

## **SESSION**

# **MODELING, SIMULATION, AND RELATED ALGORITHMS + DATA STRUCTURES + OPERATING SYSTEMS**

**Chair(s)**

**TBA**



# On Modeling Inhibitor Nets with Interval Processes and Interval Traces

Mohammed Alqarni and Ryszard Janicki

Computing and Software Department, McMaster University, Hamilton, Ontario, Canada  
alqarnma@mcmaster.ca, janicki@mcmaster.ca

**Abstract**—Two interval semantics for elementary inhibitor Petri nets, interval process semantics and interval trace semantics are discussed and proved equivalent.

**Keywords:** inhibitor Petri nets, interval processes, interval traces, semantics

## I. INTRODUCTION

It is commonly assumed (first argued by N. Wiener in 1914 [26] and analyzed in details in [10]) that any system run (execution) that can be observed by a single observer *must* be an interval order of event occurrences. This means that the most precise observational semantics is defined in terms of interval orders. Moreover, representing observations as interval orders allows to capture behaviours that neither of the standard semantics can really describe. However generating interval orders directly is problematic for most models of concurrency, as the only feasible sequence representation of interval order is by using Fishburn Theorem [6] and appropriate sequences of *beginnings* and *endings* of events involved (cf. [10]).

Elementary nets with inhibitor arcs [11] are very simple. They are just classical elementary nets of [21], [24] extended with inhibitor arcs. However they can model extremely complicated behaviours [2], [4] that cannot easily (if not at all) be represented by other models. For example they can model the case when a simultaneous execution of events  $a$  and  $b$  and the order  $a$  followed by  $b$  are allowed, but the order  $b$  followed by  $a$  is forbidden, so called ‘ $a$  not after  $b$ ’ case [9], [16]. This case cannot be represented by classical Place/Transition Nets [16]. Hence, the elementary nets with inhibitor arcs are an excellent medium for novel models of behaviours.

Interval process semantics of elementary inhibitor Petri nets has been proposed and analyzed in [3] and interval traces have been introduced in [12] and further developed in [13]. The interval processes of [3] are an extension and generalization of step-sequence process semantics of elementary inhibitor Petri nets proposed in [11] and improved in [15]; while the interval traces are a generalization of classical Mazurkiewicz traces [5], [18].

In this paper we introduce an interval traces semantics of elementary inhibitor Petri nets and show that this semantics is equivalent to the interval process semantics of [3].

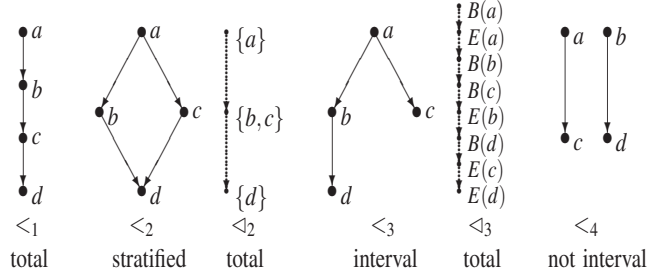


Figure 1: Various types of partial orders (represented as Hasse diagrams). The interval order  $\prec_3$  is (not uniquely) represented by a sequence that represents  $\prec_3$ , i.e.  $B(a)E(a)B(b)B(c)E(b)B(d)E(c)E(d)$ .

The process semantics as proposed in [3], i.e. in the style of [15], [21], does not usually require much validation as intuitively it is just a set of system unfoldings, so it is as natural as any operational semantics. Hence, the results of this paper can also be interpreted as a validation of the interval traces semantics.

## II. PARTIAL ORDERS AND MAZURKIEWICZ TRACES

In this section, we recall some known mathematical concepts, notations and results that will be used frequently in this paper.

A relation  $R \in X \times X$  is an *equivalence relation*, if it is *reflexive*, *symmetric* and *transitive*, i.e. for all  $a, b, c \in X$ ,  $aRa$ ,  $aRb \implies bRa$  and  $aRbRc \implies aRc$ .

If  $R$  is an equivalence relation than for every  $x \in X$ , the set  $[x]_R = \{y \mid xRy\}$  is the *equivalence class containing  $x$* .

For every relation  $R$ , the relation  $R^* = \bigcup_{i=0}^{\infty} R^i$ , where  $R^0$  is the identity relation, is the *reflexive and transitive closure* of  $R$ .

**Definition 1.** A relation  $\prec \subseteq X \times X$  is a (strict) **partial order** if it is *irreflexive* and *transitive*, i.e. for all  $a, c, b \in X$ ,  $a \not\prec a$  and  $a \prec b \prec c \implies a \prec c$ . We also define:

$$a \frown b \stackrel{df}{\iff} \neg(a \prec b) \wedge \neg(b \prec a) \wedge a \neq b,$$

$$a \frown b \stackrel{df}{\iff} a \prec b \vee a \frown b.$$

Note that  $a \frown b$  means  $a$  and  $b$  are *incomparable* (w.r.t.  $\prec$ ) elements of  $X$ .  $\square$

Let  $\prec$  be a partial order on a set  $X$ . Then:

- 1)  $<$  is *total* if  $\sim_{<} = \emptyset$ . In other words, for all  $a, b \in X$ ,  $a < b \vee b < a \vee a = b$ . For clarity, we will reserve the symbol  $\triangleleft$  to denote total orders;
- 2)  $<$  is *stratified* if  $a \sim_{<} b \sim_{<} c \implies a \sim_{<} c \vee a = c$ , i.e., the relation  $\sim_{<} \cup id_X$ , where  $id_X$  is the identity on  $X$ , is an equivalence relation on  $X$ ;
- 3)  $<$  is *interval* if for all  $a, b, c, d \in X$ ,  $a < c \wedge b < d \implies a < d \vee b < c$ , i.e.,  $<$  has no restriction that is isomorphic to  $<_4$  from Figure 4.

