Annales Univ. Sci. Budapest., Sect. Comp. 36 (2012) 159–177

NEIGHBORHOOD PRINCIPLE DRIVEN ICF ALGORITHM AND GRAPH DISTANCE CALCULATIONS

Norbert Kézdi, Katalin Pásztor Varga and Éena Jakó
(Budapest, Hungary)

Communicated by László Kozma

(Received December 19, 2011; revised January 25, 2012;
accepted February 1, 2012)

Abstract. In the field of Boolean algebra there are many well known methods to optimize/analyze formulas of Boolean functions. This paper presents a non-conventional graph-based approach by describing the main idea of the *Iterative Canonical Form* (or ICF) [5,9] of a Boolean function, and introducing an algorithm that is capable of calculating the ICF. The algorithm is iteratively processing the neighboring nodes inside the n-cube. The iteration step count is a linear function of n. Some novel efficient methods for computing distances by matching non-complete bipartite graphs, derived from the ICF and named as ICF-graphs [4,9,10] are proposed. Different metrics between the ICF-graphs are introduced, such as weighted and normalized node count distances, and graph edit distance. Some recent results and application possibilities of the ICF-graph based distance calculations from the field of molecular systematics, ecology and medical diagnostics are discussed.

1. Introduction

The efficiency of Boolean function manipulation in various applications depends on the form of representation of Boolean functions. In principle, a

Key words and phrases: Iterative Canonical Form, ICF, ICF-graph, Boolean function, Boolean algebra, mathematical logic, graph distance, graph metrics, biological applications.
2010 Mathematics Subject Classification: 03B70, 03G05, 03G10, 05C85.
1998 CR Categories and Descriptors: E.1, E.2, F.4.1, G.2.2, J.3.

Boolean function may be represented by an infinite number of Boolean formulas. In practical applications, however, it is useful to consider some restricted classes of such formulas, in which any Boolean function is represented by exactly one formula, called canonical form. The most widely used canonical forms are the *Complete Disjunctive Normal Form* (CDNF), *Complete Conjunctive Normal Form* (CCNF) and the *Zhegalkin polynomial*, known also as the *Reed-Muller form*. The Iterative Canonical Form (or ICF) of a Boolean function introduced by Jakó [5] differs from the known disjunctive (conjunctive) normal forms in using only monotone disjunctions (conjunctions), and from the Zhegalkin polynomial in using operations of Boolean algebra instead of Boolean ring. Because of this substantial difference, the ICF algorithm can take advantage of the lattice-structure of the n-dimensional Boolean space, unlike the conventional algorithms, which are using the standard rules of simplification.

Normal forms have been proved as fundamental tools in automated theorem proving, logical design and in the investigation of the complexity of logical mappings. The ICF and corresponding ICF-graph is a successful approach to that [10].

In the field of structural pattern recognition and classification, graphs constitute a powerful way of representing discrete objects. A Boolean function of n variables can be represented as a rooted, directed, acyclic graph on a corresponding n-dimensional Boolean cube (Boolean n-cube). In the Boolean n-cube a vector with the rank k (where k is the number of the true components in the vector) will generate an $(n-k)$-dimensional sub-cube. All of the nodes inside the generated sub-cube will share the truth value with the generator vector, according to the analyzed Boolean function. If all of the vectors inside the generated sub-cube are having the same truth value as the Boolean function under consideration, and there are no unprocessed vectors in the domain, then the ICF algorithm will terminate. Otherwise, if there is a vector inside the generated sub-cube with a conflicting truth value, then the algorithm will go on with the minimal nodes of these conflicting vectors. The final result will be a graph, named as ICF-graph, which is a colored sub-graph of the n-cube. The ICF-graph provides a compressed description of the Boolean function without loss of information. The nodes in the ICF-graph can be considered as generator vectors of the sub-cubes. There are two different types of nodes in the graph, called *"generator"* and *"closure"* nodes. The *"closure"* nodes are colored white, while the *"generator"* nodes are colored black. Each node is labeled by a Boolean vector (which can be considered as a monotone conjunction/disjunction of Boolean variables or their negates). The black nodes generate those vectors which will have the 1 (or true) truth values according to the given Boolean function, while the white ones will generate the 0 (or false) ones. There is an edge between two nodes if the Hamming distance of the two Boolean vectors is exactly one.

Some metrics between ICF-graphs are introduced, such as: *Node count distance*, *Weighted node count distance*, *Normalized node count distance* and *Graph edit distance*. Here the graph edit distance metric can be used for arbitrary graphs as well, although since it is not symmetric it is not a mathematical metric, but still useful in other application areas.

2. The ICF algorithm

In this section we shall lay down the basic definitions and will prove some lemmas/theorems so we can introduce the ICF calculator algorithm.

Let us denote the *n-dimensional Boolean space* by $\mathbb{B}^n$, where $\mathbb{B} = \{true, false\}$. An element from $\mathbb{B}^n$ will be an *n-dimensional Boolean vector*. In other words, an element from $\mathbb{B}^n$ is an ordered n-length list containing logical values. The logical value *true* will be denoted by 1 and *false* by 0. If $n = 4$, then, for example

$$x = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} true \\ true \\ false \\ true \end{bmatrix}$$

is an element of $\mathbb{B}^4$. The elements within a Boolean vector are called *coordinates* or *bits*. In the previous example x had the following bits: $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$.

Definition 2.1. If $x, y \in \mathbb{B}^n$, then $dc(x, y) \overset{def}{\Leftrightarrow} \exists i \in [1, n] \cap \mathbb{N} : \forall j \in \in ([1, n] \cap \mathbb{N}) \backslash \{i\} : x_j = y_j$, and $x_i = 0$, while $y_i = 1$. In other words the Hamming-distance of x and y is 1, and $\hat{x} < \hat{y}$, where $\hat{x}$ denotes the number format of x. For example, if $x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, then $x_1 = 1$, $x_2 = 0$, $x_3 = 0$ and $\hat{x} = 100_2 = 4_{10}$. One should read this as "y is a *direct child* of x".

