# Phase Transition and the Computational Complexity of Generating $r$-contiguous Detectors

Thomas Stibor

Department of Computer Science
Darmstadt University of Technology
*stibor@sec.informatik.tu-darmstadt.de*

**Abstract.** The problem of generating $r$-contiguous detectors in negative selection can be transformed in the problem of finding assignment sets for a Boolean formula in $k$-CNF. Knowing this crucial fact enables us to explore the computational complexity and the feasibility of finding detectors with respect to the number of self bit strings $|\mathcal{S}|$, the bit string length $l$ and matching length $r$. It turns out that finding detectors is hardest in the phase transition region, which is characterized by certain combinations of parameters $|\mathcal{S}|, l$ and $r$. This insight is derived by investigating the $r$-contiguous matching probability in a random search approach and by using the equivalent $k$-CNF problem formulation.

## 1 Introduction

Theoretical immunologists have proposed the $r$-contiguous matching function to abstract the affinity between an antibody and an antigen in immune system models [1]. In the field of artificial immune systems, the $r$-contiguous matching function is applied as a matching rule for change detection [2] or more generally for anomaly detection problems. In these domains, antibodies (called detectors) and antigens are abstracted as bit strings and the $r$-contiguous matching rule is applied for detecting (anomalous) antigens. More specifically, in this immune inspired anomaly detection approach, the problem is to find detectors, such that no detector match with any self antigen. This form of detector generation for the complementary space is called *negative selection* [3].

In recent years, many attempts were made (see [4,5] for an overview) to generate detectors efficiently, i.e. in polynomial time and with polynomial space occupation with regard to the matching length $r$ and number of self antigens $|\mathcal{S}|$. All attempts in designing efficient algorithms for generating $r$-contiguous detectors were limited successful. The proposed algorithms either have a time or a space complexity which is exponential[1] in the matching length $r$, i.e. $\mathcal{O}(2^r)$ or in the number of self elements $|\mathcal{S}|$, i.e. $\mathcal{O}(e^{|\mathcal{S}|})$. Stibor et al. [6] proved that the

[1] There exists a *linear time detector generating algorithm* [2], however this algorithm still requires $\mathcal{O}(2^r)$ time and space occupation. It is termed linear, because it runs linear in $|\mathcal{S}|$ under the *assumption* that $|\mathcal{S}| = \mathcal{O}(2^r)$. For real-world problems however, the assumption $|\mathcal{S}| \ll 2^r$ is justifiable.

problem of generating $r$-contiguous detectors can be transformed in a $k$-CNF satisfiability problem and argued that at least $\Omega(2^r)$ bit string evaluations are required to find *all* $r$-contiguous detectors.

In this paper we go one step further and explore the computational complexity of generating detectors with the Davis-Logemann-Loveland algorithm. Furthermore, we rigorously analyze, when detectors can be generated with respect to the number of self bit strings $|\mathcal{S}|$, the bit string length $l$ and matching length $r$. It will turn out that generating $r$-contiguous detectors is computationally *not* equally "hard". More specifically, it is relatively cheap computationally, to verify that *no* detectors can be generated or that a large number of detectors can be generated. However, there also exists a *phase transition* region which is characterized by certain combinations of parameters $|\mathcal{S}|, l$ and $r$ where finding detectors is hardest. This insight will be derived from two directions, namely by investigating the $r$-contiguous matching probability in a random search approach and by using the problem transformation of generating detectors into the $k$-CNF satisfiability problem.

## 2 Bit String Matching Rule and Generating Detectors Randomly

Let $\mathcal{U}$ be a universe which contains all $2^l$ distinct bit strings of length $l$.

**Definition 1.** *A bit string $b \in \mathcal{U}$ with $b = b_1 b_2 \dots b_l$ and detector $d \in \mathcal{U}$ with $d = d_1 d_2 \dots d_l$, match with $r$-contiguous rule, if a position $p$ exists where $b_i = d_i$ for $i = p, \dots, p + r - 1$ and $p \leq l - r + 1$.*

Loosely speaking, two bit strings, with the same length, match if at least $r$ contiguous bits are identical. In the remaining sections the expression "detectors" will refer to $r$-contiguous detectors. Sets are denoted in calligraphic letters, e.g. $\mathcal{S}$ and $|\mathcal{S}|$ denotes the cardinality. Throughout the paper, we will assume that $\mathcal{S}$ contains pairwise distinct bit strings randomly drawn from $\mathcal{U}$.

### 2.1 Randomly Generating Detectors in Negative Selection

Given $\mathcal{U}$ and its partition into distinct subsets $\mathcal{S}$ and $\mathcal{N}$. In negative selection one has to find detectors such that no detector matches (see Def. 1) with any bit string from $\mathcal{S}$. Detectors which satisfy this property match with — not necessarily all — bit strings from the complementary space $\mathcal{U} \setminus \mathcal{S}$. Algorithm (1) is a straightforward random search to generate, i.e. to find detectors. A bit string $d$ is randomly sampled from $\mathcal{U}$ and matched against all bit strings in $\mathcal{S}$. When no $r$-contiguous match occurs, $d$ is added to the detector set $\mathcal{D}$. This random sampling is repeated until a certain number of detectors is found. It is obvious that this straightforward random search is not an efficient search technique. However, a thorough probabilistic analysis of algorithm (1) reveals valuable insights, whether detectors can or can not be generated.