It is clear from these definitions that every total order is stratified and every stratified order is interval. The following simple concept will often be used in this paper.

For every partial order  $<$ , we define

$$\text{Total}(<) \stackrel{df}{=} \{ \triangleleft \subseteq X \times X \mid \triangleleft \text{ is a total order and } < \subseteq \triangleleft \}.$$

In other words, the set  $\text{Total}(<)$  consists of all the *total order extensions* of  $<$ .

By Szpilrajn's Theorem [25], we know that every partial order  $<$  is uniquely represented by the the set  $\text{Total}(<)$ . Szpilrajn's Theorem can be stated as follows:

**Theorem 1** (Szpilrajn [25]). *For every partial order  $<$ ,*

$$< = \bigcap_{\triangleleft \in \text{Total}(<)} \triangleleft. \quad \square$$

For the interval orders, the name and intuition follow from Fishburn's Theorem:

**Theorem 2** (Fishburn [6]). *A partial order  $<$  on  $X$  is interval iff there exists a total order  $\triangleleft$  on some  $T$  and two mappings  $B, E : X \rightarrow T$  such that for all  $x, y \in X$ ,*

$$1. B(x) \triangleleft E(x) \quad 2. x < y \iff E(x) \triangleleft B(y) \quad \square$$

Usually  $B(x)$  is interpreted as the beginning and  $E(x)$  as the end of an *interval*  $x$ . The intuition of Fishburn's theorem is illustrated in Figure 4 with  $<_3$  and  $\triangleleft_3$ . For all  $x, y \in \{a, b, c, d\}$ , we have  $B(x) \triangleleft_3 E(x)$  and  $x <_3 y \iff E(x) \triangleleft_3 B(y)$ . For better readability in the future we will skip parentheses in  $B(x)$  and  $E(x)$ , and just write  $Bx$  and  $Ex$ .

**Definition 2** ([5], [17], [18]). 1) *Let  $\Sigma$  be a finite set and let the relation  $ind \subseteq \Sigma \times \Sigma$  be an irreflexive and symmetric relation (called independency). The pair  $(\Sigma, ind)$  is called a trace alphabet.*

- 2) *Let  $\approx \in \Sigma^* \times \Sigma^*$  be a relation defined as follows:*  
 $x \approx y \iff \exists x_1, x_2 \in \Sigma^*. \exists (a, b) \in ind. x = x_1 a b x_2 \wedge y = x_1 b a x_2$
- 3) *Let  $\equiv_{ind}$  the reflexive and symmetric closure of  $\approx$ , i.e.  $\equiv_{ind} = \approx^*$ . Clearly is an equivalence relation.*
- 4) *For every  $x \in \Sigma$ , the equivalence class  $[x]_{\equiv_{ind}}$  is called a **Mazurkiewicz trace**, or just a **trace**.  $\square$*

We will often write  $[x]$  or  $[x]_{ind}$  instead of  $[x]_{\equiv_{ind}}$ .

One may show that  $[x][y] = [x] \circ [y] = [xy]$ , where  $\circ$  is a concatenation of sets of sequences, a symbol that is usually omitted [5], [18].

Formally, an algebra of Mazurkiewicz traces is a quotient equational monoid over sequences [5], [17], [18].

**Example 1.** *Let  $\Sigma = \{a, b, c\}$ ,  $ind = \{(b, c), (c, b)\}$ . Given three sequences  $s = abc bca$ ,  $s_1 = abc$  and  $s_2 = bca$ , we can generate the traces  $[s] = \{abc bca, abc cba, acb bca, acb cba, abbcca, accbba\}$ ,  $[s_1] = \{abc, acb\}$  and  $[s_2] = \{bca, cba\}$ . Note that  $[s] = [s_1][s_2]$  since  $[abc bca] = [abc][bca] = [abc bca]$ .  $\square$*

Each sequence of events represents a total order of *enumerated events* in a natural way. For precise definitions see for example [11], here we will be using the following notation.

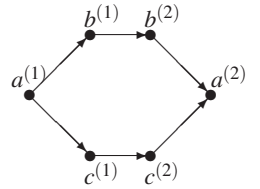
- 1) For each set of events  $\Sigma$ , let  $\widehat{\Sigma} = \{a^{(i)} \mid a \in \Sigma, i \geq 1\}$ .
- 2) For each sequence  $s \in \Sigma^*$ , let  $\hat{s} \in \widehat{\Sigma}^*$  denote its enumerated representation. For example if  $s = abbaa$  then  $\hat{s} = a^{(1)}b^{(1)}b^{(2)}a^{(2)}a^{(3)}$ .
- 3) For each sequence  $s \in \Sigma^*$ ,  $\widehat{\Sigma}_s$  denotes the set of all *enumerated events* of  $s$ . For example  $\widehat{\Sigma}_{abbaa} = \{a^{(1)}, a^{(2)}, a^{(3)}, b^{(1)}, b^{(2)}\}$ .
- 4) For each trace  $[s]$ , we define  $\widehat{\Sigma}_{[s]} = \widehat{\Sigma}_s$ .
- 5) For ever  $s \in \Sigma^*$ ,  $\triangleleft_s$  is a *total order* defined by the enumerated sequence  $\hat{s}$ . For example  $\triangleleft_{abbaa} = a^{(1)} \rightarrow b^{(1)} \rightarrow b^{(2)} \rightarrow a^{(2)} \rightarrow a^{(3)}$ .