Definition 2.2. If $x, y \in \mathbb{B}^n$, then $x \to y \overset{def}{\Leftrightarrow} dc(x, y)$ or $\exists z_1, z_2, \ldots, z_k \in \in \mathbb{B}^n : dc(x, z_1), dc(z_2, z_3), \ldots, dc(z_k, y)$. One should read this as "there is a route from x to y".

Definition 2.3. If $x, y \in \mathbb{B}^n$, then $x \widetilde{\to} y \overset{def}{\Leftrightarrow} x \to y$ or $x = y$.

Definition 2.4. If $\emptyset \neq A \subseteq \mathbb{B}^n$, then $\alpha(A) = \{x \in \mathbb{B}^n : A \ni a \widetilde{\to} x\}$. This operation shall be called as *α-extension*.

Definition 2.5. If $x \in \mathbb{B}^n$, then $DC(x) := \{y \in \mathbb{B}^n : dc(x, y)\}$.

Definition 2.6. If $x \in \mathbb{B}^n$, then $Rank(x) = |\{x_i = 1 : i = 1, \ldots, n\}|$. In other words, $Rank(x)$ is the number of true bits inside the x Boolean vector.

Definition 2.7. If $\emptyset \neq A \subseteq \mathbb{B}^n$, then $\alpha_2(A)$ (*strict α-extension*) can be calculated with the following method:

$$\begin{aligned} G_0 &:= A \\ G_1 &:= \bigcup_{x \in G_0} DC(x) \\ G_2 &:= \bigcup_{x \in G_1} DC(x) \\ &\vdots \\ G_k &:= \bigcup_{x \in G_{k-1}} DC(x) \\ &\vdots \end{aligned}$$

We continue until we reach a positive q, where $G_q = \emptyset$. This will surely happen at some point, since with the calculation of each G_k the rank of the elements will increase. If the dimension of Boolean space is n, then the maximum rank is also n, and there will be only $\binom{n}{n} = 1$ element with the rank n, the vector with the $2^n - 1$ number format. This vector has not got any child, which means $DC(2^n - 1) = \emptyset$. Since the $\mathbb{B}^n$ space is finite, we will reach $2^n - 1$ eventually.

If $\emptyset \neq A \subseteq \mathbb{B}^n$, then

$$\alpha_2(A) := \bigcup_{i=1}^{q-1} G_i.$$

Lemma 2.1. *If $\emptyset \neq A \subseteq \mathbb{B}^n$, then $\alpha_2(A) = \{x \in \mathbb{B}^n : A \ni a \to x\}$.*

Proof. If $x_i \in \alpha_2(A)(x_i \neq 0)$, then $\exists j \in \mathbb{N}^+ : x_i \in G_j \Rightarrow \exists x_{i-1} \in G_{j-1}$ that $dc(x_{i-1}, x_i)$, since

$$G_j = \bigcup_{g \in G_{j-1}} DC(g) = \{x \in \mathbb{B}^n : g \in G_{j-1} \wedge dc(g, x)\}.$$

But in this case $\exists x_{i-2} \in G_{j-2}$ that $dc(x_{i-2}, x_{i-1})$ and so on, which means $\exists x_1 \in G_1$ that $dc(x_1, x_2)$. We know from before that

$$G_1 = \bigcup_{a \in G_0} DC(a)$$

Since $G_0 = A$, this means that $\exists a \in A$ that $dc(a, x_1)$. So $dc(a, x_1)$, $dc(x_1, x_2)$, $dc(x_2, x_3)$, …, $dc(x_{i-1}, x_i)$, and the Definition 2.2. indicates that $a \to x_i$. ■

Corollary 2.1. *If $\emptyset \neq A \subseteq \mathbb{B}^n$, then $\alpha(A) = \alpha_2(A) \cup A$.*

Proof. As we saw, according to Lemma 2.1.

$$\alpha_2(A) = \{x \in \mathbb{B}^n : A \ni a \to x\},$$

which means:

$$\begin{aligned}\alpha_2(A) \cup A &= \{x \in \mathbb{B}^n : A \ni a \to x\} \cup A = \\ &= \{x \in \mathbb{B}^n : A \ni a \to x, \text{ or } x \in A\} = \\ &= \{x \in \mathbb{B}^n : A \ni a \to x, \text{ or } x = a\} = \\ &= \{x \in \mathbb{B}^n : A \ni a \widetilde{\to} x\} = \\ &= \alpha(A).\end{aligned}$$

■

Theorem 2.1. *The α_2 extension can be calculated with the following algorithm as well:*

$$\begin{aligned}C_0 &:= A \\ C_1 &:= \bigcup_{x \in C_0} DC(x) \\ C_2 &:= \bigcup_{x \in C_1 \setminus C_0} DC(x) \\ &\vdots \\ C_k &:= \bigcup_{x \in C_{k-1} \setminus (\bigcup_{j=0}^{k-2} C_j)} DC(x) \\ &\vdots\end{aligned}$$

We continue until we reach a positive q, where $C_q = \emptyset$. We claim that:

$$\alpha_2(a) = \bigcup_{i=1}^{q-1} G_i = \bigcup_{i=1}^{q-1} C_i\,.$$

Proof.

