A natural approach to studying schema processing

Submitted to the IEEE Transactions on Evolutionary Computation on 11/05/2017

Jack McKay Fletcher and Thomas Wennekers

Abstract—The Building Block Hypothesis (BBH) states that adaptive systems combine good partial solutions (so-called building blocks) to find increasingly better solutions. It is thought that Genetic Algorithms (GAs) implement the BBH. However, for GAs building blocks are semi-theoretical objects in that they are thought only to be implicitly exploited via the selection and crossover operations of a GA. In the current work, we discover a mathematical method to identify the complete set of schemata present in a given population of a GA; as such a natural way to study schema processing (and thus the BBH) is revealed. We demonstrate how this approach can be used both theoretically and experimentally. Theoretically, we show that the search space for good schemata is a complete lattice and that each generation samples a complete sub-lattice of this search space. In addition, we show that combining schemata can only explore a subset of the search space. Experimentally, we compare how well different crossover methods combine building blocks. We find that for most crossover methods approximately 25-35% of building blocks in a generation result from the combination of the previous generation's building blocks. We also find that an increase in the combination of building blocks does not lead to an increase in the efficiency of a GA. To complement this article, we introduce an open source Python package called schematax, which allows one to calculate the schemata present in a population using the methods described in this article.

Keywords—Genetic Algorithms, building block hypothesis, schema processing, complete lattice, order theory, schemata, schematic lattice, schematic completion

I. INTRODUCTION

Genetic Algorithms (GAs) are a hugely popular method for optimization and have found successes on many problems [1]. Sadly, unlike other optimization techniques such as gradient decent [17], simulated annealing [3], Ant Colony Optimization [7], Particle Swarm Optimisation [6], [28], GAs lack a rigorous explanation of exactly why and on what functions they perform well. There is, however, a chief approach to studying the power GAs, which is by considering the schemata GAs are processing.

Schemata are simple mathematical objects which describe points and hyper planes in the space of all possible words over an alphabet of the same length [16]. Specifically, a schema is a word made with an additional symbol * called the wild card symbol, which stands for ‘don’t care’. For example, the schema over the binary alphabet 1*1* represents the set of binary strings which have a 1 in positions one and three and a 1 or 0 in positions two and four. In this way, the wild card symbol is similar to a blank tile in the popular board game Scrabble. Schemata have properties. For a schema s, the order of s, denoted o(s) is the number of non-wild card symbols (that is symbols which are not ‘*’) in s. For example, the order of the above schema is 2. The defining length of s, denoted d(s) is the distance between the first and last non wild card symbol. The above schema has a defining length of 2. A word is said to be an instance of s if it matches s. For example, the word 1111 is an instance of the above schema. In the context of a GA, the fitness of s in a population is the average fitness of all of it’s instances.

Holland, in his partly philosophical work on adaptation, argues that any adaptive process test subsets of the search space through schema [16]. As each individual in the search space belongs to several schemata at once, by evaluating one individual many schemata are implicitly sampled. The idea being, if an individual is fit it suggests the schemata of which it is an instance are also fit. Thus, by testing a few individuals many schemata are sampled. This property is called implicit or intrinsic parallelism. Holland claims that natural evolution exploits this property. From this concept, Holland created a statement about the schema processing performed by GAs:

Definition I.1. A building block is a low order, low defining length and above average fitness schema.

Hypothesis I.1. The BBH (The building block hypothesis): competent GAs find increasingly better solutions by combining building blocks.

Holland’s idea of building blocks is threefold. Firstly, as building blocks have above average fitness, they have a high probability of surviving and generating offspring. Secondly, as they have a low order, they have a low probability of being disrupted by mutations. Thirdly, as they have a low defining length they have a low probability of disruption because of crossover. All of these properties point towards the BBH, that is: building blocks surviving and being combined in subsequent generations. Note that the BBH is a statement about adaptive systems in general, but in this case it is applied specifically to GAs.

The BBH was the putative explanation for the power of GAs for a long time. Schema theorems, which provide a lower bound on the expected number of instances of a schema in one generation occurring in the subsequent generation [5], [12], [21] seemed to add credence to the BBH as they show that building blocks have a high chance of surviving. However, in later times the BBH came under many philosophical and theoretical criticisms. In his paper...
method to observe processors to begin with for that matter) is that there is no as a narrative for GAs (and with seeing GAs as schema operators [15].

We believe the fundamental problem with using the BBH as a GA. In particular, encodings are often chosen as to allow BBH is as a heuristic in the design of efficient encodings for GAs. In practical terms, the most common application of the same findings would apply to 'real' building blocks found by discovered by the population itself. It is not obvious if the artificial as they are defined in the fitness function rather than convergence, specifically the GA gets stuck in local optima created by the stepping stones. While the authors offer an indirect insight into BBH, it is still unclear what exactly happens to schemata in the course of a genetic algorithm. The 'building blocks' in Mitchell et al.'s paper are somewhat artificial as they are defined in the fitness function rather than discovered by the population itself. It is not obvious if the same findings would apply to 'real' building blocks found by a GA. In practical terms, the most common application of the BBH is as a heuristic in the design of efficient encodings for GAs. In particular, encodings are often chosen as to allow building blocks to be combined meaningfully by the genetic operators [15].

We believe the fundamental problem with using the BBH as a narrative for GAs (and with seeing GAs as schema processors to begin with for that matter) is that there is no method to observe schemata being manipulated by a GA, as such it is hard, if not impossible to test accurately any meaningful statement about the type of schema processing performed by a GA. Thus, in the current work, we present a natural method for identifying the set of schemata being tested by a population. We call this method the 'schematic completion'. We also find that the schemata found by the schematic completion always forms a mathematical structure called a complete lattice, we call 'the schematic lattice'. Using these methods, one can observe the exact schema processing performed by a GA by simply calculating the schemata present in each population and thus. These methods, we hope, are useful tools for studying GAs through the conceptual lens of schema processing and we hope that they will deepen the understanding of GAs. Specifically, we hope to be able to explore what makes a function difficult or easy for a GA to optimize, understand how useful it is to see GAs as 'schema processors' and perhaps inform the design of better selection, crossover and mutation methods as to improve schema processing of a GA.

In the text that follows firstly the basics of order and lattice theory are introduced, this theory will be used to define our method for calculating the schemata present in a populating. Secondly, schemata are formally defined and the notions of schematic completion and the schematic lattice are introduced. To demonstrate the usefulness of these methods, in section 4 and 5, we show how these novel notion can be used to study GAs both theoretically and experimentally. In section 4, theoretically we show the search space for good schemata is a complete lattice and that each generation samples a complete sublattice of this search space. We also find that combining schemata is not a good method to explore the search space of schemata as in most cases only a subset of the search space can be reached by combining schemata alone. In section 5, we experimentally examine how well various crossover methods combine building blocks. We find that only 25-35% of building blocks in a generation result from the combination of the previous generation's building blocks. We also find that an increase in building block combinations does not correspond to increase in the efficiency of the GA. In the appendix of this article, an open source Python package called schematix is introduced, which efficiently calculates the schematic completion and to draws the schematic lattice. We encourage readers interested in the following work to exploit this package for their research into GAs.