**Definition 3** ([18]). *For every trace  $[x]$ , the partial order*

$$\triangleleft_{[x]}^{trace} = \bigcap_{s \in [x]} \triangleleft_s$$

*is called the partial order generated by  $[x]$ .  $\square$*

**Example 2.** *For the trace  $[s] = [abc bca]$  from Example 1, we have  $\widehat{\Sigma}_{[s]} = \{a^{(1)}, b^{(1)}, c^{(1)}, b^{(2)}, c^{(2)}, a^{(2)}\}$ . The partial order  $\triangleleft_{[s]}^{trace}$  generated by  $[s]$  is depicted as Hasse diagram on the right.  $\square$*



### III. INTERVAL TRACES

The interval traces, introduced in [12] and refined in [13], stem from Mazurkiewicz traces [17], [18] and Fishburn's representation of interval orders [6].

Let  $\Sigma$  be a finite set (of events), and let

$$\mathcal{E}_\Sigma = \{Ba \mid a \in \Sigma\} \cup \{Ea \mid a \in \Sigma\},$$

be the set of all beginnings and ends of events in  $\Sigma$ . We will often just write  $\mathcal{E}$  instead of  $\mathcal{E}_\Sigma$ . Every sequence from  $x \in \mathcal{E}^*$  defines a total order  $to(x)$ , however not every such total order can be interpreted as a representation of some interval order. or example  $BaBcBb$  represents no interval order.

Let  $\mathcal{D} \subseteq \mathcal{E}$  and let  $s \in \mathcal{D}^*$ . We standardly define the projection of  $s$  onto  $\mathcal{D}$  as:  $\pi_{\mathcal{D}}(s) \stackrel{df}{=} \varepsilon$ , and

$$\pi_{\mathcal{D}}(s\alpha) \stackrel{df}{=} \begin{cases} \pi_{\mathcal{D}}(s)\alpha & \text{if } \alpha \in \mathcal{D}, \\ \pi_{\mathcal{D}}(s) & \text{if } \alpha \notin \mathcal{D}. \end{cases}$$

For example  $\pi_{\{Ba, Ea\}}(BbBaEbBaEaEc) = BaBaEa$  and  $\pi_{\{Ba, Ea, Bc, Ec\}}(BbBaEbBaEaEc) = BaBaEaEc$ .

We say that a string  $x \in \mathcal{E}^*$  is an *interval sequence* iff

$$\forall Bt, Et \in \mathcal{E}^*. \pi_{\{Bt, Et\}}(x) \in (BtEt)^*.$$

We will write  $\text{InSeq}(\mathcal{E}^*)$  to denote the set of all interval sequences of  $\mathcal{E}^*$ . For example  $BbBaEbBaEaEc \notin \text{InSeq}(\mathcal{E}^*)$ , while  $BaBcBbEbEaEc \in \text{InSeq}(\mathcal{E}^*)$ .

**Definition 4** ([12]). Let  $x \in \text{InSeq}(\mathcal{E}_\Sigma^*)$ , and let  $\triangleleft_x$  be a relation on  $\hat{\Sigma}$ , defined by

$$a^{(i)} \triangleleft_x b^{(j)} \iff Ea^{(i)} \triangleleft_x Bb^{(j)}.$$

By Theorem 2, the relation  $\triangleleft_x$  is an interval order, and it is called the interval order defined by the sequence  $x$  of beginnings and ends.  $\square$

For example if  $x = BaEaBbBcEbBdEcEd$  then  $\triangleleft_x$  is the interval order  $<_3$  from Figure 1.

**Definition 5** ([12]). Let  $\text{ind} \subseteq \mathcal{E} \times \mathcal{E}$  be a symmetric and irreflexive relation such that for all  $a, b \in \Sigma$

- 1)  $(Ba, Ea) \notin \text{ind}$  and  $(Ea, Ba) \notin \text{ind}$ ,
- 2)  $(Ba, Bb) \in \text{ind}$  and  $(Ea, Eb) \in \text{ind}$ .

The relation  $\text{ind}$  is called *interval independency*, and the pair  $(\mathcal{E}, \text{ind})$  is called *interval trace alphabet*.  $\square$

The condition (1) above follows from the fact that in any representation of any order, the beginning of an event always precede the end so that cannot commute. The condition (2) follows from the generalization of the observation that the interval sequences  $BaBbEaEb$ ,  $BbBaEaEb$ ,  $BaBbEbEa$ , and  $BbBaEbEa$  represent the same fact, namely that  $a$  and  $b$  are simultaneous.

The interval traces are defined as a special distinctive class Mazurkiewicz traces.

**Definition 6** ([12]). A trace  $[x]_{\text{ind}}$  over the interval trace alphabet  $(\mathcal{E}, \text{ind})$  is called an *interval trace* if  $[x]_{\text{ind}} \subseteq \text{InSeq}(\mathcal{E}^*)$ .  $\square$

The soundness of the above definition follows from the following non-trivial result.