$$\begin{aligned}G_0 &= A = C_0 \\ G_1 &= \bigcup_{x \in G_0} DC(x) = C_1 \\ G_2 &= C_2 \cup \bigcup_{x \in C_1 \cap C_0} DC(x)\end{aligned}$$

$$G_2 = C_2 \cup \bigcup_{x \in C_1 \cap C_0} DC(x)$$

$$\vdots$$

$$G_k = C_k \cup \bigcup_{x \in C_{k-1} \cap (\bigcup_{j=0}^{k-2} C_j)} DC(x)$$

$$\vdots$$

Note that for every $k \in \mathbb{N}^+$ number,

$$\bigcup_{x \in C_{k-1} \cap (\bigcup_{j=0}^{k-2} C_j)} DC(x) \subset \bigcup_{i=1}^{k-1} C_i,$$

which means that:

$$\bigcup_{i=1}^{q-1} C_i = \bigcup_{i=1}^{q-1} G_i = \alpha_2(A). \qquad \blacksquare$$

Corollary 2.2. *If $A, B \subseteq \mathbb{B}^n$, and $A \neq \emptyset$ and $B \neq \emptyset$ then $\alpha(A \cup B) = \alpha(A) \cup \alpha(B)$.*

Proof. This is a trivial consequence of Definition 2.4. $\blacksquare$

Corollary 2.3. *If $A := a_1, a_2, \ldots, a_n \subseteq \mathbb{B}^n$, then*

$$\alpha(A) = \bigcup_{i=1}^{n} \alpha(a_i)$$

Proof. This is a trivial consequence of Corollary 2.2. $\blacksquare$

Corollary 2.4. *If $\emptyset \neq A \subseteq \mathbb{B}^n$, and $a \in A$, then $\alpha(\{a\}) \subseteq \alpha(A)$.*

Proof. According to Corollary 2.2.,

$$\alpha(A) = \alpha(A \backslash \{a\} \cup \{a\}) = \alpha(A \backslash \{a\}) \cup \alpha(\{a\}),$$

which means $\alpha(A) = \alpha(A \backslash \{a\}) \cup \alpha(\{a\})$. Therefore, $\alpha(\{a\}) \subseteq \alpha(A)$. $\blacksquare$

Lemma 2.2. *If $x, y \in \mathbb{B}^n$, and $x \to y$, then $\alpha(\{y\}) \subset \alpha(\{x\})$.*

Proof. According to definition 2.4., $\alpha(\{x\}) = \{b \in \mathbb{B}^n : x \widetilde{\to} b\}$, also $\alpha(\{y\}) = \{b \in \mathbb{B}^n : y \widetilde{\to} b\}$, therefore $y \in \alpha(\{x\})$.

Definition 2.1. and 2.2. tells us that the $\to$ relation is a strict partial order, so it is transitive, ergo $\forall b \in \mathbb{B}^n : y \widetilde{\to} b \Rightarrow x \to b$, since $x \to y$.

According to Lemma 2.1., $\{\forall b \in \mathbb{B}^n : x \to y \widetilde{\to} b\} \subseteq \alpha_2(\{x\})$, which is a subset of $\alpha(\{x\})$, ergo $\alpha(\{y\}) \subset \alpha(\{x\})$. $\blacksquare$

Corollary 2.5. *If* $\emptyset \neq A \subseteq \mathbb{B}^n$, *and* $x \notin A$, *also* $\exists a \in A : a \to x$, *then* $\alpha(\{x\}) \subset \alpha(A)$.

Proof. This is a consequence of Corollary 2.4. and Lemma 2.2. since $\alpha(\{a\}) \subseteq A$, as well as $\alpha(\{x\}) \subset \alpha(\{a\})$, therefore $\alpha(\{x\}) \subseteq A$. ■

Definition 2.8. If $X, Y, Z \subseteq \mathbb{B}^n$, then $\beta(X) = Y \Leftrightarrow Y \subseteq X$ and $\alpha(Y) = \alpha(X)$, but $\neg\exists Z \subset Y : \alpha(Z) = \alpha(X)$. This operation is called *β-reduction.*

Lemma 2.3. *If* $A := \{a_1, a_2, \ldots, a_m\} \subseteq \mathbb{B}^n$, *and* $\exists (a_i, a_j) \in A^2 : a_i \to a_j$, *then* $\alpha(A) = \alpha(A \backslash \{a_j\})$.

Proof. According to Corollary 2.3.

$$\alpha(A) = \bigcup_{k=1}^{m} \alpha(\{a_k\})$$

$$\alpha(A\backslash\{a_j\}) = \bigcup_{k=1, k\neq j}^{m} \alpha(\{a_k\}).$$

At first we could think that $\alpha(A)$ contains every element of $\alpha(\{a_j\})$, while $\alpha(A\backslash\{a_j\})$ will not necessarily contain all of them. But $a_i \in \alpha(A\backslash\{a_j\})$, since $a_i \to a_j$, ergo $a_i \neq a_j$. Therefore, as Corollary 2.5. would suggest $\alpha(\{a_j\}) \subset$ $\subset \alpha(A\backslash\{a_j\})$. ■

Lemma 2.4. *If* $A := \{a_1, a_2, \ldots, a_m\} \subseteq \mathbb{B}^n$, *and* $\exists j \in \{1, 2, \ldots, m\} : \forall i \in$ $\in \{1, 2, \ldots, m\}\backslash\{j\} : a_i \not\to a_j$ *then* $\alpha(A) \neq \alpha(A\backslash\{a_j\})$.