It should be noted that the mathematical insights presented in this article are not particularly difficult to understand in and of themselves. However to situate schemata in the broader mathematics of order and lattice theory some basic mathematical definitions and proofs are required. If the reader is not acquainted with this subject area we advise them to look simply at the examples and figures in section 3 to intuit the notions of the schematic completion and schematic lattice.

II. BACKGROUND MATHEMATICS

In this section, we cover the background mathematics required to introduce the basic insights into schemata presented in the next section. Many of the following definitions regarding order theory and lattice theory are adapted from Ganter and Wille's book on Formal Concept Analysis [10] and also Birkhoff's seminal book on lattice and order theory [4]. If these areas are understood, please skip ahead to the next section. Firstly we cover Order and Lattice theory, secondly, we introduce Closure Systems and Galois Connections.

A. Order and Lattice theory

Definition II.1. A relation \( \leq \) is called a partial order on a set \( S \) if for all \( a, b, c \in S \) it satisfies:

1) reflexivity: \( a \leq a \)

-
Definition II.2. A lower neighbour of an element $b$ is another element $a$ such that there is no element $c$ with: $a < c < b$. In this case, $b$ is an upper neighbour of $a$ and we write $a \prec b$ to indicate this. In the literature, it is also said as $b$ covers $a$.

Every finite partially ordered set, $(S, \leq)$ can be represented by a Hasse diagram. Each element in $S$ is depicted by a circle. For any $a, b \in S$ if $a \prec b$ a line is drawn between the circles representing $a$ and $b$. Using a Hasse diagram one can read off any order relation: $a < b$ iff there is a descending path from $b$ to $a$. Figure 1 shows two Hasse Diagrams.

![Fig. 1: Two Hasse diagrams with 8 elements](image)

Definition II.3. A rank function, $\rho$, over a poset $(M, \leq)$ is a function which maps each element in $M$ to the natural numbers such that for $x, y \in M$ the following two properties are satisfied:

\[
x \leq y \implies \rho(x) \leq \rho(y)
\]

\[
x \leq y \implies \rho(y) = \rho(x) + 1
\]

Definition II.4. Let $(M, \leq)$ be a poset and let $A \subseteq M$. A lower bound of $A$ is an element $m \in M$ with $m \leq a$ for all $a \in A$. An upper bound of $A$ can be defined dually. If there is a largest element in the set of all lower bounds of $A$, this element is called the infimum of $A$, denoted $\bigwedge A$. Dually, if there is a smallest element in the set of all upper bounds of $A$, this element is called the supremum of $A$, denoted $\bigvee A$. If $A = \{x, y\}$, the infimum of $A$ is called the join and is denoted $x \vee y$, dually the supremum of $A$ is called the meet and is denoted $x \wedge y$.

Intuitively, one can think of the supremum of a $A$ as the “smallest element in $M$ which is greater than or equal to all elements in $A$”. The infimum of $A$ can be seen as “the largest element in $M$ which is less than or equal to all elements in $A$”.

Definition II.5. We call an ordered set $L := (L, \leq)$ a lattice if for every $x, y \in L$, $x \wedge y$ and $x \vee y$ exist. We call $(L, \leq)$ a complete lattice if for every $X \subseteq L$, the infimum $\bigwedge X$ and the supremum $\bigvee X$ always exist. Every complete lattice, $L$ has a largest element $\bigvee L$ called the unit element of $L$, denoted $1_L$. Dually it has a smallest element, $\bigwedge L$, called the zero element of $L$, denoted $0_L$.

Every complete lattice is of course a lattice. Moreover, every finite non-empty lattice is a complete lattice.

Example II.1. The left Hasse Diagram in figure 1 is not a lattice nor a complete lattice as the join on the top most elements does not exist. However, the right Hasse diagram in figure 1 is a complete lattice (and thus a lattice) as the supremum and infimum exist for any subset of elements. Any closed real interval $[x, y]$ with $\leq$ under its normal interpretation as an ordering is a complete lattice. However, any unbounded set of real numbers is not a complete lattice, but is a lattice. The power set of any non-empty set with $\subseteq$ as an ordering is an exemplary complete lattice. The Hasse diagram of the complete lattice formed by $(\mathcal{P}(\{a, b, c\}), \subseteq)$ is shown in figure 2.

![Fig. 2: The complete lattice formed by $(\mathcal{P}(\{a, b, c\}), \subseteq)$](image)

Definition II.6. A subset $A$ of a complete lattice $L$ which is closed under suprema and infima, specifically:

\[
X \subseteq A \implies \bigvee X, \bigwedge X \in A
\]

is called complete sublattice. If $A$ is only closed under suprema it is called a meet-subsemilattice. Dually, if $A$ is closed under infima only, it is called a join-subsemilattice.

Example II.2. The poset $(\{\{a, b, c\}, \{a, b\}, \{b, c\}, \{b\}\}, \subseteq)$ is a complete sublattice of the complete lattice defined in figure 2. This complete sublattice can be seen in figure 3 below:

Definition II.7. Let $(L, \leq)$ be a complete lattice. An element $a \in L$ is called an atom of $L$ if $0_L < a$ and there does not exist an element $x \in L$ such that $0_L < x < a$. $L$ is called atomic if every element $b > 0_L$ implies that $b$ is an atom or that $b$ has an atom below it. That is, $b \geq a > 0$. $L$ is called atomistic if every element in $L$ can be given by the supremum of a subset of the atoms.

Example II.3. The set of atoms of the complete lattice shown in figure 2 is: $\{\{a\}, \{b\}, \{c\}\}$. This complete lattice is atomic and atomistic. The complete lattice shown on the right of figure 1 is atomic, however it is not atomistic as the top element cannot be reached by the supremum on any subset of the atoms.
B. Closure Systems and (monotone) Galois Connections

Definition II.8. A closure system on a set $S$ is a set of subsets of $S$ which contains $S$ and is closed under intersection. Specifically, $A \subseteq P(S)$ is called a closure system on $S$ if $S \in A$ and:

$$X \subseteq A \implies \bigcap X \in A$$

Example II.4. Consider, $S = \{a, b, c\}$ and let $A = \\{\{a, b, c\}, \{a, b\}, \{a\}\}$. $A$ is a closure system on $S$, as $S$ is included in $A$ and $A$ is closed under intersection as any intersection of elements in $A$ is also a member of $A$.

Definition II.9. A closure operator on a set $S$ is a map, $cl : P(S) \to P(S)$ which satisfies the following for all $X, Y \subseteq S$:

1) extensity: $X \subseteq cl(X)$.
2) monotonicity: $X \subseteq Y \implies cl(X) \subseteq cl(Y)$.
3) idempotency: $cl(cl(X)) = cl(X)$.

Closure operators and closure systems are closely linked, as can be seen in the following theorem.

Theorem II.5. If $cl$ is a closure operator on a set $S$ then the set:

$$A_{cl} := \{cl(X) | X \subseteq S\}$$

(the set of all closures of a closure operator) is a closure system. Conversely if $A$ is a closure system on $S$ then the following operator:

$$cl_A(X) := \bigcap \{a \in A | X \subseteq a\}$$

Defines a closure operator.