**Proposition 1** ([13]). Let  $(\mathcal{E}, \text{ind})$  be an interval trace alphabet, and let  $x, y \in \text{InSeq}(\mathcal{E}^*)$ .

- 1) For each  $x, y \in \mathcal{E}^*$ , if  $x \in \text{InSeq}(\mathcal{E}^*)$  and  $y \in \text{InSeq}(\mathcal{E}^*)$  then  $xy \in \text{InSeq}(\mathcal{E}^*)$ .
- 2) For each  $s \in \mathcal{E}^*$ , we have:  
 $s \in \text{InSeq}(\mathcal{E}^*) \iff \forall x \in [s]_{\text{ind}}. x \in \text{InSeq}(\mathcal{E}^*)$ .
- 3) For each  $x, y \in \mathcal{E}^*$ ,  
if  $[x]_{\text{ind}} \subseteq \text{InSeq}(\mathcal{E}^*)$  and  $[y]_{\text{ind}} \subseteq \text{InSeq}(\mathcal{E}^*)$ , then  
 $[x]_{\text{ind}}[y]_{\text{ind}} = [xy]_{\text{ind}} \subseteq \text{InSeq}(\mathcal{E}^*)$ .
- 4)  $\triangleleft_x = \triangleleft_y \iff x \equiv_{\text{ind}} y$ .  $\square$

As a partial orders generator, each interval trace can be interpreted twofold. First, it is also a Mazurkiewicz trace so it generates a partial order by Definition 3, second, each element of the interval trace is an interval sequence, so the trace can also be interpreted as representing a set of appropriate interval orders.

**Definition 7.** Let  $[x] \subseteq \text{InSeq}(\mathcal{E}^*)$  be an interval trace.

- 1) The partial order  $\triangleleft_{[x]}^{\text{trace}}$  defined as:

$$\triangleleft_{[x]}^{\text{trace}} = \bigcap_{s \in [x]} \triangleleft_s$$

is called *canonical order* defined by  $[x]$ .

- 2) The set  $\text{interv}^{\text{trace}}([x]) = \{\triangleleft_t \mid t \in [x]\}$

is the set of all *interval orders defined by*  $[x]$ .  $\square$

Both the canonical order and the interval orders defined by an interval trace will be used to show the equivalence of interval order semantics and interval process semantics for elementary inhibitor nets.

#### IV. ELEMENTARY NETS WITH INHIBITOR ARCS

*Inhibitor arcs* allow a transition to check for an *absence* of a token. They have been introduced in [2] to solve a synchronization problem not expressible in classical Petri nets. In principle they allow ‘test for zero’, an operator the standard Petri nets do not have (c.f. [20], [23]). *Activator arcs* (also called ‘read’, or ‘contextual’ arcs [4], [19]), formally introduced in [11], [19], are conceptually orthogonal to the inhibitor arcs, they allow a transition to check for a *presence* of a token.

*Elementary nets with inhibitor arcs* [11] are very simple. They are just classical *elementary nets* of [21], [24] extended with inhibitor arcs. Nevertheless they can easily express complex behaviours involving ‘not later than’ cases or non-transitive simultaneity as illustrated in Figure 2.

**Definition 8** ([2], [11]). An *Elementary Net with Inhibitor Arcs* (ENI) is a tuple  $\mathbb{N} = (P, T, F, C_{\text{init}}, I)$  such that

- $P$  and  $T$  are finite and disjoint sets of places and transitions represented, respectively, as circles and rectangles;
- $F \subseteq (P \times T) \cup (T \times P)$  is the flow relation of  $\mathbb{N}$  - represented as directed arcs between places and transitions;
- $C_{\text{init}} \subseteq P$  is the initial marking of  $\mathbb{N}$  (generally, any  $C \subseteq P$  is a marking); and
- $I \subseteq P \times T$  is a set of inhibitor arcs - represented as arcs with small circles as arrowheads.  $\square$

The net  $\mathbb{N}$  in Figure 2 is an example of ENI. For every  $x \in P \cup T$  we define, its input  $\bullet x = \{y \mid (y, x) \in F\}$  and its output  $x^\bullet = \{y \mid (x, y) \in F\}$ . We assume that for every  $t \in T$ ,  $\bullet t \neq \emptyset \neq t^\bullet$  and  $\bullet t \cap t^\bullet = \emptyset$ . Moreover, for each  $t \in T$ , the set  ${}^\circ t = \{p \mid (p, t) \in I\}$  is the set of places that are connected with transition  $t$  by inhibitor arcs. We also standardly define for any subset  $U$  of  $T$ :  $\bullet U = \bigcup_{t \in U} \bullet t$ ,  $U^\bullet = \bigcup_{t \in U} t^\bullet$  and  ${}^\circ U = \bigcup_{t \in U} {}^\circ t$ .