Proof. Let us denote $A' := A\backslash\{a_j\}$. According to Lemma 2.1., $\alpha_2(A') =$ $= \{y \in \mathbb{B}^n : A' \ni a \to y\}$, which means $a_j \notin \alpha_2(A')$. Moreover, Corollary 2.1. indicates that $\alpha(A') = \underbrace{\alpha_2(A')}_{a_j \notin} \cup \underbrace{A'}_{a_j \notin} \Rightarrow a_j \notin \alpha(A')$, but the initial conditions tell us that $a_j \in A \Rightarrow a_j \in A \cup \alpha_2(A) = \alpha(A)$. Ergo $\alpha(A) \neq \alpha(A') =$ $= \alpha(A\backslash\{a_j\})$. ■

Theorem 2.2. *If* $\emptyset \neq A \subseteq \mathbb{B}^n$, *then* $\beta(A) = A\backslash\alpha_2(A)$.

Proof. According to Lemma 2.1., $\alpha_2(A) = \{x \in \mathbb{B}^n : A \ni a \to x\}$. Let us denote $H := A \cap \alpha_2(A)$, in this case $\forall h \in H\ \exists a \in A : a \to h$ and $H \subset A$. Let us denote $A' := A\backslash H$, Lemma 2.3. indicates that $\alpha(A') = \alpha(A)$.

Now let us prove that A' is a minimal set with the previous quality. It is clear that $A' = A\backslash H = A\backslash\alpha_2(A)$, also according to Lemma 2.1. $\alpha_2(A) =$ $= \{x \in \mathbb{B}^n : A \ni a \to x\}$, which means we will delete all of those vectors from A that are related to each other with the $\to$ relation, in other words: $\neg\exists (x, y) \in A' \times A' : x \to y$. Ergo, if we drop an arbitrary element from A', then we will not be able to restore this element with an α-extension because of Lemma 2.4. ■

Corollary 2.6. *If* $\emptyset \neq A \subseteq \mathbb{B}^n$, *then* $\beta(A) = \{x \in A : \neg\exists y \in A : y \to x\}$.

Proof. This is a trivial consequence of Theorem 2.2. ■

Definition 2.9. If $\emptyset \neq A \subseteq \mathbb{B}^n$, then then we can split A into $n+1$ disjoint *rank-classes*. Each rank-class will contain those elements of A which are having the same rank. Let us denote this with $A_R := \{x \in A : Rank(x) = R\}$.

Theorem 2.3. *If* $A \subseteq \mathbb{B}^n$, *then we can calculate* $\beta(A)$ *with the following algorithm:*

Let us denote $m := \min\{Rank(x) : x \in A\}$, *and* $M := \max\{Rank(x) : x \in \in A\}$. *Let us see the following sets:*

$$\begin{aligned}
B_0 &:= A_m \\
B_1 &:= B_0 \cup \{y \in A_{m+1} : \neg\exists z \in B_0 : z \to y\} \\
B_2 &:= B_1 \cup \{y \in A_{m+2} : \neg\exists z \in B_1 : z \to y\} \\
&\vdots \\
B_k &:= B_{k-1} \cup \{y \in A_{m+k} : \neg\exists z \in B_{k-1} : z \to y\} \\
&\vdots
\end{aligned}$$

where $1 \leq k \leq M - m$. *In this case:*

$$\beta(A) = B_{M-m}.$$

Proof. It is a trivial consequence of definition 2.6. and 2.9. that $A = \bigcup_{i=0}^{n} A_i = \bigcup_{i=m}^{M} A_i$, because $A_0 = A_1 = \ldots = A_{m-1} = \emptyset$, and $A_{M+1} = \ldots = A_n = \emptyset$ (if $m > 0$ and $M < n$). As we know that $B_0 = A_m$, it is clear that $\forall a \in A_m : Rank(a) = m \Rightarrow \neg\exists(a, b) \in A_m \times A_m : a \to b$. Furthermore, since the minimal rank of an element in set A is m, $\neg\exists x \in A : x \to a$ $(a \in A_m)$. As we saw $B_k = B_{k-1} \cup \{y \in A_{m+k} : \neg\exists z \in B_{k-1} : z \to y\}$, which means $\neg\exists(a, b) \in B_k \times B_k : a \to b$. Since $1 \leq k \leq M-m$, we have covered the whole A set with the B_k sets, and we filtered out those elements that are related to each other with the $\to$ relation. Therefore, $B_{M-m} = \{x \in A : \neg\exists y \in A : y \to x\}$, and this will be equal with $\beta(A)$ according to Corollary 2.6. ■

Definition 2.10. Let us denote the truth set of an f Boolean function with $M_f^{(1)}$. The ICF (Iterative Canonical Form) of f can be calculated iteratively by executing the β-reduction and α-extension operations in the following algorithm:

1. i:=1
2. $V_{1,1} := M_f^{(1)}$

3. $S_{1,i} := \beta(V_{1,i})$
4. $V_{0,i} := \alpha(S_{1,i}) \backslash M_f^{(1)}$
5. If $V_{0,i} = \emptyset$ then goto 11
6. $S_{0,i} := \beta(V_{0,i})$
7. $V_{1,i+1} := \alpha(S_{0,i}) \cap M_f^{(1)}$
8. If $V_{1,i+1} = \emptyset$ then goto 11
9. $i := i + 1$
10. goto 3
11. Collect the vectors from the $S_{i,j}$ sets. The "generator" (black) vectors can be found in the $S_{1,j}$, while the "closure" (white) vectors in the $S_{0,j}$ sets.

Definition 2.11. We can easily get the *ICF-graph* of a Boolean function by the following algorithm:

1. Calculate the ICF of the Boolean function under consideration
2. Draw down the n-cube.
3. Delete those nodes (vectors) that cannot be found in any of the $S_{i,j}$ sets.
4. Color the nodes (vectors) in the $S_{1,j}$ sets to black.
5. Color the nodes (vectors) in the $S_{0,j}$ sets to white.