**Algorithm 1**: Random search for detectors in negative selection

**input** : $l, r, t \in \mathbb{N}$ where $1 \leq r \leq l$ and $\mathcal{S} \subset \mathcal{U}$
**output**: Set $\mathcal{D} \subset \mathcal{U}$ of $r$-contiguous detectors

```
1 begin
2     D := ∅
3     while |D| < t do
4         Sample randomly a bit string d ∈ U
5         if d does not match with any bit string of S then
6             D := D ∪ {d}
7 end
```

### 2.2 Probability of Matching in Random Detector Generation

The probability that two randomly drawn bit strings from $\mathcal{U}$ are *not* matching with the $r$-contiguous rule can be determined with approaches from probability theory, namely recurrent events and renewal theory [7]. In Feller's textbook [7] on probability theory an equivalent[2] problem is formulated as follows:

> "*A sequence of $n$ letters $S$ and $F$ contains as many $S$-runs of length $r$ as there are non-overlapping uninterrupted blocks containing exactly $r$ letters $S$ each*".

Given a Bernoulli trial with outcomes $S$ (success) and $F$ (failure), the probability of no success running of length $r$ in $l$ trials is according to Feller

$$P_{WF} = \frac{1 - px}{(r + 1 - rx)q} \cdot \frac{1}{x^{l+1}} \tag{1}$$

where

$$p = q = \frac{1}{2} \quad \text{and} \quad x = 1 + qp^r + (r+1)(qp^r)^2 + \ldots$$

A simpler approximation — however only valid for $r \geq l/2$ — is provided by Percus et al. [1]:

$$P_{JP} = 1 - 2^{-r}\left[(l - r)/2 + 1\right]. \tag{2}$$

From (1) one can straightforwardly conclude that the probability of finding $t$ detectors when given $l, r$ and $|\mathcal{S}|$ results in:

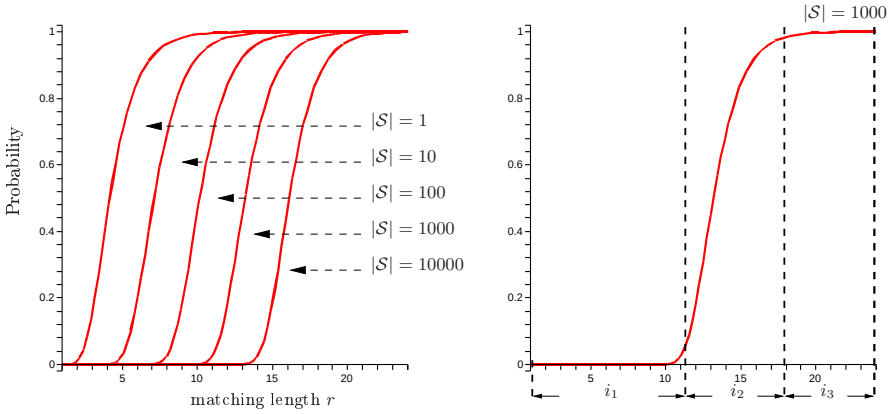
$$\textbf{Prob}[\text{find } t \text{ detectors}] = t^{-1} \cdot (P_{WF})^{|\mathcal{S}|}. \tag{3}$$

[2] The Link between recurrent events, renewal theory and the $r$-contiguous matching rule was discovered originally by Percus et al. [1] and rediscovered by Ranang [8]. Percus et al. presented in [1] the approximation (2) which is only valid for $r \geq l/2$, but mentioned the full approximation for $1 \leq r \leq l$ indirectly by mentioning the name de Moivre and citing Uspensky's textbook (see pp. 77 in [9]).

Moreover, from (3) one can conclude how often on average step 4 in algorithm (1) is executed when given $t$, or in other words how many bit strings one has to sample before finding $\widehat{t}$ detectors.

$$\widehat{t} = \frac{1}{t^{-1} \cdot P_{WF}^{|\mathcal{S}|}}. \tag{4}$$

Result (4) is equivalent to an earlier result on negative selection [3], when $P_{WF}$ is replaced by $P_{JP}$.




(a) Matching probability for finding a detector randomly for $l := 24$, $r := [1 \ldots 24]$ and $|\mathcal{S}| := \{1, 10, 100, 1000, 10000\}$.

(b) If $r$ lies within interval $i_1$, then with high probability no detectors will be found, whereas if $r$ lies within interval $i_3$, then with high probability, detectors will be found. There also exists an interval $i_2$ where the probability rapidly changes from 0 to 1.