The operational semantics of ENI is defined through the ‘token game’ which simulates the occurrence of transitions and the changes of tokens in places. ENI differs from ordinary elementary Petri nets only by introducing a requirement that a transition cannot be enabled if there is a token in a place to which it is connected by an

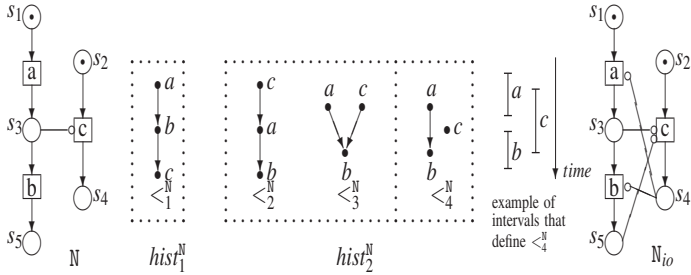


Figure 2: Inhibitor nets  $N$  and  $N_{io}$  and all their behaviours involving one occurrence of  $a$ ,  $b$  and  $c$ . The net  $N$  generates  $\prec_1^N, \prec_2^N, \prec_3^N, \prec_4^N$  and two concurrent histories, while  $N_{io}$  generates only an interval order  $\prec_4^N$ . Partial orders are represented by Hasse diagrams. The net  $N_{io}$  generates only the interval order  $\prec_4^N$ .

inhibitor arc. A transition  $t$  is enabled at a configuration  $C$  if  $\bullet t \subseteq C$  and  $(t^\circ \cup t^\circ) \cap C = \emptyset$ . An enabled transition  $t$  can fire leading to a new configuration  $C' = (C \setminus \bullet t) \cup t^\circ$ . We denote this by  $C[t]C'$ . We will also write  $C[t_1 \dots t_n]C'$  if  $C[t_1]C_1 \dots C_{n-1}[t_n]C'$  for some configurations  $C_1, \dots, C_{n-1}$ .

There are two standard operational semantics for ENI, one in terms of **firing sequences** and another in terms of **firing step sequences** (c.f. [11]), however in this paper we will only use the firing sequenc semantics.

**Definition 9.** A **firing sequence** of an ENI is any sequence of transitions  $t_1, \dots, t_n$  for which there are markings  $C_1, \dots, C_n$  satisfying:

$$C_{init}[t_1]C_1[t_2]C_2 \dots [t_n]C_n. \quad \square$$

## V. INTERVAL ELEMENTARY NET WITH INHIBITOR ARCS

Following [3] in this section we show how a given inhibitor net can generate appropriate sequences of event beginnings and ends, so we will be able to describe all interval orders the net generates.

The basic idea of defining the set of firing interval sequences for a given inhibitor net  $N$  is briefly presented in Figure 3 as a transformation of  $N$  into  $\mathcal{N}$ .

We assume that the events (transitions) are not instantaneous, on contrary, they are interpreted as representations of activities whose completion takes some time. However, their beginnings and ends are instantaneous.

In principle the transformation is based on the replacement of a transition  $t$  by the net  $\boxed{Bt} \rightarrow (t) \rightarrow \boxed{Et}$  as first proposed in [27] and additionally taking into account specific behaviours induced by inhibitor arcs.

**Definition 10** ([3]). Let  $N = (P, T, F, I, C_{init})$  be an ENI system. We define  $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{I}, C_{init})$ , its **interval representation** as follows:

- $\mathcal{P} = P \cup T$
- $\mathcal{T} = \{Bt \mid t \in T\} \cup \{Et \mid t \in T\}$
- $\forall p \in P, t \in T. (p, t) \in F \iff (p, Bt) \in \mathcal{F}$

- $\forall p \in P, t \in T. (t, p) \in F \iff (Et, p) \in \mathcal{F}$
- $\forall t \in T. (Bt, t), (t, Et) \in \mathcal{F}$
- $\forall p \in P, t \in T. (p, t) \in I \iff (p, Bt) \in \mathcal{I} \wedge (\forall r \in p^\circ)(r, Bt) \in \mathcal{I}. \quad \square$

For the detailed arguments that the net  $\mathcal{N}$  fully describes the behaviour of the net  $N$  the reader is referred to [3].

The nets  $N$  and  $\mathcal{N}$  in Figure 3 illustrate the above definition. Note that for example each of the following sequences  $BaBcEaBbEbEc$ ,  $BaBcEaBbEcEb$ ,  $BcBaEaBbEbEc$  and  $BcBaEaBbEcEb$  are *firing sequences* of  $\mathcal{N}$ , and each of them represents the *interval order*  $\prec_4^N$  from Figure 2 via Fishburn Theorem (Theorem 2). *This means that event  $b$  follows event  $a$  and event  $c$  overlaps both events  $a$  and  $b$  in the original net  $N$ .*

Directly from the above definition we have:

**Fact 1** ([3]). Let  $N = (P, T, F, I, C_{init})$  be an ENI system and  $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{I}, C_{init})$  its interval representation. Then for each  $t \in T$  we have:  $\bullet Bt = \bullet t$ ,  $Bt^\circ = \{t\}$ ,  $\bullet Et = \{t\}$ ,  $Et^\circ = t^\circ$ ,  $Bt^\circ = t^\circ \cup (t^\circ)^\circ$ , and  $Et^\circ = \emptyset$ .  $\square$

Since  $\mathcal{N}$  is just another inhibitor net, we may try to use the standard definition of a firing sequence from Definition 9. We will write  $\llbracket \dots \rrbracket$  instead of  $[\dots]$  to indicate firing in  $\mathcal{N}$  (or  $\mathcal{EN}$ ) and not in  $N$ .