There is a bijection between closure operators and closure systems. Every closure operator has a corresponding closure system and every closure system has a corresponding closure operator. A closure system can be seen as the set of all closures of a closure operator. What's more, closure systems (and thus closure operators) are closely linked with complete lattices, as will be seen in the next proposition.

Proposition II.6. If $A$ is a closure system then $(A, \subseteq)$ is a complete lattice, where for $X \subseteq A$ the infimum, $\bigwedge X$ is given by $\bigcap X$ and the supremum $\bigvee X$ is given by $cl_A(\bigcup X)$. Every complete lattice is isomorphic to the lattice of all closures of a closure system.

Example II.7. The complete lattice formed by the closure system $A$ appearing in the example above is seen in figure 4, below:

$$\{a, b, c\}$$

$$\{a, b\}$$

$$\{a\}$$

$$\emptyset$$

Fig. 4: The complete lattice formed by the closure system $A$ with $\subseteq$ as an ordering.

Definition II.10. Suppose we have two partially ordered sets, $(A, \leq)$ and $(B, \leq)$. Two monotone functions over these sets, $F : A \to B$ and $G : B \to A$ are called a Galois connection of $A$ and $B$ if we have for all $a \in A$ and $b \in B$:

$$F(a) \leq b \iff a \leq G(b)$$

In this case, $F$ is called the lower adjoint and $G$ the upper adjoint. Equivalently, if $F$ and $G$ satisfy the following conditions they also form a Galois connection. For all $a_1, a_2 \in A$ and all for call $b_1 \in B$ we have:

1) $a_1 \leq a_2 \implies F(a_1) \leq F(a_2)$
2) $a_1 \leq GF(a_1)$
3) $a_1 \leq G(b_1) \implies F(a_1) \leq b_1$

Proposition II.8. The composition $GF : A \to A$ is a closure operator.

III. Schemata

In this section we define schemata and define two basic operations, the expansion and compression. Secondly, using these operators we define the notions of the schematic completion and the schematic lattice and prove properties about these notions.

Let $\Sigma$ be a finite alphabet which does not contain symbol $\ast$. We use $\Sigma^l$ to denote the set of all words of length $l$ over $\Sigma$.

Definition III.1. The schematic alphabet of $\Sigma$ is $\Sigma$ with an extra symbol, $\ast$, the wild card symbol. We use $\Sigma_*$ to denote the schematic alphabet of $\Sigma$. Symbols in $\Sigma_*$ are not the wild card symbol are called fixed symbols.

Definition III.2. A schema is a word over $\Sigma_*$. We use $\Sigma_*^l$ to denote all schemata of length $l$ over $\Sigma_*$ including the empty schema, $\epsilon_*$.

Example III.1. Let $\Sigma$ be the binary alphabet, that is $\{1, 0\}$. The schematic alphabet of $\Sigma$, denoted $\Sigma_*$, is the alphabet $\{1, 0, \ast\}$. An example of a schema in $\Sigma_*^4$ is $1 \ast 1 \ast$. 

\[
\begin{align*}
\{a, b, c\} \\
\{a, b\} \\
\{a\} \\
\emptyset
\end{align*}
\]
Definition III.3. For any schema \( s \in \Sigma_\ell \) we define the following operator \( \downarrow s \), called the expansion of \( s \), which maps \( s \) to a subset of words in \( \Sigma^\ell \):

\[
\downarrow s := \{ b \in \Sigma^\ell | b_i = s_i \text{ or } s_i = \ast \text{ for each } i \in \{1, \ldots, l\}\}
\]

where subscript \( i \) denotes the character at position \( i \) in a word or schema. When \( s = \epsilon_\ast \), then \( \downarrow s = \emptyset \). More simply put, \( \downarrow s \) is the set of all words in \( \Sigma^\ell \) that can be made by exchanging the \( \ast \) symbols in \( s \) with symbols from \( \Sigma \).

Example III.2. Continuing the example above, \( \uparrow 1 \ast \ast = \{100, 110, 101, 111\} \). The 1 in the first position is fixed. Note \( \uparrow 111 = \{111\} \) and \( \uparrow \epsilon_\ast = \emptyset \).

Definition III.4. Conversely, for any \( A \subseteq \Sigma_\ell \) we define \( \downarrow A \), called the compression of \( A \), which maps \( A \) on to a schema \( s \in \Sigma^\ell \):

\[
\downarrow A := s
\]

where \( s \) is a schema of length \( l \) such that the symbol at position \( i \) in \( s \) is determined in the following way: if \( x_i = y_i \) for all \( x, y \in A \) then \( s_i = x_i \); otherwise \( s_i = \ast \). If \( A = \emptyset \) then \( \downarrow A = \epsilon_\ast \). One can think of this operator as stacking up all the items in \( A \) and if all elements in a column are equivalent, the symbol at that position in \( s \) takes this value, otherwise there is a wild card symbol.

Example III.3. Let \( A = \{100, 000, 010\} \) then \( \downarrow A = \ast \ast \ast \). Note if \( A = \emptyset \) then \( \downarrow A = \epsilon_\ast \). If \( A = \{100\} \) then \( \downarrow A = 100 \).

Definition III.5. Schemata can be ordered. For any \( a, b \in \Sigma^\ell_\ast \) we say \( a \leq b \) if and only if \( \uparrow a \subseteq \uparrow b \). It follows that \( \leq \) is a partial ordering on a set of schemata from the reflexivity, antisymmetry and transitivity of the subset relation.

Example III.4. Again let \( \Sigma = \{1, 0\} \). Consider the following schema in \( \Sigma^2_\ell \): \( \epsilon_\ast, 11, 1* \ast, \ast \ast \). They are ordered in the following way: \( \epsilon_\ast \leq 11 \leq 1* \leq \ast \ast \). This is because \( \uparrow \epsilon_\ast \subseteq \uparrow 11 \subseteq \uparrow 1* \subseteq \uparrow \ast \ast = \emptyset \subseteq \{1\} \subseteq \{11, 10\} \subseteq \{11, 10, 01, 00\} \).

Definition III.6. It is possible to define compression in terms of expansion:

\[
\downarrow A := s
\]

such that \( A \subseteq \uparrow s \) and for any \( r \in \Sigma^\ell_\ast \)

\[
A \subseteq \uparrow r \implies s \leq r
\]

That is, \( \downarrow A \) is the schema whose expansion includes \( A \) and is the smallest such schema to do so.

Definition III.7. Conversely we can define expansion in terms of compression:

\[
\uparrow s := A
\]

such that \( \downarrow A = s \) and for any \( B \subseteq \Sigma^\ell \) we have:

\[
\downarrow B = s \implies B \subseteq A
\]

That is, \( \uparrow s \) is the largest subset of words whose compression is equal to \( s \).

We will soon see that definitions 3 and 4 of the expansion and compression operators are useful computationally while definitions 6 and 7 are useful in proving properties about schemata.

Proposition III.5. For any schema \( s \in \Sigma^\ell_\ast \), we have \( \downarrow \uparrow s = s \).