**Fig. 1.** Coherence between the probability of finding a detector randomly and the parameters $l, r$ and $|\mathcal{S}|$. There exists a sharp transition boundary where the probability rapidly changes from 0 to 1.

### 2.3 Probability Transition in $r$-contiguous Matching

Knowing the probability $P_{WF}$ enables us to investigate the combinations of parameters $|\mathcal{S}|, l$ and $r$ where, with high probability detectors can be generated (i.e. exist) or with high probability can not be generated. The graphs in figure 1 show the probability for finding a detector for fixed $l$ and variable $r$ and $|\mathcal{S}|$ according to term (3). One can see, that the larger the cardinality of $\mathcal{S}$, the larger the interval for $r$ where the resulting probability is nearly 0 to find a detector, or in other words where no detectors exist. On the other hand, the

smaller the cardinality of $\mathcal{S}$, the larger the interval for $r$ where the resulting probability is nearly 1 to find a detector. In figure 1(b) the same graph, but only for $|\mathcal{S}| = 1000$ is highlighted. One can see in detail that three different intervals $(i_1, i_2, i_3)$ exist. One can either find with high probability a detector in interval $i_1$, or find with high probability no detector in interval $i_3$. Moreover there exists a third interval $i_2$ where the probability rapidly changes from 0 to 1. For the sake of conformity with the subsequent sections, we denote the interval $i_2$ as *phase transition* region. We will later see, that finding detectors in this region, which is characterized by certain combinations of parametes $|\mathcal{S}|, l, r$ is hardest from the perspective of computationally complexity.

To summarize this section, if parameters $|\mathcal{S}|, l$ and $r$ are chosen such that term (3) results in a value very close to 0, then in the worst case no detectors can be generated, never mind which algorithms, i.e. search techniques are applied to generate detectors, because there exist no detectors. On the other hand, if term (3) is close to 1, then a large number of detectors exist.

### 2.4 Coherence of Matching Length $r$, Self Set $\mathcal{S}$ and Random Detector Search

In the artificial immune system community seems to exist some confusion regarding the time complexity of algorithm (1). More specifically, authors in [3] argued that generating detectors when applying the random search approach can be performed linearly in $|\mathcal{S}|$. Their argument is based on the observation that $\widehat{t}$ in (4) is minimized by choosing $1 - P_{JP} \approx 1/|\mathcal{S}|$. In other words, the number of bit strings one has to sample before finding $t$ detectors is linear proportionally to $|\mathcal{S}|$, when using algorithm (1). This observation implies that the matching threshold $r$ purely depends on the cardinality of $S$ when $l$ is fixed. To be more precise, suppose $r \geq l/2$, then

$$2^{-r}\left[(l-r)/2+1\right] \approx |S|^{-1} \Longleftrightarrow r \approx l + 2 - \frac{W(8\ln(2)2^l/|S|)}{\ln(2)} \tag{5}$$

where $W(x)$ is the Lambert $W$-function which can be expressed as the series expansion

$$W(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1}k^{k-2}}{(k-1)!}x^k. \tag{6}$$

Practically speaking, once $|S|$ and $l$ are fixed, the matching length $r$ is such chosen that it will fall in interval $i_3$ (see Fig. 1(b)) and consequently this implies that a large number of detectors can be generated.

With regards to anomaly detection problems, it is known [10,11,12] that the $r$-contiguous matching rule is a positional biased detection rule. That means that the value of $r$ is inextricably linked to the underlying data being analyzed. The assumption $1 - P_{JP} \approx 1/|\mathcal{S}|$ however, implies that $r$ grows with $|\mathcal{S}|$ (see term (5)), and does not consider the positional bias. On the other side, if $l$ and $r$

are fixed[3] and $|\mathcal{S}|$ is considered as the variable parameter then $\widehat{t} = \mathcal{O}(e^{|\mathcal{S}|})$, that is, $r$ will lie within interval $i_1$ for some large $|\mathcal{S}|$ and this consequently implies that a random search for detectors results in an exponential time complexity — when detectors exist.

### 2.5 Average Number of Detectors and Holes

For the sake of completeness, we present results on the average number of detectors that can be generated and the resulting holes. The results are straightforward conclusions from the previous section 2.2.

Recall, algorithm (1) fails to find any detector when a certain parameter combination of $\mathcal{S}, l$ and $r$ exists. More specifically, the universe $\mathcal{U}$ is not only composed of sets $\mathcal{S}, \mathcal{D}$ and $\mathcal{N}$, but also of set $\mathcal{H}$. Recall, the set $\mathcal{N}$ contains all bit strings which are detectable by the detectors from $\mathcal{D}$ and hence $\mathcal{D} \subset \mathcal{N}$. The set $\mathcal{H}$, termed hole set contains all bit strings which are not detectable by any detector, however, $\mathcal{H}$ does not contain any bit strings from $\mathcal{S}$, i.e. $\mathcal{H} \cap \mathcal{S} = \emptyset$ and hence, $|\mathcal{H}|$ is directly linked with interval $i_1$ (see Fig.1(b)). More specifically, if a parameter combination of $l, r$ and $\mathcal{S}$ is chosen such that term (3) is very close to 0, then $|\mathcal{N}| \ll |\mathcal{H}|$ or in the extreme case $|\mathcal{N}| = 0$, i.e. the universe $\mathcal{U}$ is only composed of sets $\mathcal{S}$ and $\mathcal{H}$.

