoresis (lane1, digestion product; lane2, 50bp DNA molecular weight marker):



The bands with 719bp and 393bp are excised from the gel and the DNA is extracted.

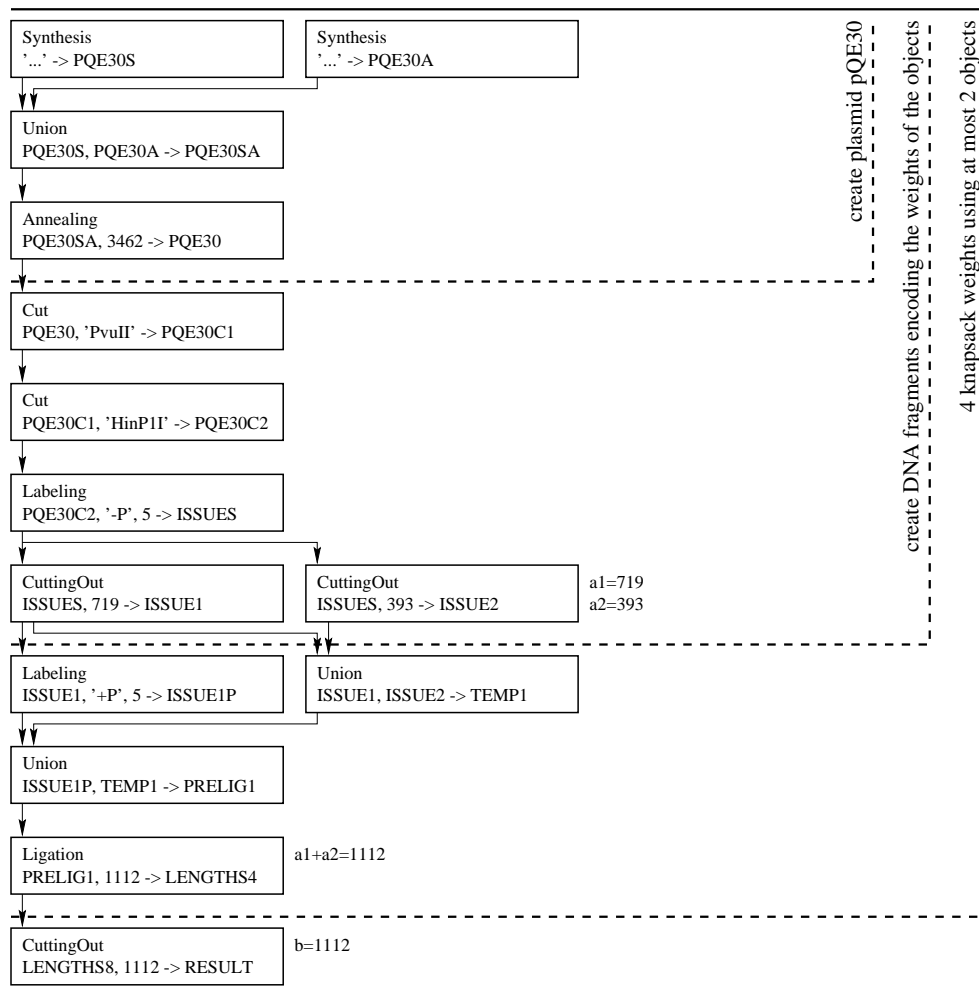
The following step produces all possibilities to pack the knapsack using the objects with the weights  $a_1$  and  $a_2$ . That means that DNA fragments with the lengths 0bp, 719bp, 393bp, and 719+393bp should be created. The ligation concatenates end compatible DNA double strands. The condition for concatenating is that at least one of the two strands that is to be ligated is phosphorylated at the 5' end. For this purpose, an aliquot from the tube with the 719bp fragments is phosphorylated at the 5' end. After that, the ligation runs with the dephosphorylated 719bp fragments, the 5' phosphorylated 719bp fragments and the dephosphorylated 393bp fragments. This ensures that only one 393bp fragment can be joint to one 719bp fragment to avoid the generation strands with more than one 393bp fragment. Fragments with 719+719bp, 719+719+393bp and longer can also appear but they are outside the considered interval. This ligation produces DNA strands with exactly these lengths representing the knapsack's weights of all packing possibilities except the empty knapsack. The bands can be visualized by agarose gel electrophoresis (lane1, ligation product; lane2, 50bp DNA molecular weight marker):



Further researching laboratory activities focus on an increasement of problem size and on the optimization of molecular biological techniques to use.