Proof: Let \( A = \uparrow s \) definition III.6 trivially yields \( \downarrow A = s \), thus \( \downarrow \uparrow s = s \).

Proposition III.6. For \( A \subseteq \Sigma^\ell \), we have \( A \subseteq \uparrow \downarrow A \).

Proof: Let \( s = \downarrow A \), definition III.7 trivially yields \( A \subseteq \uparrow s \), thus \( A \subseteq \uparrow \downarrow A \).

Proposition III.7. Compression is monotonic, that is for \( A, B \subseteq \Sigma^\ell \):

\[
A \subseteq B \implies \downarrow A \subseteq \downarrow B
\]

Proof: Assume \( A \subseteq B \) we will show \( \downarrow A \leq \downarrow B \). Proposition III.6 gives \( B \subseteq \uparrow \downarrow B \). Since \( A \subseteq B \subseteq \uparrow \downarrow B \) the transitivity of the subset relation yields \( A \subseteq \uparrow \downarrow B \). Let \( \downarrow A = a \) and \( \downarrow B = b \). Since we have \( A \subseteq \uparrow b \), definition III.6 applied to \( \downarrow A \) yields \( a \leq b \). Thus we have \( \downarrow A \subseteq \downarrow B \).

Proposition III.8. For \( A \subseteq \Sigma^\ell \) and \( b \in \Sigma^\ell_\ast \) we have:

\[
A \subseteq \uparrow b \implies \downarrow A \subseteq b
\]

Proof: We will assume \( A \subseteq \uparrow b \) and show \( \downarrow A \leq b \). Proposition III.7 gives us, \( \downarrow A \leq \uparrow \downarrow b \). Proposition III.5 yields \( \downarrow A \leq b \).

Lemma III.9. The compression and expansion operators form a Galois connection, where \( \downarrow \) is the lower adjoint and \( \uparrow \) the upper adjoint.

Proof: Let \( A, B \subseteq \Sigma^\ell \) and \( s \in \Sigma^\ell_\ast \). From definition II.9, it is sufficient to show:

1) \( A \subseteq B \implies \downarrow A \leq \downarrow B \)
2) \( A \subseteq \uparrow \downarrow A \)
3) \( A \subseteq \uparrow s \implies \downarrow A \leq s \)

1) was shown in proposition III.7, 2) was shown in III.6 and 3) in proposition III.8.

Definition III.8. For a set \( A \subseteq \Sigma^\ell \), we call the process of calculating the compression on each subset of \( A \), that is \( \{\downarrow X | X \subseteq A\} \), the schematic completion of \( A \), denoted \( S(A) \).

Example III.10. Let \( \Sigma^2 = \{1, 0\}^3 \) and \( A = \{110, 100, 001, 000\} \) the schematic completion of \( A \), results in the following set:

\[
\{001, 100, 000, 110, 00*, 000, 1*0, *00, *0*, *0*, *, **, \epsilon_\ast\}
\]

For example, the schema \( *00 \) comes from the compression on the subset \( \{110, 000\} \).

Theorem III.11. (The fundamental theorem of schemata) For any \( A \subseteq \Sigma^\ell \), the schematic completion of \( A \), \( S(A) \) ordered by \( \leq \) forms a complete lattice, that is the poset \((S(A), \leq)\) is a complete lattice. We call this lattice the schematic lattice of \( A \). Let \( X \subseteq S(A) \), the supremum, \( \bigvee X \), is given by \( \downarrow \bigcup \uparrow s \).

The infimum, \( \bigwedge Y \), is given by \( \downarrow \bigcap \{\downarrow s | a \in A\} \).

Proof: Lemma III.9 tells us \( \downarrow \) and \( \uparrow \) form a Galois connection, where \( \downarrow \) is the lower adjoint and \( \uparrow \) is the upper adjoint.
adjoint. As such, proposition II.8 yields \( \uparrow \downarrow \) as a closure operator. Hence, from Theorem II.5, \( \{ \uparrow \downarrow X \mid X \subseteq A \} \) is a closure system. Thus, the poset \( \{ \uparrow \downarrow X \mid X \subseteq A \} \) forms a complete lattice (proposition II.6). Proposition II.6 also yields for \( Y \subseteq \{ \uparrow \downarrow X \mid X \subseteq A \} \), the supremum is given by \( \uparrow \bigcup Y \) and the infimum, \( \bigwedge Y \). From the definition of ordering over schemata, we then have \( \{ \downarrow X \mid X \subseteq A \} \) as a complete lattice and the infimum, \( \bigwedge Y \), is given by \( \downarrow \{ a \in \bigwedge X \mid a \in A \} \).

It is easy to check that the atoms of the schematic lattice \( (\mathcal{S}(A), \leq) \) is the set \( A \) and \( (\mathcal{S}(A), \leq) \) is atomistic.

**Example III.12.** Continuing the above example, the schematic lattice formed from schematic completion on \( A \) can be seen in figure 5.

![Fig. 5: The schematic lattice formed by the schematic completion on the set \( A = \{110, 100, 001, 000\} \) ordered by \( \leq \), that is the complete lattice \( (\mathcal{S}(A), \leq) \)](image)

**Example III.13.** Of course, the schematic completion is not restricted to words over the binary alphabet. Let \( \Sigma = \{a, b, c, \ldots, z\} \) and consider the set \( X \subseteq \Sigma^4 \):

\[
X := \{ \text{help, kelp, yell, tell, talk, walk} \}
\]

The schematic completion of this set gives us the schematic lattice in figure 6.

We can more precisely define some of the original properties of schema using the above definitions.

**Definition III.9.** The order (not to be confused with partial order) of schema \( s \in \Sigma^*_x \) is the number of fixed symbols in \( s \) and is denoted \( \alpha(s) \). The order of \( s \) can be equivalently defined as:

\[
\alpha(s) := l - \log_{|\Sigma|}(|s|)
\]

Similarly the antorder, denoted \( \alpha' \) of \( s \) is the number of wildcard symbols in \( s \), which can be defined as:

\[
\alpha'(s) := \log_{|\Sigma|}(|s|)
\]

**Example III.14.** Let \( s = 11 \ast \ast \). We can count the number of fixed symbols to give us \( \alpha(s) = 2 \). Equivalently \( l - \log_{|\Sigma|}(|s|) = 4 - \log_2(|\{1100, 1110, 1101, 1111\}|) = 4 - \log_2(4) = 2 \).

**Proposition III.15.** If we set, \( \alpha'(\varepsilon_s) = -1 \) then \( \alpha' \) is a rank function over schemata.

**Proof:** To show \( \alpha' \) is a rank function over schemata, it is sufficient to show for \( x, y \in \Sigma^*_x \):

1. \( x \leq y \implies \alpha'(x) \leq \alpha'(y) \)
2. \( x \leq y \implies \alpha'(y) = \alpha'(x) + 1 \)

First we will show 1). Assume \( x \leq y \), using the definition of ordering we have \( \uparrow x \subseteq \uparrow y \). It follows then that \( |\uparrow x| \leq |\uparrow y| \), and thus \( \alpha'(x) \leq \alpha'(y) \). Now we show 2). Assume \( x \leq y \) then have \( \uparrow x \subseteq \uparrow y \) with \( |\uparrow y| = |\uparrow x| + |\Sigma| \), thus \( \alpha'(y) = \alpha'(x) + 1 \).