**Definition 11** ([3]). Let  $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{I}, C_{init})$  be an interval ENI. A sequence  $x = \alpha_1 \dots \alpha_n \in \mathcal{T}^*$  is an **interval firing sequence** of  $\mathcal{N}$  if there are markings  $C_1, \dots, C_n$  satisfying:

$$C_{init} \llbracket t_1 \rrbracket C_1 \llbracket t_2 \rrbracket C_2 \dots \llbracket t_n \rrbracket C_n. \quad \square$$

The following result validates the above definition.

**Proposition 2** ([3]). If  $x$  is an interval firing sequence of  $\mathcal{N}$ , then  $x \in \text{InSeq}(\mathcal{T}^*)$ .  $\square$

Since all transitions of interval ENI's are instantaneous, simultaneous executions of any kind are disallowed, so the only operational semantics is the firing sequences semantics. The firing step sequences, as in [11], [15] do not make any sense in this case.

The net  $\mathcal{N}$  from Figure 3 have ten interval firing sequences that involve all elements of  $\mathcal{T} = \{Ba, Ea, Bb, Eb, Bc, Ec\}$ , namely  $BaEaBbEbBcEc$  - which represents a total order  $\prec_1^N$  from Figure 2;  $BcEcBaEaBbEb$  - which represents a total order  $\prec_2^N$ ;  $BaBcEcEaBbEb$ ,  $BaBcEaEcBbEb$ ,  $BcBaEcEaBbEb$ ,  $BcBaEaEcBbEb$  - all four represent a stratified order  $\prec_3^N$  of Figure 2; and  $BaBcEaBbEbEc$ ,  $BaBcEaBbEcEb$ ,  $BcBaEaBbEbEc$ ,  $BcBaEaBbEcEb$  - all four represent an interval order  $\prec_4^N$  of Figure 2. It is important to stress that if observations are not allowed to be recorded as interval firing sequences, then  $\prec_4^N$  cannot be generated. It can neither be generated by firing sequence nor by firing step-sequence. This order is an interval order, but it is not stratified, so

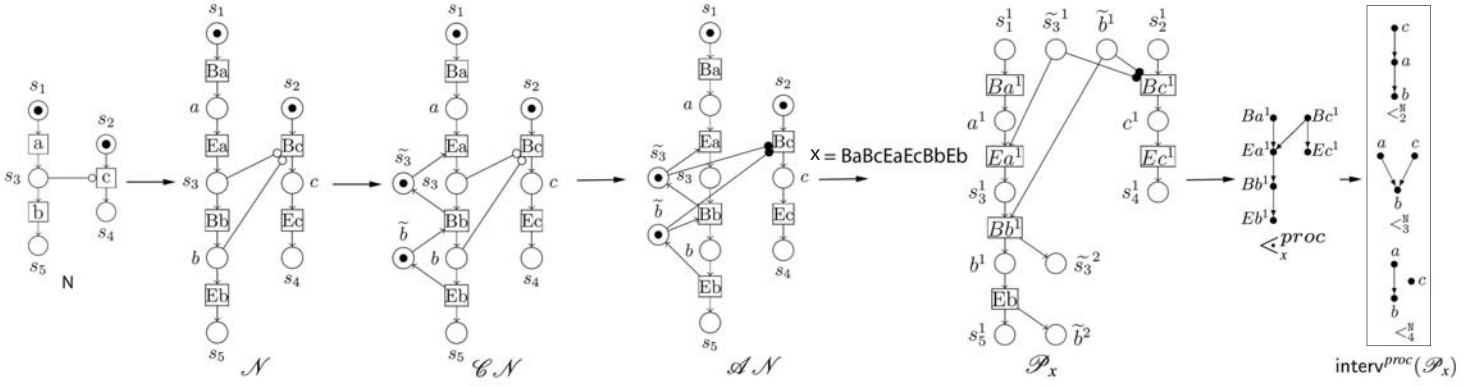


Figure 3: An example of an inhibitor net, its interval representation, a process and a concurrent histories it generates.

step-sequences (as in [11], [15]) do not work.

It was shown in [3] that if all interval orders generated by a net are stratified then observational semantics of this model is equivalent to that of [11], [15].

The net  $N_{io}$  from Figure 2 can generate neither any firing sequence nor any step-sequence. It can generate only the interval order  $\prec_4^N$  (see [3], [13] for details).

## VI. PROCESS SEMANTICS

In case of concurrent systems many of system runs/executions are equivalent, but this aspect is difficult to capture when only operational semantics is considered. Abstractions of these equivalent executions are often called *concurrent histories*, and, dependently on the assumptions about systems and systems runs, are usually modelled by partial orders [7], [21], stratified order structures or interval order structures (c.f. [8]), or *processes* (c.f. [15]).

For the net  $N$  from Figure 2, the runs  $\prec_2^N, \prec_3^N, \prec_4^N$  are equivalent as in all cases we have event  $c$  occurs no later than event  $a$ , so  $N$  has two concurrent histories involving all three events  $a, b, c$ . In  $\prec_1^N$ ,  $a$  and  $b$  occur before  $c$ , so  $\prec_1^N$  belongs to a different concurrent history<sup>1</sup> (see [8], [9] for details).