Theorem 2.4. *The algorithm in Definition* 2.10 *will terminate in a finite time.*

Proof. The number of iteration steps is in linear correlation with the dimension of the Boolean space. The algorithm will terminate if $V_{1,j}$ or $V_{0,j}$ is an empty set for a positive j. With every new iteration, the minimum of the ranks will be increased, also for a positive k number the set $S_{0,k}$ will contain those vectors that are enclosing the generator nodes in $S_{1,k}$. There are two possible outcomes:

- In the last iteration the maximal element of the space (the vector with the rank n) shall be inside the set $S_{1,m}$ ($m \in \mathbb{N}^+$). Since all of the elements in the n-dimensional Boolean space are in relation with the n-ranked element, this will be the one and only vector in $S_{1,m}$. As the maximal element does not have any child, additional closure nodes will not be required, which means $V_{0,m+1}$ will be the empty set.

- The other possible outcome is that we have already processed all of the nodes inside the truth set of the Boolean function, which means that at least in the next step all of these will be enclosed by the closure nodes. However, this means that we cannot add any additional generator nodes, nor we need to add any additional closure nodes.

It is clear that the n-dimensional Boolean space is finite, and the maximal element will have the rank n. So there cannot be any more nodes with a higher rank. ■

Theorem 2.5. *If we have calculated the ICF of a Boolean function, then we can describe this function with the following formula:*

$$M_f^{(1)} = \bigcup_{i=0}^{n} (\alpha(S_{1,i}) \backslash \alpha(S_{0,i})).$$

Proof. We have $(n+1)$ different rank-classes if the dimension of the Boolean space is n, but the null-vector is related with every other element in the Boolean space, so the number of $S_{1,j}$ sets cannot be more than n. Also every closure set must have a corresponding generator set. The "non-existent" sets can be looked as empty sets, so these will not have any impact on the result. For an arbitrary positive k number the set $S_{1,k}$ will contain generator nodes, and exactly these shall be enclosed by the nodes within $S_{0,k}$. Thanks to the α-extension with $\alpha(S_{1,k}) \backslash \alpha(S_{0,k})$, we will get a sub-set of the truth set of the original Boolean function, since every nodes within $S_{1,k}$ is having a true value, and the node inside $S_{0,k}$ are the first nodes — according to the $\rightarrow$ partial order relation — that will have a false value. If we calculate the $\alpha(S_{1,k}) \backslash \alpha(S_{0,k})$ $(1 \leq \leq k \leq n)$ for every generator and closure set pairs, then

$$M_f^{(1)} = \bigcup_{i=0}^{n} (\alpha(S_{1,i}) \backslash \alpha(S_{0,i})),$$

since with the $\alpha(S_{i,j})$ sets we already covered all of the vectors in the range of the Boolean function. ■

3. ICF graph distance

In this section we shall introduce some distance concepts especially for ICF-graphs. These can be used for ICF-graph comparison, so in a way we can

compare anything that is representable with ICF-graphs. The ICF method has already been used successfully in various biological applications.

Let us denote the set of all n-dimensional ICF-graphs with $\mathbb{ICF}_n$. Let us assume that G is an n-dimensional ICF-graph. The set of black nodes within G are denoted by G^1. ($G^1 := \{x | x \text{ is a black node of } G\}$), and the set of the white nodes are denoted by G^0.

Definition 3.1. If $X, Y \in \mathbb{ICF}_n$, then $X \triangle Y = Z \overset{def}{\Leftrightarrow} Z^1 = X^1 \triangle Y^1 \wedge$ $\wedge X^0 \triangle Y^0$, where $\triangle$ is the symmetric difference set operation.

Remark. Note that Z in Definition 3.1. not necessarily will be a valid ICF-graph. Let us just look at it as a set pair.

Definition 3.2. Assume $X, Y \in \mathbb{ICF}_n$, and $X \triangle Y = Z$, in this case the *node count distance* between X and Y is:

$$\delta(X, Y) := |Z| := |Z^1| + |Z^0| =: |X \triangle Y|$$

Definition 3.3. Until now, every node in each graph had the constant weight of 1. Instead of this fix weight, let us use a weight function; the *weighted node count distance* between $X, Y \in \mathbb{ICF}_n$ is:

$$\delta_{f_n}(X, Y) := |X \triangle Y|_{f_n} := \sum_{x \in X \triangle Y} f_n(x),$$

where $f_n : [0, 2^n - 1] \cap \mathbb{N}_0 \to \mathbb{R}^+$ is the weight function.

Definition 3.4. Assume $X, Y \in \mathbb{ICF}_n$, then the *normalized node count distance* of X and Y is:

$$\delta_{norm}(X, Y) = \begin{cases} \frac{|X \triangle Y|_{f_n}}{|X \cup Y|_{f_n}} & \text{if } X \neq Y \neq \emptyset; \\ 0 & \text{if } X = Y = \emptyset. \end{cases}$$

Remark. Each node should get a weight value independently of the other nodes. This is important in order to make sure that the distance between two graphs will be 1 only if the two graphs are disjoint.

The normalization will set boundaries for the distance: $0 \leq \delta_{norm}(X, Y) \leq$ ≤ 1, since $X \triangle Y \subseteq X \cup Y$, also $\delta_{norm}(X, Y) = 1 \Leftrightarrow X \cap Y = \emptyset$, because $\delta_{norm}(X, Y) = 1 \Leftrightarrow |X \cup Y| = |X \triangle Y|$, but $|X \triangle Y| = |X| + |Y| - 2 \cdot |X \cap Y|$, and $|X \cup Y| = |X| + |Y| - |X \cap Y|$, therefore $|X| + |Y| - 2 \cdot |X \cap Y| =$ $= |X| + |Y| - |X \cap Y| \Leftrightarrow X \cap Y = \emptyset$.