**Corollary III.15.1.** Using the same method, it is possible to show that the order of a schema is a dual rank function over schemata if we make \( \alpha(\varepsilon_s) = l + 1 \). That is for \( x, y \in \Sigma^*_x \):

1. \( x \leq y \implies \alpha(x) \geq \alpha(y) \)
2. \( x \leq y \implies \alpha(x) = \alpha(y) + 1 \)

**Definition III.10.** A word \( a \in \Sigma^4_x \) is said to be an instance of schema \( s \) if and only if \( a \in \uparrow s \).

**Example III.16.** The word 111 is an instance of the schema 1 \( \ast \ast \) as 111 \( \in \uparrow 1 \ast \ast = 111 \in \{111, 110, 101, 100\} \).

We now introduce some novel properties of schemata not originally described in previous works.

**Definition III.11.** For some \( A \subseteq \Sigma^4 \), the confidence of a schema \( s \in \mathcal{S}(A) \) is given as:

\[
\frac{|\{ a \in \uparrow s \mid a \in A \}|}{|\uparrow s|}
\]

In more simple terms, the confidence of \( s \in \mathcal{S}(A) \) is the proportion of \( \uparrow s \) that is found in \( A \).

The confidence of a schema is useful in GAs for understanding how confident one can be in the fitness assigned to a schema. In particular, the more instances of a schema has, the more we can trust it’s fitness.
The following lemmas and proposition are useful for computations over schemata as they allow us to determine the ordering of schemata by considering only the characters and wild cards rather than having to compute the expansion explicitly.

**Lemma III.17.** Let \( A, B \subseteq \Sigma^i \), \( a = \downarrow A \) and \( b = \downarrow B \). \( A \subseteq B \) if and only if for all \( i \) we have:

\[
\forall i \quad a_i = b_i \quad \text{or} \quad b_i = \star
\]

**Proof:** This follows directly from the definition of compression. Let \( C = B \setminus A \). If \( C \) is empty, then clearly \( a_i = b_i \) for all \( i \), otherwise, for any \( x \in C \) and any \( i \), \( x_i \) can either differ from \( a_i \), thus \( b_i = \star \) or \( x_i \) can equal \( a_i \), thus \( b_i = a_i \).

**Proposition III.18.** For \( a, b \in \Sigma^i \), \( a \preceq b \) if and only if for all \( i \)

\[
\forall i \quad a_i = b_i \quad \text{or} \quad b_i = \star
\]

**Proof:** Let \( a, b \) be any schema in \( \Sigma^i \).

(\( \Rightarrow \)) We will assume \( a \preceq b \) and show \( a_i = b_i \) or \( b_i = \star \) for all \( i \). From the definition of order over schema we have \( \uparrow a \subseteq \uparrow b \), let \( a' = \downarrow \uparrow a \), \( b' = \downarrow \uparrow b \), since \( \uparrow a \subseteq \uparrow b \) lemma III.18 yields

\[
a_i' = b_i' \quad \text{or} \quad b_i' = \star
\]

for all \( i \), Proposition III.5 gives us \( \downarrow \uparrow a = a \) and \( \downarrow \uparrow b = b \), meaning \( a = a' \) and \( b = b' \), thus:

\[
a_i = b_i \quad \text{or} \quad b_i = \star
\]

for all \( i \).

(\( \Leftarrow \)) We will assume \( (a_i = b_i \) or \( b_i = \star \) for all \( i \)) and show \( a \preceq b \). From Lemma III.18, there exist \( A, B \subseteq \Sigma^i \) with \( \downarrow A = a \) and \( \downarrow B = b \) such that \( A \subseteq B \).

Proposition III.6 yields: \( B \subseteq \uparrow b \). Since we have \( A \subseteq B \) and \( B \subseteq \uparrow b \), the transitivity of the subset relation yields \( A \subseteq \uparrow b \).

This concludes out order theoretical interpretation of schemata. However, for brevity’s sake, many interesting many properties regarding schemata have not been mentioned here. For example a link to a area of mathematics called Formal Concept Analysis, which is concerned with finding concept hierarchies in object-feature relationships \[10\]. Secondly, as schemata are a (small) subset of regular expressions, this makes the schematic completion akin to the induction of regular languages \[13, 22\] where the search space is known to be a complete lattice \[8\].

In the subsequent section it is shown how these simple insights into schemata, specifically the schematic completion and the schematic lattice, can be used to study how combining schemata explores the search space.

**IV. HOW DOES COMBINING SCHEMATA EXPLORE THE SEARCH SPACE?**

It is common to visualize the search space of GAs as a hypercube in which each schema defines a hyperplane

\[ [31] \] In this section, we offer a complementary view based on the mathematics in the previous section. We show that the search space of schemata is a complete lattice and that each generation of a GA samples a complete sublattice of this complete lattice. Finally, we consider how combining schemata explores the search space.

For a GA working on binary strings (that is, \( \Sigma = \{0, 1\} \)) of size \( n \), the search space of all schemata is the set \( \Sigma^n \). If we order this set the search space is revealed to be the complete lattice \( (\Sigma^n, \leq) \). We could also construct the search space by using the schematic completion on the set of all possible words of length \( n \), meaning:

\[
(\Sigma^n, \leq) = (S(\Sigma^n), \leq)
\]

Figure 8 shows the search space of schemata for GAs working on binary strings of length 3.

Given a generation \( G_t \) at time \( t \) of a GA, the schematic completion on \( G_t \), \( S(G_t) \), yields at least a subset of the schemata being tested by \( G_t \). However, it is still unclear if the schematic completion on \( G_t \) yields all the schemata being tested by \( G_t \). To explore this possible limitation, consider the population

\[
G_t = \{1010, 1111, 1100, 1000\},
\]

The schematic completion on \( G_t \) returns the set:

\[
\{1111, 1101, 1100, 1000, 1*1*, 11***, 10*0, 1*00, 1**0, 1***, \epsilon_+\}
\]

Which forms the schematic lattice shown below in figure 7.

![Fig. 7: The complete lattice formed from the schematic completion on the population \( G_t \). That is, the lattice \((S(G_t), \leq)\).](image-url)
without having a 1 in the beginning (as both 1100 and 1000 begin with a 1), hence the schema which is being sampled by this population is 1 * 00 which does appear in the schematic lattice. Indeed, as the fitness of a schema is given by the average of its instances the fitness of * * 00 and 1 * 00 have the same fitness, however 1 * 00 is the accurate description of the schema being tested. A similar argument can be made for any schema which appears to be omitted, thus, the schematic completion returns all schemata being sampled by \( G_t \). Each generation of a GA then defines a set of schemata, \( S(G_t) \), which is the set of schemata being sampled by \( G_t \). When \( S(G_t) \) is ordered by \( \subseteq \), it is a complete sublattice of the search space \( (\Sigma^n_*, \leq) \).