For Petri nets, *processes* are plain or modified unfoldings, called *occurrence nets*. It was shown in [11], [15] that for nets *with* inhibitor arcs, plain unfolding does not work, since the absence of a token, unlike the presence of a token, cannot be tested. Hence we have to replace inhibitor arcs by appropriate activator arcs. The idea is that an *inhibitor arc* which tests whether a place is empty, can be simulated by an *activator arc* which tests whether its *complement place* is not empty. To do such simulation, each inhibitor place must have its complements, if it does not we can always add it, as it does not change the net behaviour (c.f. [7], [11], [15], [21]). This construction is illustrated in Figure 3, where adding complement places changes  $\mathcal{N}$  into  $\mathcal{C}\mathcal{N}$ . Clearly

<sup>1</sup>Concurrent history is a set of runs that agree on causality invariants as “always earlier than” or “always not later than” (see [8], [9] for formal arguments).

the behaviours of  $\mathcal{N}$  and  $\mathcal{C}\mathcal{N}$  are identical (c.f. [11], [15], [21] for details).

**Definition 12** ([7], [15]). 1) Places  $p, q \in P$  are **complementary** ( $p$  is a complement of  $q$  and vice versa) if  $p \neq q$ ,  $\bullet p = q^\bullet$  and  $p^\bullet = \bullet q$ , and  $|C_{init} \cap \{p, q\}| = 1$ .  
2) An elementary inhibitor net is **complement closed** if every inhibitor place has its complement, i.e.  
 $(p, t) \in I \implies \tilde{p} \in P$ .  $\square$

If  $p$  and  $q$  are complementary we will write  $p = \tilde{q}, q = \tilde{p}$ , and clearly  $p = \tilde{\tilde{p}}, q = \tilde{\tilde{q}}$ .

We define the *processes generated by a firing sequence*  $y = t_1 \dots t_n$  as  $P_y = N_n$ , where  $N_n$  is the last *activator occurrence net* in the sequence  $N_0, \dots, N_n$ . Each net  $N_k = (B_k, E_k, R_k, A_k), 0 \leq k \leq n$ , is a net with *activator arcs* that model an unfolding of the net  $N$  by the sequence  $t_1 \dots t_k$ . The first three components of  $N_k$  correspond to places  $P$ , transitions  $T$ , and flow relation  $F$  of the underlying ENI system, while  $A_k \subseteq B_k \times E_k$  is the set of activator arcs derived from inhibitor arcs  $I$ .

The elements of  $B_k \cup E_k$  are of the form  $r^i$ , where  $r \in P \cup T$  and  $i \geq 1$ . We will denote  $l(r^i) = r$  and  $n(r^i) = i$ . Moreover, for every  $r \in P \cup T$  and  $k \leq n$ ,  $\Delta r$  is the number of nodes of  $N_{k-1}$  labelled by  $r$  (i.e. the number of  $\alpha \in B_k \cup E_k$  such that  $l(\alpha) = r$ ).

**Algorithm 1** (Constructing  $P_y$ , for  $y = t_1 \dots t_n$ , [11], [15]).

- Step 0.  $N_0 = (\{(p^1) \mid p \in C_{init}\}, \emptyset, \emptyset, \emptyset)$
- Step  $k$ . Given  $N_{k-1}$ , we define  $N_k$  in the following way:
  - $B_k = B_{k-1} \cup \{p^{1+\Delta p} \mid p \in t_k^\bullet\}$
  - $E_k = E_{k-1} \cup \{t_k^{1+\Delta t_k}\}$
  - $R_k = R_{k-1} \cup \{(p^{\Delta p}, t_k^{1+\Delta t_k}) \mid p \in \bullet t_k\} \cup \{(t_k^{1+\Delta t_k}, p^{1+\Delta p}) \mid p \in t_k^\bullet\}$
  - $A_k = A_{k-1} \cup \{(\tilde{p}^{\Delta \tilde{p}}, t_k^{1+\Delta t_k}) \mid p \in t_k^\circ\}$   $\square$

The above algorithm is illustrated in Figure 3 (a part from  $\mathcal{C}\mathcal{N}$  to  $\mathcal{P}_x$ ). When it is applied to the net  $\mathcal{C}\mathcal{N}$  with the sequence  $x = BaBcEaEcBaBb$  it results in the process  $\mathcal{P}_x$ . Intuitively it is just a plain unfolding of the net  $\mathcal{A}\mathcal{N}$ .

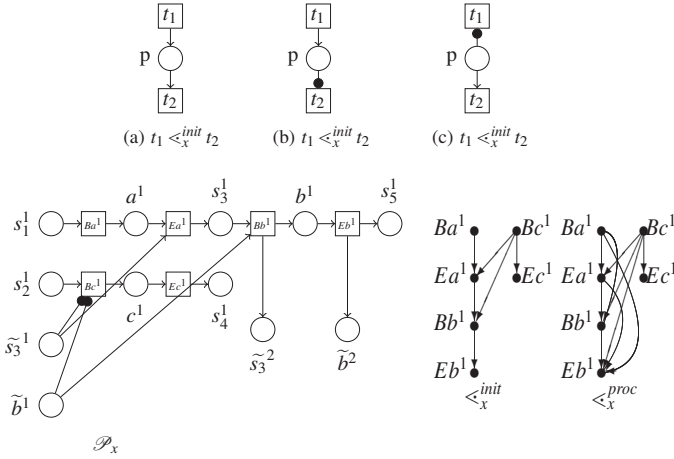


Figure 4: An illustration of Definition 13(1) (top) and Definition 13(2) (bottom).

An extension of Algorithm 1 to the case of firing step sequence  $x = U_1 \dots U_n$  was first proposed in [11] and refined in [15] is rather straightforward, but in will not be discussed here as it will not be used.