Remark 3.1. The three distance concepts above are metrics, because it can be proved that:

Let δ denote any of the previously defined distances. If $X, Y, Z \in \mathbb{ICF}_n$, then

1. $\delta : \mathbb{ICF}_n \times \mathbb{ICF}_n \to \mathbb{R}$,
2. $\delta(X, Y) \geq 0$,
3. $\delta(X, Y) = \delta(Y, X)$,
4. $\delta(X, Y) = 0 \Leftrightarrow X = Y$,
5. $\delta(X, Y) \leq \delta(X, Z) + \delta(Z, Y)$.

3.1. Graph edit distance

In some cases such as chemical or molecular biological applications, we would like to define distance concepts that are not necessarily symmetric. In spite of that the following distance is not symmetric — therefore, not a mathematical metric at all —, it is still a valid approach in some situations.

A simple way to define the distance between two graphs is to determine the minimal cost transformation chain that is needed to make the "start" graph isomorphic to the "final" graph [11].

The allowed distortions/manipulations (or operations) can be freely chosen. When we work with ICF-graphs, the following three operations are allowed:

1. Node insertion: Insert(x) or I(x)
2. Node deletion: Delete(x) or D(x)
3. Node substitution: Substitute(x) or S(x)

We will transform the white and black nodes separately. The distance will be the sum of the two costs of the optimal transformation of the black and white nodes.

Let us introduce cost functions for each edit operation. The main idea is that a low cost edit operation represents a minor change, while a high cost operation will represent a significant change in the graph. These cost values will be stored in a matrix.

In the cost matrix's row header the black/white nodes of the "start" graph $U = \{u_1, \ldots, u_n\}$ are being listed, while in the column header the nodes of the "final" graph $V = \{v_1, \ldots, v_m\}$, respectively. The cost matrix's (i, j) element will represent the cost value of the operation, which will transform the u_i node into the v_j node.

Remark 3.2. We denote an operation sequence by $\text{Action}_1(x)$; $\text{Action}_2(y)$, which means that after we had preformed the Action_1 operation on the argument x (x could be a vector), we preform the Action_2 operation on the argument

y (y could be a vector). $Cost(T)$ or $C(T)$ is the summarized cost of the T operation sequence. If $C(S(x,y)) > C(D(x);I(y))$, then we use the $D(x);\, I(y)$ operation sequence instead of the $S(x,y)$ operation.

If the row count of the cost matrix is n and the column count of the cost matrix is m, then $k := \min\{n,m\}$. In order to get the optimal transformation between the "start" and "final" graph, we have to choose those k different cells from the cost matrix, which make the addition of the chosen cells minimal.

This problem can be solved optimally with the Hungarian algorithm. It has been originally proposed to solve the assignment problem in polynomial time. The name "Hungarian method" came from the fact that the algorithm was largely based on the earlier works of two Hungarian mathematicians: Dénes Kőnig and Jenő Egerváry.

Let G_1 and G_2 each be an n-dimensional ICF-graph. $B_i := G_i^1 \backslash G_1^1 \cap G_2^1$ and $W_i := G_i^0 \backslash G_1^0 \cap G_2^0$, where $i \in \{1,2\}$.

$\delta(G_1,G_2) := C(B_1 \to B_2) + C(W_1 \to W_2)$, where $\to$ denotes the optimal transformation.

3.1.1. The Hungarian method

This section will describe a variant of the Hungarian method. The input of the algorithm will be two sets. One will contain the "workers", and the other the "jobs". We also know which worker can, and with how much cost preform a certain job. In our case the "workers" set will contain the nodes of the "start" graph, and the "jobs" set will contain the nodes of the "final" graph. The cost of an assignment will be the cost of the node transformation.

1. Generate the cost-matrix, where each element represents the cost of a single node assignment: $C_{i,j} := min\{C(S(u_i,v_j)), C(D(u_i);I(v_j))\}$.

2. If $n \neq m$, then insert additional $|n-m|$ rows or columns respectively into the cost matrix, so it will become a square matrix. In these additional rows/columns each element represents the cost of a single node insertion/deletion respectively.

3. For each row r in C, subtract its smallest element from every element in r.

4. For each column c in C, subtract its smallest element from every element in c.

5. For all zeros in $C\,(C_{i,j} = 0)$, count the amount of zeros in row i and in column j, let us denote this value with $Z_{i,j} = |\{C_{i,k} = 0 : k \in \in \{1,2,\ldots,\max\{m,n\}\}\}| + |\{C_{k,j} = 0 : k \in \{1,2,\ldots,\max\{m,n\}\}\}|$.

Mark a zero with a star if there is no starred zero in its row or column, and $Z_{i,j}$ is minimal. (Assignment pairs are indicated by the positions of the starred zeros in the matrix.)

6. If there are $\max\{m, n\}$ starred zeros in the matrix, then GOTO 13.
7. Tick those rows which do not contain any starred zeros. Tick the corresponding column if there is a zero element in a ticked row. Tick the corresponding row if there is a starred zero in the ticked column. Repeat the procedure until there are no other rows or columns that can be ticked.
8. Cover each unticked row and ticked column. (Now all the zeros should be covered. The number of covered rows/columns indicates the number of the possible assignment pairs.)
9. If not all the elements are covered in the matrix, then we know for certain that we should be able to select $max\{m, n\}$ zeros to mark with a star. GOTO 5, but this time try a different zero selection to mark with star, following the rules defined in STEP 5. as close as possible.
10. Save the smallest uncovered element: θ.
11. Add θ to every double-covered element, and subtract it from every uncovered element.
12. Clear all the covers, stars and ticks. GOTO 5.
13. Assignment pairs are indicated by the positions of starred zeros in the cost-matrix.

Remark. The steps 3 and 4 will make sure that there will be at least one zero in every row and column. We can only select one assignment pairs in every row and column. The optimal solution for the reduced matrix will be the optimal solution for the original matrix as well.