Knowing this coherence between term (3) and the universe composition, the average number of detectors that can be generated results in

$$\mathbf{E}[|\mathcal{D}|] = 2^l \cdot (P_{WF})^{|\mathcal{S}|}. \tag{7}$$

As the universe is composed of $\mathcal{U} = \mathcal{S} \cup \mathcal{N} \cup \mathcal{H}$ when applying the negative selection, the number of holes results in

$$|\mathcal{H}| = |\mathcal{U}| - |\mathcal{N}| - |\mathcal{S}| \tag{8}$$

where

$$\mathbf{E}[|\mathcal{N}|] = 2^l \quad - \quad \underbrace{2^l \cdot (P_{WF})^{\mathbf{E}[|\mathcal{D}|]}}_{\substack{\text{Number of bit strings} \\ \text{not detected by } \mathbf{E}[|\mathcal{D}|] \\ \text{detectors}}} \tag{9}$$

and hence the average number of holes results in

$$\mathbf{E}[|\mathcal{H}|] = 2^l \cdot (P_{WF})^{\mathbf{E}[|\mathcal{D}|]} - |\mathcal{S}|. \tag{10}$$

## 3 Link between $r$-contiguous Detectors and $k$-CNF Satisfiability

Stibor et al. [6] proved that the problem of generating detectors in negative selection can be transformed in an equivalent problem of finding assignment sets

[3] To capture the semantical representation of the data being analyzed.

for a Boolean formula in $k$-CNF. Satisfying a Boolean formula in $k$-CNF is an instance of the satisfiability problem [13], where one has to decide if there is some assignment of *true* and *false* values that will make a Boolean formula in conjunctive normal form *true*. For the sake of clarity, we summarize the transformation steps presented in [6].

Let $b \in \{0,1\}$ and $\mathfrak{L}(b)$ a mapping defined as:

$$\mathfrak{L}(b) \rightarrow \begin{cases} x & \text{if} \quad b = 0 \\ \overline{x} & \text{otherwise} \end{cases}$$

where $x, \overline{x}$ are literals. Moreover, let $k, l \in \mathbb{N}$, where $k \leq l$ and $s \in \mathcal{U}$, where $s[i]$ denotes the bit at position $i$ of bit string $s$. A mapping from bit string $s$ into the $l$-$k$-CNF[4] is defined as follows:

$$\begin{aligned} \mathfrak{C}(s,k) \rightarrow & (\mathfrak{L}(s[1]) \ \vee \ \mathfrak{L}(s[2]) \ \vee \ \ldots \ \vee \ \mathfrak{L}(s[k])) \ \wedge \\ & (\mathfrak{L}(s[2]) \ \vee \ \mathfrak{L}(s[3]) \ \vee \ \ldots \ \vee \ \mathfrak{L}(s[k+1])) \ \wedge \\ & \vdots \\ & (\mathfrak{L}(s[l-k+1]) \ \vee \ \ldots \ \vee \ \mathfrak{L}(s[l])) \,. \end{aligned}$$

The resulting Boolean formula is constructed by an AND-combination of all bit strings in $\mathcal{S}$, i.e.

$$\widehat{\phi}_{rcb} := \mathfrak{C}(s_1,k) \ \wedge \ \mathfrak{C}(s_2,k) \ \wedge \ \ldots \ \wedge \ \mathfrak{C}(s_{|S|},k) \text{ for } s_i \in \mathcal{S}, \ i = 1, \ldots, |\mathcal{S}|$$

**Proposition 1 (Stibor et al. [6]).** *Given a universe $\mathcal{U}$ which contains all $2^l$ distinct bit strings of length $l$, a set $\mathcal{S} \subset \mathcal{U}$ and the set $\mathcal{D}$ which contains all generable $r$-contiguous detectors, which do not match any bit string from $\mathcal{S}$. The Boolean formula $\widehat{\phi}_{rcb}$ which is obtained by $\mathfrak{C}(s,r)$ for all $s \in S$ is satisfiable only with the assignment set $\mathcal{D}$.*

To summarize, instead of searching for detectors e.g. by means of algorithm (1), one can use SAT-Solvers [14] to find assignments sets of $\widehat{\phi}_{rcb}$. This crucial fact can be exploited for quantifying the computational complexity of finding detectors. However, one must estimate the *average number of distinct clauses* after applying the transformation steps, otherwise one would consider equal clauses several times — and this consequently would make the problem "harder" then it is.