The following flowchart describes the algorithm to solve an integer knapsack problem in DNA-HASKELL. The boxes contain the name of the operation (first line) and the parameters used (e.g. reaction tube name) according to the specification (second line). Executing the whole process in the laboratory needs nearly 40 hours:



In case the tube **RESULT** is empty, no packing possibility with  $b = 1112$  exists. **RESULT** contains DNA strands that encode the packing possibility "object1 and object2". That's why the solution of the integer knapsack problem is "yes".

The number of required biochemical operations increases linearly in the number  $n$  of available objects.

## References

- [Adl94]  
L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021-1024, November 1994.
- [Adl95]  
L.M. Adleman. On constructing a molecular computer. [ftp://usc.edu/pub/csinfo/papers/adleman/molecular\\_computer.ps](ftp://usc.edu/pub/csinfo/papers/adleman/molecular_computer.ps), University of Southern California, January 1995.
- [Adl98]  
L.M. Adleman. Computing with DNA. *Scientific American*, pages 34 – 41, August 1998.
- [Amo96]  
M. Amos, A. Gibbons, D. Hodgson. Error-resistant implementation of DNA computations. <http://www.csc.liv.ac.uk/~martyn/princeton.ps>, 1996.
- [Amo97]  
M. Amos, A. Gibbons, P.E. Dunne. The complexity and viability of DNA computations. <http://www.csc.liv.ac.uk/~ctag/archive/t/CTAG-97001.ps>, 1997.
- [Bac96]  
E. Bach, A. Condon, E. Glaser, C. Tanguay. DNA Models and Algorithms for NP-complete Problems, *IEEE Computer Society Press*, pages 290 – 299, 1996.
- [Ber92]  
P. Berg, M. Singer. Die Sprache der Gene. Grundlagen der Molekulargenetik. *Spektrum, Akademischer Verlag*, 1992.
- [Bon96a]  
D. Boneh, C. Dunworth, J. Sgall. On the computational power of DNA. *Discrete Applied Mathematics*, 71(1-3):79–94, 1996.
- [Bon96b]  
D. Boneh, R.J. Lipton. Making DNA computers error resistant. <ftp://ftp.cs.princeton.edu/pub/people/dabo/bioerror.ps.Z>, 1996.
- [Bov94]  
D.P. Bovet, P. Crescenzi. Introduction to the Theory of Complexity. *Prentice Hall*, New York, London, Toronto, Sydney, Tokyo, Singapore, 1994.
- [Das98]  
J.H.M. Dassen. A Bibliography of Molecular Computation and Splicing Systems. <http://www.wi.leidenuniv.nl/~jdassen/dna.html>, September 1998.
- [Gua96]  
F. Guarnieri, M. Fliss, C. Bancroft. Making DNA add. *Science*, 273(5272):220–223, July 1996.
- [Hea87]  
T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49(6):737–759, 1987.

- [Kap96]  
P.D. Kaplan, G. Cecchi, A. Libchaber. DNA based molecular computation: template-template interactions in PCR, 1996.
- [Kar97]  
L. Kari. From Micro-soft to Bio-Soft: Computing with DNA.  
<http://www.csd.uwo.ca/~lila>, September 1997.
- [Kri94]  
P. Karlson, D. Doenecke, J. Koolman. Biochemie für Mediziner und Naturwissenschaftler. *Georg Thieme Verlag Stuttgart*, 1994.
- [Lip95a]  
R.J. Lipton. Using DNA to solve NP-complete problems.  
*Technical report, Princeton University*, 1995.
- [Lip95b]  
R.J. Lipton. Using DNA to solve SAT.  
<http://www.cs.princeton.edu/~dabo/bio-comp/sat.ps>, 1995.
- [Lip95c]  
R.J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, April 1995.
- [Ouy97]  
Q. Ouyang, P.D. Kaplan, S. Liu, A. Libchaber. DNA solution of the maximal clique problem. *Science*, 278:446–449, 1997.
- [Pis97]  
N. Pisanti. A survey on DNA computing. *Technical Report TR-97-07, Università di Pisa*, April 1997.
- [Rei95]  
J.H. Reif. Parallel molecular computation: Models and simulations. *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA95), Santa Barbara*, pages 213–223, June 1995.
- [Roo95]  
D. Roß, K.W. Wagner. On the power of DNA-computers. *Technical report, University of Würzburg*, 1995.
- [Wat92]  
J. Watson, M. Gilman, J. Witkowski, M. Zoller. Recombinant DNA. *Scientific American Books*, New York, NY, 2nd edition, 1992.
- [Win95]  
E. Winfree. On the computational power of DNA annealing and ligation.  
<http://dope.caltech.edu/winfree/Papers/ligation.ps.gz>, 1995.