The solution will be optimal, since the Hungarian method is proved to give optimal solution for the assignment problem. Also as seen in STEP 1, the cost matrix has been created by this rule: $C_{i,j} := \min\{C(S(u_i, v_j)), C(D(u_i); I(v_j))\}$. This means the cost of the additional node deletions/insertions cannot be decreased further.

3.1.2. The case of arbitrary graphs

We can use a similar approach for arbitrary, undirected graphs. We will use the Hungarian method to assign the nodes of the "start" graph to the nodes of the "final" graph. When we work with ICF-graphs, the storage of the edges

is unnecessary. Now we have to handle the edges as well. At first we only transform the nodes of the graph from the "start" to the "final". Let G_1 be the "start" and G_2 the "final" graph. Let us denote $U = \{u_1, \ldots, u_n\}$ the nodes of G_1 and $V = \{v_1, \ldots, v_m\}$ the nodes of G_2.

The cost matrix will be calculated by the following formula:

$$C_{i,j} := |\deg(u_i) - \deg(v_j)|,$$

where $\deg(x)$ denotes the x node's degree.

The cost of the deletion or insertion of the node x will be: $\max\{C_{i,j} : i \in \in \{1, \ldots, n\} \wedge j \in \{1, \ldots, m\}\} + \deg(x)$.

With this cost-matrix we will assign the "similar" nodes to each other from a graph-topology viewpoint. After we finished with the transformation of the nodes by executing the Hungarian method, we shall transform the edges separately. If we have preformed the $S(x, x')$ and $S(y, y')$ node operations, and the edge e connected the node x to y in the original graph, then e should connect now the x' and y' nodes to each other. If we deleted a non isolated node, then we should connect the corresponding edges to a "fictional" node. It is possible that we will have to delete or insert additional edges. At the end, the edges should be transformed to be isomorphic to the "final" graph. We shall achieve this by preforming the Hungarian method. The cost function can be chosen freely for this.

3.2. The cost functions and their functionality

3.2.1. The case of ICF-graphs

When we use "weighted node count distance", than we give the nodes of the ICF-graph a cost value. In an ICF-graph the storage of the edges are unnecessary, also the nodes have a special functionality. Therefore, we really only need to give cost value to the nodes.

Definition 3.5. *Rank cost function:* Assume $X, Y \in \mathbb{ICF}_n$, and let $Z := := X \triangle Y$ $(n \geq 2)$. We introduce two node cost functions:

$$f_n(x) = \begin{cases} 1 & \text{if } x \in Z^0 \wedge x \notin Z^1 \\ n - Rank(x) + 1 & \text{if } x \in Z^1 \wedge x \notin Z^0 \\ 2 \cdot n + 3 - Rank(x) & otherwise, \end{cases}$$

$$g_n(x) = \begin{cases} Rank(x) & \text{if } x \in Z^0 \wedge x \notin Z^1 \\ n - Rank(x) + 1 & \text{if } x \in Z^1 \wedge x \notin Z^0 \\ 2 \cdot n + 2 & otherwise. \end{cases}$$

Remark. It is clear from the definition that $f_n : [0, 2^n - 1] \cap \mathbb{N}_0 \to \mathbb{N}^+$, and the same goes for g_n, since in a correct ICF-graph the rank of a white node cannot be 0.

Definition 3.6. *Absorption cost function:* If $G \in \mathbb{ICF}_n$, and $x \in G^1$, then the absorption cost of x will be $|\alpha(\{x\})\backslash\alpha(G^0)|+1$. This will count the number of nodes generated by x, and increase this value by one in the case of a zero generation. If y is in G^0, then the absorption cost should be 1. Or it could be the following:

If $y \in S_{0,i} \subseteq G^0$, then $H := \{x \in S_{1,i} : x \to y\}$. In this case, the absorption cost of y will be $|\alpha(H) \cap \alpha(\{y\})|$; this will count how many nodes have been "closed" by y.

Remark. This cost function should not be used with the normalized node count distance, since in this case the cost value of the nodes is not independent of each other.

When we use "graph edit distance" then by cost function, we mean cost-matrix. A cell in the cost-matrix will indicate the cost of the substitution of two nodes. We should transform the black and white nodes separately, and then sum the two transformation costs.

Definition 3.7. *Rank cost function:* If $x \neq y$, then

$$C(S(x, y)) := |Rang(x) - Rang(y)| + 1,$$

where x is a black/white node of the "start", and y is a node of the "final" graph, and y is the same color as x. If $x = y$ and x shares the same coloration as y, then $C(S(x, y)) := 0$. If $x = y$ and x has a different color than y, then $C(S(x, y)) := Rang(x) + Rang(y) + 1$. Let G_1 be the "start" graph, and G_2 be the "final" graph. $G_1 \in \mathbb{ICF}_n$ and $G_2 \in \mathbb{ICF}_m$. $k := \max\{m, n\}$. The cost value of a node insertion/deletion will be $k + 2$; these operations will be the most expensive ones.

3.2.2. The case of "arbitrary" graphs

By cost function we mean cost-matrix in this particular case. A cell in the cost-matrix will indicate the cost of the substitution of two edges similar to what was shown in section 3.1.2.

After we transformed the nodes of the "start" graph by preforming the Hungarian method with the cost-matrix defined in section 3.1.2., we have to transform the edges as well. We will do this by using the Hungarian method again, but now the cost-matrix will contain cost values for the edge edit operations. If G is the "final" graph after the node transformation, and there $\exists e_1, e_2 \in$

$\in Edge(G)$ where e_1 and e_2 have a vertex in common and $G' = G - e_1 + e_2$, then we say that G is transformed into G' by an edge rotation [3]. Let us introduce the following cost-matrix for the edge operations $C(S(e_1, e_2))$, which will indicate the needed amount of edge rotations. We should give the edge insertion/deletion the most expensive cost value: $\max\{C_{i,j} : i \in \in \{1, \ldots, n\} \wedge j \in \{1, \cdots, m\}\} + 1$.