By calculating the schematic completion on the population for each generation, one can observe how schemata change during the course of a GA. There are many natural experiments and questions which can be examined using this method (for example, one could test how well various schema theorems apply to a GA with a finite population size), however in this section we will focus on the more fundamental question of how combining schemata explores the search space.

It is proposed that a GA combines ‘good’ schemata, however it is unknown how directly combining schemata explores the space of all schemata. To investigate, a method is required to identify when a schema results from the combination of a set of schemata. To do this we introduce the notion of schematic blending.

**Definition IV.1.** For \( X \subseteq \Sigma^n_* \) the **schematic blending** of \( X \), denoted \( bl(X) \) is the schema given by

\[
bl(X) := \downarrow(\bigcap_{s \in X} \uparrow s)
\]

If \( bl(X) \) returns the empty schema then \( X \) is said to be unblendable, otherwise \( X \) is said to be blendable. Given a set of schemata \( A \), the set of all schematic blends in \( A \), that is \( \{ bl(x) | x \subseteq A \} \) is denoted \( B(A) \).

**Example IV.1.** \( bl((1 * * *, **11*) = 1111 *, 11 ** * and 01 ** * are unblendable, \( bl(1 * 10 * *, **11011 *) = 1101 * \).

It is clear that the order of the schematic blend of \( A \), (if it is not empty) is greater than or equal to the order of any member of \( A \). In addition, when performing the schematic blend on schemata which result from the schematic completion on a set of words, it is common for the blend to result in schemata which already exist in schematic completion. Indeed, the following lemma explores this idea and is useful for understanding how combining schemata explores the schema space.

**Lemma IV.2.** (The schematic blending lemma) For a set of schemata \( A \subseteq \Sigma^n_* \), \( bl(A) \) returns the largest schema \( s \in \Sigma^n_* \) such that \( s \leq s' \) for all \( s' \) in \( A \).

**Proof:** Let \( A \subseteq \Sigma^n_* \). Then we have:

\[
bl(A) = \downarrow(\bigcap_{s \in A} \uparrow s)
\]

Let:

\[
bl(A) = \downarrow(\uparrow s_1 \cap \uparrow s_2 \cap \cdots \cap \uparrow s_n)
\]

Thus from the definition of compression we have \( t \) as the largest schema with:

\[
\uparrow t \subseteq \uparrow s_1 \text{ and } \uparrow t \subseteq \uparrow s_2 \text{ and } \uparrow t \subseteq \uparrow s_3 \text{ and } \cdots \text{ and } \uparrow t \subseteq \uparrow s_n
\]

and from the definition of partial ordering for schemata we have:

\[
t \leq s_1 \text{ and } t \leq s_2 \text{ and } t \leq s_3 \text{ and } \cdots \text{ and } t \leq s_n
\]

In simple terms this lemma says: blending a set of schemata \( A \) returns the largest schema \( s \in \Sigma^n_* \) which is smaller than all elements in \( A \). In this way the schematic blending is similar to the infimum operator over the schematic lattice. However, instead of returning a schema \( s \in A \), a schema \( s \in \Sigma^n_* \) is required.

So far we understand this much: given a generation at time \( t \), \( G_t \) of a genetic algorithm, the schemata being tested by this generation is the schematic completion of \( G_t, S(G_t) \), this samples a complete sub-lattice of the lattice all the possible schemata. The set of schemata which can be reached by blending these schemata is then \( B(S(G_t)) \). However, the schematic blending lemma tells us that blending only searches the spaces inbetween the layers of the lattice, and not ‘sideways’ or upwards. What is more, schematic blending is idempotent that is: \( B(B(S(G_t))) = B(S(G_t)) \). Thus, by combining schemata alone, only a search over the lower neighbours of pre-existing schemata can be performed, meaning only a subset of the space of all schemata can be reached. Figure 8 demonstrates how blending explores the space of all schemata and is a good visual summary of the results from this section. Thus we conclude, blending schemata alone is not a good tool for exploring the space of all schemata.

It is proposed that the power of GAs comes through the combination of a particular type of schemata, namely building blocks. However, as combining schemata in general is not a good exploratory tool, it sheds a serious doubt on how useful combining building blocks is as a search tool. Specifically, if a building block is not ‘enclosed’ within the schematic lattice defined by the initial generation of a GA, combining schemata alone will not discover it. This suggests the disruption and construction of schemata through the imperfect combination of schemata (via crossover) and mutation may play a vital role in allowing a greater exploration of the search space. This contrasts with the traditional view of GAs, where the disruption of schemata via crossover is traditionally seen as a nuisance as they hinder the combination of good schemata \([30],[31]\). There has been some work however which suggests that the construction of novel schemata through crossover play a useful exploratory role \([23]\).

In the next section, we demonstrate how the concepts of the schematic completion and the schematic lattice can
Fig. 8: The search space for all schema of length three arranged as a complete lattice. The schemata surrounded by bold circles are those sampled by the generation $G_t = \{010, 011, 001, 101, 100\}$, that is the elements in the complete sub-lattice $S(G_t)$. The schemata in dotted circles are those which cannot be reached through any combination of schemata in $S(G_t)$, while the plain circles show the schemata which can be reached through the combination schema in $S(G_t)$. This figure was made using the schematax package.

be used experimentally to observe the schema processing performed by a GA.

V. OBSERVING BUILDING BLOCKS

In this section, we use the schematic completion to observe the building blocks during the course of a GA. First, however, to study the building block hypothesis using the above methods, we must more precisely define building blocks. In Holland’s framing (definition I.1), the phrases ‘low order’ and ‘low defining length’ could refer to an absolute value, such that ‘low order’ schema are schema whose order is less than say 4. However, for the purpose of this article we take ‘low’ to be relative to the given generation, so that ‘low order’ schema are schema whose order is less than average. However, for the purpose of this article we take ‘low’ to be relative to the given generation, so that ‘low order’ schema are schema whose order is less than average. Building blocks are then redefined as follows:

Definition V.1. A building block is a schema with below average order, below average defining length and above average fitness $^1$.

Using this definition one can find the building blocks present in a given generation by firstly using the schematic completion on the generation to find all schemata being tested, then by secondly filtering out the building blocks using the definition above. In our framing, the BBH is a statement about the map from a schematic lattice in generation $t$ to the schematic lattice in generation $t + 1$. In particular, the BBH states the building blocks in generation $t + 1$ should to some degree result from the combination of building blocks in generation $t$.

However, before we examine how well building blocks are combined using the above updated version of the building block hypothesis and the notion of schematic blending, it is first interesting to examine how the average order and defining length of building blocks change over the course of a GA. If the order and defining length of building blocks increase, it suggests that building blocks are getting larger and more clumped together (as suggested by the BBH). To investigate, we consider the Canonical GA$^2$ solving the all ones problem (where the fitness of an individual is the number of ones found in the string). We use binary strings of length 64 and pick the mutation rate to be 0.005 and the population size as 30. We run the GA for 120 generations, and for each generation, we calculate the schematic completion on each to yield all the schemata being tested by that generation, filter out the building blocks using our modified definition of the BBH above and then calculate the average order and defining length of the set of building blocks. We plot the results in Figure 9 averaged over 20 simulations, where each simulation is started with a random initial population. In addition, to give the reader an indication of what the building blocks may look like for the all ones problem, we display the 3 building blocks from generations, 0, 40, 80 and 120 from one simulation in figure 9. As can be seen in Figure 9, the order and defining

---

$^1$It should be noted that when computing the average order, fitness and defining length of schemata we do not include the empty schema nor schema with no wildcards (i.e the individuals of the population).