### 3.1 Average Number of Distinct Clauses

Let $\mathcal{S}$ be a subset of $\mathcal{U}$ which contains pairwise distinct bit strings $s_1, s_2, \ldots, s_{|\mathcal{S}|}$ which are randomly drawn from $\mathcal{U}$. The constructed Boolean formula $\widehat{\phi}_{rcb}$ does not necessarily contains pairwise distinct clauses. Two clauses are distinct from each other, if they differ in at least one literal.

[4] The Boolean formula is denoted as $l$-$k$-CNF, because it is a special type of a $k$-CNF.

*Example 1.* Let $\mathcal{S} := \{0101, 1101\}$ and $r = 3$, hence $\widehat{\phi}_{rcb}$ results in

$$(x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_2 \vee x_3 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_2 \vee x_3 \vee \overline{x}_4).$$

Example 1 shows that the second and the fourth clause are equal, because the last three bits of 0101 and 1101 are equal.

**Proposition 2.** *Given bit string length $l$, matching length $r$ and let $\mathcal{S}$ be a subset of $\mathcal{U}$ which contains pairwise distinct bit strings $s_1, s_2, \ldots, s_{|\mathcal{S}|}$ randomly drawn from $\mathcal{U}$. The average number of pairwise distinct clauses is*

$$\mathbf{E}[|\widehat{\phi}_{rcb}|] = 2^r\,(l-r+1) - \left(1 - \frac{1}{(l-r+1)\,2^r}\right)^{|\mathcal{S}|(l-r+1)} (l-r+1)\,2^r. \quad (11)$$

*Proof.* Construct a lookup table $\mathfrak{T}$ which contains all $2^r \cdot (l-r+1)$ clauses with label $T$ and is of the form

| *clause* | *label* |
|---|---|
| $(x_1 \vee x_2 \vee \ldots \vee x_{r-1} \vee x_r)$ | $T$ |
| $(x_2 \vee x_3 \vee \ldots \vee x_r \vee x_{r+1})$ | $T$ |
| $\vdots$ | $\vdots$ |
| $(x_{l-r+1} \vee x_{l-r+2} \vee \ldots \vee x_{l-1} \vee x_l)$ | $T$ |
| $(x_1 \vee x_2 \vee \ldots \vee x_{r-1} \vee \overline{x}_r)$ | $T$ |
| $(x_2 \vee x_3 \vee \ldots \vee x_r \vee \vee \overline{x}_{r+1})$ | $T$ |
| $\vdots$ | $\vdots$ |
| $(x_{l-r+1} \vee x_{l-r+2} \vee \ldots \vee x_{l-1} \vee \overline{x}_l)$ | $T$ |
| $\vdots$ | $\vdots$ |
| $(\overline{x}_1 \vee \overline{x}_2 \vee \ldots \vee \overline{x}_{r-1} \vee \overline{x}_r)$ | $T$ |
| $(\overline{x}_2 \vee \overline{x}_3 \vee \ldots \vee \overline{x}_r \vee \overline{x}_{r+1})$ | $T$ |
| $\vdots$ | $\vdots$ |
| $(\overline{x}_{l-r+1} \vee \overline{x}_{l-r+2} \vee \ldots \vee \overline{x}_{l-1} \vee \overline{x}_l)$ | $T$ |

Transform $\mathcal{S}$ into the corresponding Boolean formula $\widehat{\phi}_{rcb}$ and set the label to $F$ whenever a clause in $\mathfrak{T}$ is member of $\widehat{\phi}_{rcb}$. As $\mathcal{S}$ is randomly drawn without replacement from $\mathcal{U}$, the $F$ and $T$ labels are binomially distributed in $\mathfrak{T}$. The probability of finding *no* clauses which are labeled with $F$ when randomly drawn $|\mathcal{S}| \cdot (l-r+1)$ clauses from $\mathfrak{T}$ results in

$$\left(1 - \frac{1}{(l-r+1)\,2^r}\right)^{|\mathcal{S}|(l-r+1)}$$

and hence, the total number of clauses with label $F$ results in

$$2^r\,(l-r+1) - \left(1 - \frac{1}{(l-r+1)\,2^r}\right)^{|\mathcal{S}|(l-r+1)} (l-r+1)\,2^r.$$

□

## 4 Computational Complexity of Generating Detectors

A common approach to quantify the computational "hardness" of an instance of a Boolean formula in $k$-CNF is to count the number of backtracking attempts in the Davis-Logemann-Loveland (DLL[5]) algorithm. The DLL algorithm [17] is based on the elimination rules proposed by Davis and Putnam [18] and terminates either with result unsatisfiable (empty clause) or satisfiable (empty $\phi$). More specifically, the algorithm is a depth-first search technique and uses recursive backtracking for guiding the exploration. The algorithm constructs a decision tree, where assignments of the variables coincide with paths from the root to the leafs. If a path leads to an unsatisfiable result, then the algorithm backs up to a different branch. This recursive search is reiterated until it terminates with a satisfiable or unsatisfiable result. In the worst case the whole decision tree has to be inspected, i.e. it will take an exponential number of evaluations — similar to an exhaustive search. However on average the DLL algorithm is much faster because it can prune whole branches from the decision tree without exploring the leaves.