4. Discussion

The algorithms and graphs based on the ICF of Boolean functions together with different metrics can be used in various fields of applications, such as computer aided design of digital systems, classification and pattern recognition, ecological, chemical, biological applications and medical diagnostics.

The ICF algorithm was initially used in logical design of combinational circuits and Programmable Logic Arrays (PLAs) [6] as well as for recognition and classification of discrete objects [2]. Here the ICF-based contraction algorithms were considered for describing functions with a finite partially ordered domain of definition and a linearly ordered range of values. Application possibilities for k-valued functions are shown from the field of microbiology and engineering diagnostics on examples of contraction of the genetic code and as an operational model of a logical device which implements a partial k-valued function.

In ecological modeling, the ICF representations were applied to recognition of spatial patterns in plant coalitions [4]. The traditional methods mostly focus on spatial dependence relations between *pairs* of species, which may result in an over-simplified coalition-system, while many relationships remain unexplored. The simulation experiments and analysis of field data suggested that, in comparison with the florula diversity approach, the ICF was much more sensitive to changes in the combinations. The ICF graphs and analytical expressions were used also as generalized molecular descriptors in comparative sequence analysis and classification of biological macromolecules [8].

Some recent results, based on ICF-graph distance calculations were obtained in molecular systematics for generation of phylogenetic trees [1]. The performance and reliability of the ICF method were tested and compared with the standard phylogenetic methods, using simulated — artificially evolved — nucleotide sequences and the 22 mitochondrial tRNA genes of the great apes. The initial tree of artificial sequence evolution and the generally well established phylogeny of Hominidae were used as a benchmark. Considering the reliability values, the authors used different phylogenetic methods for the simulated sequence trees, and the ICF appeared to be the most reliable method. Although

in some cases the results were the same as with the traditional methods, the speed-performance was much more appealing.

The "weighted node count" metric introduced in this paper is a generalized method for calculating distances between ICF-graphs in comparison with the calculations used in the previous investigations. This new approach is more flexible due to the fact that we can choose the weight function freely, while in the original algorithm fix rules are considered, which in some cases may cause minor data loss in the process. Since the original method uses merged ICF-graphs for the distance calculation, it is possible that there are nodes with the same label but with a different coloration in the graphs. This was solved originally by simply ignoring the conflicting white nodes. In contrast, the newly proposed modified approach will take this case into account.

An other prospective application of this recently developed methods is related to pattern recognition and classification in medical diagnostics (Kézdi, et al., 2011, unpublished results)

Acknowledgment. We thank Eszter Ari and Péter Ittzés for their useful comments.

References

[1] **Ari, E., P. Ittzés, J. Podani, T. Quynh and É. Jakó,** Comparison of Boolean analysis and standard phylogenetic methods using artificially evolved and natural mt-tRNA sequences from great apes, *Mol Phylogenet Evol.* (in press).

[2] **Frolov, A. and É. Jakó,** Algorithms for recognition of partially ordered objects and their application, *Scripta Technica*, (1991), 109–118, Originally published in *Tekhnicheskaja Kibernetika*, **N5** (1990), 95–104 (in Russian).

[3] **Goddard, W. and H.C. Swart,** Distances between graphs under edge operations,
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.5311&rep=rep1&type=pdf`
(available: 2011.12.19.)

[4] **Ittzés, P., É. Jakó, Á. Kun, A. Kun and J. Podani,** A discrete mathematical method for the analysis of spatial pattern, *Community Ecology*, **6(2)** (2005), 177–190.

[5] **Jakó, É.,** *Iterative Canonical Decomposition of Boolean Functions and its Application to Logical Circuits Design*, Ph.D. Thesis, Moscow, 1983 (in Russian).

[6] **Jakó, É. and I. Fábián,** Programmable logic array (PLA). *Hungarian Patent No 203 173 INT CL: HO3K 19/173.* Priority data: 28 April 1994.

[7] **Jakó, É. and P. Ittzés,** A discrete mathematical approach to spatial pattern analysis in vegetation, *Abstracta Botanica*, **22** (1998), 121–142.

[8] **Jakó, É.,** Generalized Boolean descriptors for biological macromolecules, *Proceedings of American Institute of Physics (AIP)*, **2** (2007), 552–557.

[9] **Jakó, É., E. Ari, P. Ittzés, A. Horváth and J. Podani,** BOOL-AN: A method for comparative sequence analysis and phylogenetic reconstruction, *Mol Phylogenet Evol.*, **52(3)** (2009), 887–97.

[10] **Pásztor Varga, K. and M. Várterész,** Many-valued Logic, Mappings, ICF Graphs, Normal Forms, *Annales Univ. Sci. Budapest., Sect. Comp.* **31** (2009), 185–202.

[11] **Riesen, K., M. Neuhaus and H. Bunke,** Bipartite graph matching for computing the edit distance of graphs, *GraphBased Representations in Pattern Recognition*, **4538** (2007), 1–12.

N. Kézdi and K. Pásztor Varga
Department of Programming Languages and Compilers
Faculty of Informatics
Eötvös Loránd University
H-1117 Budapest, Pázmány P. sétány I/C
Hungary
kezdi@inf.elte.hu
pkata@ludens.elte.hu

É. Jakó
Department of Plant Systematics, Ecology and Theoretical Biology
Faculty of Science
Eötvös Loránd University
H-1117 Budapest, Pázmány P. sétány I/C
Hungary
jako@elte.hu