$^2$The Canonical GA is a binary GA, with roulette wheel selection, single point crossover and mutation.
length of the building blocks is indeed increasing during the course of a GA. It seems, at least for the all ones problem, that the defining length and order of the building blocks quickly increase, and then level out around generation 60.

The results in figure 9, hint at the BBH being implemented GA. However, it is still unclear if and how many building blocks are being explicitly combined by the genetic operators of a GA. Perhaps the results in figure 9 are not best explained by building blocks being combined but instead it is to be expected when a population becomes less random. In particular, when a set of words is more random the average order and defining length of the schemata will be lower because the words ‘agree’ less. As such, the less random the population becomes (as is the case in a GA working on a reasonable fitness function), the more the words ‘agree’ and thus the schemata as result will have higher order and defining length. It is interesting to note that the building blocks which are found are far less neat than those suggested by [19] in the royal road fitness functions.

For our second experiment, we use the concept of schematic blending to examine exactly how many building blocks from a generation \( t + 1 \) result from the combination of building blocks in generation \( t \). We test how well building blocks are combined using different crossover methods. The GA is setup the same as above, however to keep the calculation of the schematic completion and blends tractable, we use a population of size 12 working over individuals of length 16. We calculate the building blocks of generation \( t \) as before, we then find the set of all schematic blends on the building blocks, let’s call this set \( B \). It is then checked what percentage of the building blocks in generation \( t + 1 \) are members of \( B \). Figure 10 displays the results. The results, in this case, are averaged over 100 simulations. On average only approximately\( 25\% - 35\% \) of building blocks from a generation are created by the combination of building blocks from the previous generation in the case of 1 to 9 point crossover as well as uniform crossover (UX). Interestingly, it is proposed that UX disturbs the combination of building blocks compared to traditional crossover methods [24], [26], however we find it combines building blocks equally well as other crossover methods. In general the reason why building blocks are not combined optimally are several and mostly well known: firstly individuals which are instances of building blocks are not guaranteed to be selected by roulette wheel selection for crossover, thus those building blocks cannot be blended, secondly mutation can disrupt the blending of building blocks if a fixed symbol is mutated after crossover, thirdly crossover is not guaranteed to blend schemata if a suboptimal crossover point is chosen. Thus, we can conclude combining building blocks from generation \( t \) only accounts for \( 25 - 35\% \) of the building blocks in generation \( t + 1 \), the remaining building blocks from generation \( t + 1 \) are created by other means.

Probabilistic crossover (PX) which is similar to UX but chooses bits using a weighted probability proportional to the fitness of the parents, combines building blocks the most effectively (due to it picking fitter bits with a higher probability), yet finds the optimal solution in a later generation. It is proposed that “competent genetic algorithms combine building blocks” [16]. PX offers a counter-example to this statement as it combines building blocks well, but is not ‘competent’ in that it takes a greater number of generations to find the optimal solution compared to other crossover methods which combine building blocks poorly. It is possible that greater combination of building blocks limits the exploration of the GA. To explain further: the schematic blending lemma tells that combining building blocks will only explore the lower neighbors of pre-existing building blocks, thus the GA (with PX) explores this subset well, but does not search other areas of the schematic lattice effectively. Much like Mitchell et al. [19] we believe that the over emphasis of building blocks forces the GA into a locally optimal sublattice of the search space. We conclude that the ability of a genetic algorithm to combine building blocks does correspond to how quickly it will find the optimal solution and that the novel creation of schemata (through methods other than combining building blocks) is vital for a competent GA.

VI. DISCUSSION AND CONCLUSION

The schematic completion and the schematic lattice, are fruitful in the field of GAs both theoretically, as demonstrated in the mathematics presented in sections 4, and experimentally as demonstrated in section 5. It seems using both methods however, inconsistencies are found the original schema processing theory for GAs. Specifically, section 4 shows that combining schemata (and thus building blocks) explicitly is not a good method to explore the search space of schemata as only the lower neighbors of pre-existing schemata can be reached. While section 5 shows that ‘competent’ GAs do not seem to be very concerned with combining building blocks to begin with, as only approximately \( 25-35\% \) of building blocks seem to come from the combination of the previous generations building blocks for most crossover methods. In addition, an increase in the combination of building blocks (as seen in PX) does not correspond with an increase in efficiency of a GA, rather, it hinders the GA. The reason for this follows from the schematic blending lemma, specifically combining building blocks only explores a subset of the search space, thus the more a GA combines building blocks the more it gets stuck in this subset. Finally, the increase of order and defining length of building blocks over time (as is seen in figure 9), is better explained by a decrease in randomness in the population rather than by building blocks being combined. However, we believe the most significant contribution of this article are the methods introduced to explicitly calculate the schemata present in a population and the identification underlying lattice structures involved in schema processing. We hope that these methods will deepen the understanding of GAs. If the reader is interested in these methods, we encourage them to exploit the schematax software which is introduced in the following appendix section.
To complement this article, we introduce an open source Python package which implements schemata and all of their properties defined above. Importantly, the package allows one to compute the schematic completion and to draw the schematic lattice, it can downloaded from https://github.com/STB/python-schemata

![Diagram](image)

(a) The average order and defining length of building blocks during the course of a GA (averaged over 20 simulations). In this plot, the dashed line represents the average defining length of the building blocks while the solid line represents the average order of the building blocks.

Fig. 9

| Crossover method | Solution found | % Building Blocks combined |
|------------------|----------------|---------------------------|
| 1-point          | 107.3 ± 25.0   | 0.25 ± 0.16               |
| 2-point          | 104.77 ± 28.27 | 0.37 ± 0.16               |
| 3-point          | 101.16 ± 29.13 | 0.37 ± 0.17               |
| 4-point          | 105.06 ± 29.55 | 0.37 ± 0.16               |
| 5-point          | 94.19 ± 38.36  | 0.37 ± 0.16               |
| 6-point          | 104.41 ± 39.23 | 0.36 ± 0.17               |
| 7-point          | 103.11 ± 32.28 | 0.36 ± 0.16               |
| 8-point          | 94.37 ± 36.06  | 0.36 ± 0.16               |
| 9-point          | 99.51 ± 33.19  | 0.36 ± 0.17               |
| UX               | 99.95 ± 31.20  | 0.36 ± 0.16               |
| PX               | 119.88 ± 1.19  | 0.53 ± 0.18               |

Fig. 10: Time taken for various crossover methods to find the optimal solution and the average percentage of building blocks combined by the respective crossover method. UX here stand for uniform crossover, while PX stands for probabilistic crossover. Each GA is solving the all ones problem on strings of size 16, with a population of size 12 and uses roulette wheel selection. The results are averaged over 100 simulations, each starting with a random initial population.