Given a Boolean formula $\phi$ in CNF, a literal $l$ in $\phi$ and the reduction function $R(\phi, l)$ that outputs the residual formula of $\phi$ by:

- removing all the clauses that contain $l$,
- deleting $\overline{l}$ from all the clauses that contain $\overline{l}$,
- removing both $l$ and $\overline{l}$ from the list of literals.

A clause that contains one literal is called *unit clause*, and a literal $l$ is called *monotone*, if $\overline{l}$ appears in no clause of $\phi$. In lines 2-7 the reduction function is applied whenever a unit clause or a monotone literal is found. The subsequent recursive call is performed in lines 11, 13 respectively. "Easy" input instances imply that the DLL algorithm requires few backtracking attempts because clauses and literals can be efficiently eliminated by means of $R(\phi, l)$ without executing many subsequent recursive calls. On the other hand, "hard" instances imply that many recursive calls or backtracking attempts are required. In the next section, the terms "easy" and "hard" are clarified. More specifically, it will be shown that parameters $|\mathcal{S}|, l$ and $r$ specify the ratio of the number of clauses to variables of the $\widehat{\phi}_{rcb}$ instances and therefore characterize the computational complexity of the DLL algorithm.

### 4.1 Phase Transition in $k$-CNF Satisfiability

The $k$-CNF satisfiability problem is $\mathcal{NP}$-complete for $k > 2$, however, this fact does not imply that *all* instances of the $k$-CNF satisfiability problem are intractable to solve. In point of fact, there exists many problem instances which are "easy" to solve, i.e. one can efficiently decide whether the instance is satisfiable or is unsatisfiable. On the other hand there also exists problem instances

[5] The DLL algorithm is sometimes also called DPL or DPLL algorithm [15,16].

**Algorithm 2**: Davis-Logemann-Loveland algorithm (DLL($\cdot$))

```
   input  : φ (Boolean formula in CNF)
   output: SATISFIABLE or UNSATISFIABLE
 1 begin
 2     forall unit clauses {l} in φ do
 3         φ ← R(φ, l)
 4         if φ includes empty clause then
 5             return UNSATISFIABLE
 6     forall monotone literals l in φ do
 7         φ ← R(φ, l)
 8     if φ is empty then
 9         return SATISFIABLE
10     choose a literal l in φ
11     if DLL(R(φ, l)) = SATISFIABLE) then
12         return SATISFIABLE
13     if DLL(R(φ, l̄)) = SATISFIABLE) then
14         return SATISFIABLE
15     return UNSATISFIABLE
16 end
```

which are "hard", i.e. one can *not* efficiently decide whether the instance is satisfiable or is not satisfiable. The computational "hardness" of finding assignments sets for randomly generated instances is characterized by the ratio [19]

$$r_k = \frac{\text{number of clauses}}{\text{number of variables}}. \tag{12}$$

If the Boolean formula $\phi$ has many variables and few clauses, then $\phi$ is *under-constrained* and as a result it exists many assignment sets. The DLL algorithm requires for under-constrained problem instances few backtracking attempts and therefore "easily" deduces the satisfiability. On the other hand, if the ratio of the number of clauses to variables is large, then $\phi$ is *over-constrained* and almost has no satisfying assignment set. Such over-constrained instances are likewise "easily" deducible for the DLL algorithm. However, there also exists a transition from under-constrained to the over-constrained region. In such a *phase transition* region the probability of the instances being satisfiable equals 0.5 and thus one has the largest uncertainty whether the instances are satisfiable or are unsatisfiable.

For the 3-CNF satisfiability problem, the ratio (phase transition threshold) is experimentally approximated by $\approx 4.24$ [15,20]. That means, when $r_3$ is close[6] to 4.24, the DLL algorithm has to backtrack most frequently to determine the

[6] It is still an open problem to prove where the *exact* phase transition threshold is located. Latest theoretical work showed that the threshold $r_k$ lies within the boundary $2.68 < r_k < 4.51$ for $k = 3$ [21].

final result. If the Boolean formula is under-constrained ($r_3 < 4.24$) or over-constrained ($r_3 > 4.24$), then the algorithm prunes whole branches from the decision tree without exploring the leaves and terminates after few recursive calls.