**VII. SCHEMATAX - A PYTHON SOFTWARE PACKAGE FOR SCHEMA**

To complement this article, we introduce an open source Python package which implements schemata and all of their properties defined above. Importantly, the package allows one to compute the schematic completion and to draw the schematic lattice, it can downloaded from https://github.com/STB/python-schemata

Naively calculating the schematic completion using the definitions given above requires iterations over the powerset and thus is very computationally expensive ($O(2^n)$). Thus we introduce an algorithm, algorithm 1 (below), to compute the schematic completion. This algorithm is based on the algorithm presented in [20]. In this algorithm, the join operation is used on each pair of schemata. This algorithm exploits the commutativity ($x \lor y = y \lor x$) and idempotency ($x \lor x = x$) of the join operation. Meaning we do not have to compute $x \lor x$, and if $x \lor y$ is computed, we do not need to compute $y \lor x$. This allows the inner loop only to loop over a subset of schemata. Additionally, we exploit the atomistic nature of the schematic lattice to build the lattice from the bottom up.

**Proposition VII.1.** Algorithm 1’s worst case time complexity is $O(nkN^2)$ where $N$ is the total number of schemata found, $k$ is the length of the strings and $n$ the initial number of strings, that is the size of $P$.

**Proof:** The outer loop (line 4) loops over $P$, thus clearly looping $n$ times. For each $n$, the inner loop (line 5) loops $p_c \cdot N - c$ times, where $p_c \leq 1$ is the proportion of $N$ currently found. join$(x,y)$ takes $k$ steps (line 6), where $k$ is the length of the strings. To check if $s$ is not in the current set of schemata found takes $p_c \cdot N$ steps (line 7). So we have:

$$O\left(\sum_{c=1}^{n} (p_c \cdot N - c) \cdot k \cdot (p_c \cdot N)\right)$$

$$= O\left(\sum_{c=1}^{n} N \cdot k \cdot N\right)$$

$$= O\left(N^2 \cdot k \sum_{c=1}^{n} 1\right)$$
\[ = \mathcal{O}(nkN^2) \]

To draw the schematic lattice we exploit the Graphviz software \cite{Gansner2000} which allows one to draw aesthetically pleasing lattices efficiently. In addition, we use the package cython to transfer the Python code into C \cite{Ganter1999}. This dramatically increases the efficiency of the schematax package. For more information the reader is referred to the documentation of the software package at: https://github.com/iSTB/python-schemata.

**Algorithm 1** Schematic completion

```python
1: procedure Complete(P)
2: schemata \rightarrow P
3: c \leftarrow 1
4: for x in P do
5: check \leftarrow in schemata[c:]
6: for y in check do
7: s \leftarrow join(x, y)
8: if s \notin schemata then
9: schemata = schemata \cup \{s\}
10: c \leftarrow c + 1
11: return schemata \cup \{e_x\}
```

**Algorithm 2** Join

```python
1: procedure join(x, y)
2: k \leftarrow length(x)
3: s \leftarrow empty string
4: for for i in 0, \ldots, k do
5: if x[i] = y[i] then
6: s[i] \leftarrow x[i]
7: else
8: s[i] \leftarrow y
9: return s
```

**ACKNOWLEDGMENT**

This work was supported by the Marie Curie Initial Training Network FP7-PEOPLE-2013-ITN (CogNovo, grant number 604764), the Engineering and Physical Sciences Research Council (BABEL, grant number EP/J004561/1), and the University of Plymouth (through a PhD studentship to Jack McKay Fletcher). We also wish to thank Diego Maranan, Sue Denham and John Matthias for their valuable comments.

**REFERENCES**

[1] Chang Wook Ahn and R. S. Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Transactions on Evolutionary Computation*, 6(6):566–579, Dec 2002.

[2] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31–39, 2011.

[3] Claude JP Bélisle. Convergence theorems for a class of simulated annealing algorithms on rd. *Journal of Applied Probability*, pages 885–895, 1992.

[4] Garrett Birkhoff, Garrett Birkhoff, Garrett Birkhoff, and Garrett Birkhoff. *Lattice theory*, volume 25. American Mathematical Society New York, 1948.

[5] Clayton L Bridges and David E Goldberg. An analysis of reproduction and crossover in a binary-coded genetic algorithm. *Grefenstette*, 878:9–13, 1987.

[6] Maurice Clerc and James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

[7] Marco Dorigo, Mauro Birattari, and Thomas Stützle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006.

[8] Pierre Dupont, Laurent Miclet, and Enrique Vidal. What is the search space of the regular inference? In *International Colloquium on Grammatical Inference*, pages 25–37. Springer, 1994.

[9] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*, 30(11):1203–1233, 2000.

[10] Bernhard Gauner, Rudolf Wille, and Rudolf Wille. *Formal concept analysis*, volume 284. Springer Berlin, 1999.

[11] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[12] David E Goldberg and Kumara Sastry. A practical schema theorem for genetic algorithm design and tuning. In *Proceedings of the genetic and evolutionary computation conference*, pages 328–335, 2001.

[13] Vincent Grauville, Mirko Křivánek, and Jean-Paul Rasson. Simulated annealing: A proof of convergence. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(6):652–656, 1994.

[14] Chi-Ruey Hwang. Simulated annealing: theory and applications. *Acta Applicandae Mathematicae*, 12(1):108–111, 1988.

[15] Cezary Z Janikow and Zbigniew Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In *ICGA*, pages 31–36, 1991.

[16] Holland John. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, 1992.

[17] Krzysztof C Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, 90(1):1–25, 2001.

[18] GA Miller and N Chomsky. Pattern conception. In *Paper for Conference on pattern detection, University of Michigan*, 1957.

[19] Melanie Mitchell, Stephanie Forrest, and John H Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In *Proceedings of the first european conference on artificial life*, pages 245–254. Cambridge: The MIT Press, 1992.

[20] Lhouari Nourine and Olivier Raynaud. A fast algorithm for building lattices. *Information processing letters*, 71(5-6):199–204, 1999.

[21] Riccardo Poli. Exact schema theorem and effective fitness for gp with one-point crossover. In *GECCO*, pages 469–476, 2000.

[22] Ray J Solomonoff. A new method for discovering the grammars of phrase structure languages. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 230–237, 1991.

[23] William M Spears and Kenneth A De Jong. On the virtues of parameterized uniform crossover. In *In Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–237, 1991.

[24] William M Spears. Recombination parameters. In *The Handbook of Evolutionary Computation*, pages 1–3. University Press, 1997.

[25] Thomas Stützle and Marco Dorigo. A short convergence proof for a class of ant colony optimization algorithms. *IEEE Transactions on evolutionary computation*, 6(4):358–365, 2002.

[26] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.
[27] Chris Thornton. The building block fallacy. *Complexity International*, 4, 1997.

[28] Ioan Cristian Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information processing letters*, 85(6):317–325, 2003.

[29] Michael D Vose. *The simple genetic algorithm: foundations and theory*, volume 12. MIT press, 1999.

[30] Michael D Vose and Gunar E Liepinsl. Punctuated equilibria in genetic search. *Complex systems*, 5:31–44, 1991.

[31] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.