## 5 Experiment with $\widehat{\phi}_{rcb}$ Instances

The computational complexity of finding detectors is experimentally investigated with the DLL algorithm. More specifically, the parameters $l = 75, r = 3$ are chosen and $|\mathcal{S}|$ is varied from 1 to 25, i.e. for each cardinality value from 1 to 25, $\mathcal{S}$ contains distinct bit strings which are randomly drawn from $\mathcal{U}$. As a result, one obtains Boolean formulas $\widehat{\phi}_{rcb}$ in 75-3-CNF with 75 variables and $(75 - 3 + 1) \cdot |\mathcal{S}|$ clauses, $\mathbf{E}[|\widehat{\phi}_{rcb}|]$ distinct clauses, respectively. To obtain a large number of different $\widehat{\phi}_{rcb}$ instances, for each value of $|\mathcal{S}|$, 300 instances are randomly generated. The DLL algorithm is applied on each instance and the results: satisfiable/unsatisfiable and the number of backtracking attempts are noted. The result is depicted in figure 2. The abscissa denotes the ratio of the average number of distinct clauses to variables. The ordinate denotes the number of backtracking attempts (computational costs). The resulting ordinate values are colored gray if the DLL algorithm outputs satisfiable, otherwise it outputs unsatisfiable and the values are colored black.

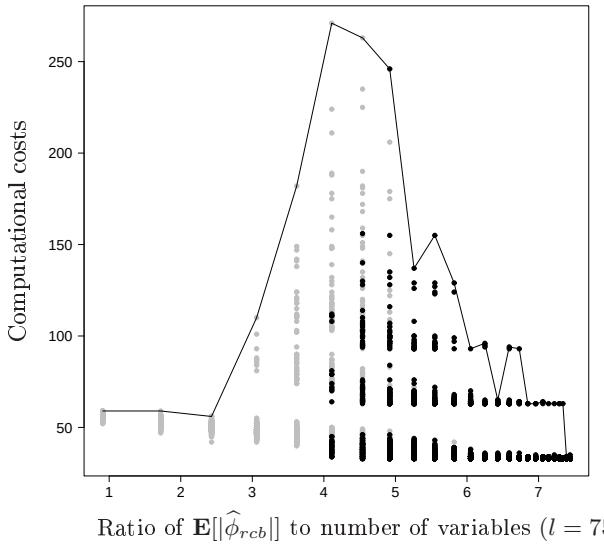



**Fig. 2.** Number of backtracking attempts (computational costs) of the DLL algorithm to decide whether a $\widehat{\phi}_{rcb}$ instance is satisfiable or unsatisfiable. The gray points denote satisfiable instances whereas black points denote unsatisfiable instances. The "hardest" instances are lying in the interval 4 to 5, termed phase transition region.

One can see in figure 2 that for ($r_3 < 4$) a large number of satisfiable instances exist. Or to say it the other way around, for small values of $|\mathcal{S}|$ the resulting Boolean formula $\widehat{\phi}_{rcb}$ is under-constrained and therefore a large number of satisfiable instances exist. The DLL algorithm hence "easily" deduces a satisfiability result. The number of satisfiable and unsatisfiable instances is nearly equal for ($4 < r_3 < 5$). These instances have the largest uncertainty for the DLL algorithm. As a consequence, the DLL algorithm requires the most backtracking attempts to determine whether the instances are satisfiable or are unsatisfiable. A ratio ($r_3 > 5$) implies that a large number of over-constrained instances exist and hence, the DLL algorithm "easily" deduces the unsatisfiable result. Another

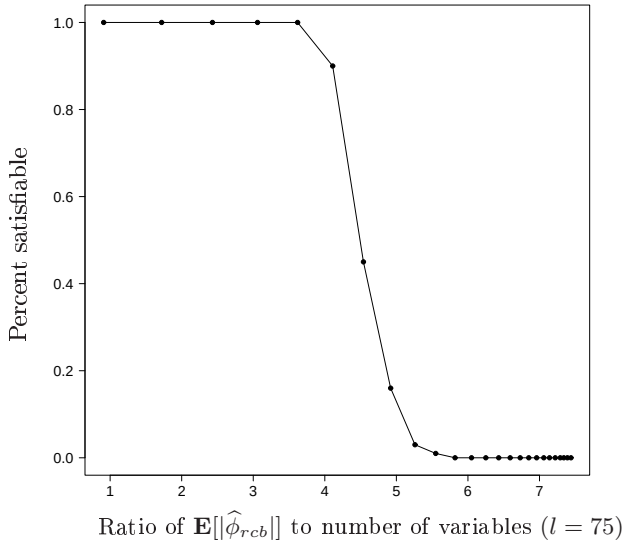



**Fig. 3.** Coherence between the percentage of satisfiable instances and the ratio of $\mathbf{E}[|\widehat{\phi}_{rcb}|]/l$. The "hardest" instances live in the region where the number of satisfiable and unsatisfiable instances is equal, or in other words, the probability of instances being satisfiable equals 0.5.

way to visualize this "easy-hard-easy" pattern, is to plot the percentage of satisfiable instances on the ordinate (see Fig. 3). One can see that the probability of the instances being satisfiable equals 0.5 when ($4 < r_3 < 5$) and rapidly changes to 1 for ($r_3 < 4$) and to 0 for ($r_3 > 5$).

## 6 Conclusion

We have rigorously analyzed the feasibility of generating detectors with respect to the number of self bit strings $|\mathcal{S}|$, the bit string length $l$ and matching length $r$.

With high probability detectors either can be generated or in contrast, can not be generated. However, there also exists a region where the probability rapidly changes from 0 to 1. This behavior can be explained by transforming the problem of finding detectors into the $k$-CNF satisfiability problem. If a large number of self bit strings exist and $r$ is close to 0, then the resulting Boolean formula is over-constrained and no assignment sets exist. In contrast, if a small number of self bit strings exist and $r$ is close to $l$, then the resulting Boolean formula is under-constrained and as a result a large number of assignment sets exist. Moreover we exploited the problem transformation to investigate the computational complexity of finding detectors by means of the DLL algorithm. Finding detectors is "easy" for under-constrained Boolean formulas. It is also "easy" to determine for over-constrained Boolean formulas that no detectors exist. However, for parameter combinations of $|\mathcal{S}|, l$ and $r$ where the resulting ratio of the average number of distinct clauses to variables is close to the phase transition threshold, finding detectors is "hardest". For such "hard" instances the DLL algorithm requires the most backtracking attempts, because the probability of the instances being satisfiable equals 0.5 and thus one has the largest uncertainty whether the instances are satisfiable or are unsatisfiable.

## Acknowledgment

The author thanks Erin Gardner for her valuable suggestions and comments.

## References

1. Percus, J.K., Percus, O.E., Perelson, A.S.: Predicting the size of the T-cell receptor and antibody combining region from consideration of efficient self-nonself discrimination. Proceedings of National Academy of Sciences USA **90** (1993) 1691–1695
2. D'haeseleer, P., Forrest, S., Helman, P.: An immunological approach to change detection: algorithms, analysis, and implications. In: Proceedings of the Symposium on Research in Security and Privacy, IEEE Computer Society Press (1996) 110–119
3. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. In: Proceedings of the Symposium on Research in Security and Privacy, IEEE Computer Society Press (1994) 202–212
4. Ayara, M., Timmis, J., de Lemos, R., de Castro, L.N., Duncan, R.: Negative selection: How to generate detectors. In: Proceedings of the 1nd International Conference on Artificial Immune Systems (ICARIS), University of Kent at Canterbury Printing Unit (2002) 89–98
5. Stibor, T., Timmis, J., Eckert, C.: On the appropriateness of negative selection defined over hamming shape-space as a network intrusion detection system. In: Proceedings of Congress On Evolutionary Computation (CEC), IEEE Press (2005) 995–1002
6. Stibor, T., Timmis, J., Eckert, C.: The link between r-contiguous detectors and k-CNF satisfiability. In: Proceedings of Congress On Evolutionary Computation (CEC), IEEE Press (2006) 491–498

7. Feller, W.: An Introduction to Probability Theory and its Applications. 3. edn. Volume 1. John Wiley & Sons (1968)
8. Ranang, M.T.: An artificial immune system approach to preserving security in computer networks. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet (2002)
9. Uspensky, J.V.: Introduction to Mathematical Probability. McGraw-Hill (1937)
10. Freitas, A.A., Timmis, J.: Revisiting the foundations of artificial immune systems: A problem-oriented perspective. In: Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS). Volume 2787 of Lecture Notes in Computer Science., Springer-Verlag (2003) 229–241
11. González, F., Dasgupta, D., Gómez, J.: The effect of binary matching rules in negative selection. In: Genetic and Evolutionary Computation – GECCO-2003. Volume 2723 of Lecture Notes in Computer Science., Chicago, Springer-Verlag (2003) 195–206
12. Stibor, T., Timmis, J., Eckert, C.: Generalization regions in hamming negative selection. In: Intelligent Information Processing and Web Mining. Advances in Soft Computing, Springer-Verlag (2006) 447–456
13. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. Second edn. MIT Press (2002)
14. Kullmann, O.: The SAT 2005 solver competition on random instances. Journal on Satisfiability, Boolean Modeling and Computation **2** (2006) 61–102
15. Freeman, J.W.: Hard random 3-SAT problems and the Davis-Putnam procedure. Artificial Intelligence **81**(1–2) (1996) 183–198
16. Ouyang, M.: How good are branching rules in DPLL. Discrete Applied Mathematics **89**(1-3) (1998) 281–286
17. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5**(7) (1962) 394–397
18. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM (JACM) **7**(3) (1960) 201–215
19. Gent, I.P., Walsh, T.: The SAT phase transition. In: Proceedings of the 11th European Conference on Artificial Intelligence, John Wiley & Sons (1994) 105–109
20. Selman, B., Mitchell, D.G., Levesque, H.J.: Generating hard satisfiability problems. Artificial Intelligence **81**(1–2) (1996) 17–29
21. Achlioptas, D., Naor, A., Peres, Y.: Rigorous location of phase transitions in hard optimization problems. Nature **435** (2005) 759–764