Algorithm 1 can be applied to *any* elementary net with inhibitor arcs, however in this paper we will use it only to *complement closed interval representations*, as  $\mathcal{C}\mathcal{N}$  in Figure 3.

## VII. INTERVAL PROCESSES

We will now introduce interval processes and show how they represents interval runs/executions.

Let  $\mathbb{N} = (P, T, F, I, C_{init})$  be an ENI system,  $\mathcal{C}\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{I}, C_{init})$  be its complement closed interval representation and let  $x = \alpha_1 \dots \alpha_n$  be an interval firing sequence of  $\mathcal{C}\mathcal{N}$ .

Assume that Algorithm 1 applied to  $\mathcal{C}\mathcal{N}$  with  $x = \alpha_1 \dots \alpha_n$  produced a process (an occurrence net)  $P_x$ .

Assume that  $P_x = \mathcal{N}_n = (\mathcal{B}_n, \mathcal{E}_n, \mathcal{R}_n, \mathcal{A}_n)$ , where  $\mathcal{N}_n$  is the last step of Algorithm 1.

A partial order  $\prec_x^{proc}$  derived from the process  $\mathcal{P}_x$  is defined as follows.

**Definition 13** ([15]). Let  $\mathcal{P}_x = \mathcal{N}_n = (\mathcal{B}_n, \mathcal{E}_n, \mathcal{R}_n, \mathcal{A}_n)$  be the process generated by  $x$ . We define a **canonical partial order**  $\prec_x^{proc}$  on  $\mathcal{E}_n$  as follows:

- 1) For all  $\alpha, \beta \in \mathcal{E}_n$ ,  
 $\alpha \prec_x^{proc} \beta \iff \alpha(\mathcal{R}_n \circ \mathcal{R}_n)\beta \vee \alpha(\mathcal{R}_n \circ \mathcal{A}_n)\beta \vee \alpha(\mathcal{A}_n^{-1} \circ \mathcal{R}_n)\beta$ ,  
 where  $\circ$  is a composition of relations.
- 2) For all  $\alpha, \beta \in \mathcal{E}_n$ ,  $\alpha \prec_x^{proc} \beta \iff \alpha(\prec_x^{init})^+ \beta$   $\square$

The above construction is illustrated in Figure 4. In most cases many different  $x$ 's can generate the same process  $\mathcal{P}_x$ , but the canonical partial order  $\prec_x^{proc}$  captures all the cases.

**Proposition 3** ([3]). For each interval firing sequence  $x$ ,  
 $\text{total}(\prec_x^{proc}) = \{\triangleleft_y \mid \mathcal{P}_x = \mathcal{P}_y\}$ .  $\square$

We will now define formally interval orders and interval order structures generated by interval firing sequences of  $\mathcal{N}$ .

**Definition 14.** Define  $E_n = \{t \mid Bt \in \mathcal{E}_n \wedge Et \in \mathcal{E}_n\}$ . Let  $x \in \text{InSeq}(\mathcal{T}^*)$ , and let  $\triangleleft_x$  be a relation on  $E_n$ , defined by  
 $a^i \triangleleft_x b^j \iff Ea^i \triangleleft_x Bb^j$ .

By Theorem 2 the relation  $\triangleleft_x$  is an **interval order**.  $\square$

Each  $\mathcal{P}_x$  is generated from  $\mathcal{N}$  by an interval sequence  $x$  and each interval sequence  $x$  defines an interval order  $\triangleleft_x$ . The set of all interval orders that can be derived from  $\mathcal{P}_x$  or  $\prec_x^{proc}$  is defined as follows.

**Definition 15.** For each interval firing sequence  $x$ , we define  
 $\text{interv}^{ord}(\prec_x^{proc}) = \text{interv}^{proc}(\mathcal{P}_x) = \{\triangleleft_y \mid \mathcal{P}_x = \mathcal{P}_y\}$ .  $\square$

For the example from Figure 3,  $\text{interv}^{ord}(\prec_x^{proc}) = \text{interv}^{proc}(\mathcal{P}_x) = \{\prec_2^N, \prec_3^N, \prec_4^N\}$ , i.e. the concurrent history  $\text{hist}_2^N$  from Figure 2. We will show that the same result is obtained when the interval traces approach is used.

## VIII. INTERVAL TRACE SEMANTICS OF INHIBITOR NETS

Let  $\mathbb{N} = (P, T, F, I, C_{init})$  be an ENI system and let  $\mathcal{C}\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{I}, C_{init})$  be its complement closed interval representation.

We define the **interval trace independency** relation  $\text{ind}_{\mathcal{C}\mathcal{N}} \subseteq \mathcal{T} \times \mathcal{T}$  as follows.

**Definition 16.** For all distinct  $a, b \in T$ :

- 1)  $(Ba, Bb) \in \text{ind}_{\mathcal{C}\mathcal{N}} \wedge (Ea, Eb) \in \text{ind}_{\mathcal{C}\mathcal{N}}$
- 2)  $(Ba, Eb) \in \text{ind}_{\mathcal{C}\mathcal{N}} \iff$   
 $[(Ba^\bullet \cup \bullet Ba) \cap (Eb^\bullet \cup \bullet Eb) = \emptyset] \wedge$   
 $[(Ba^\circ \cap \bullet Eb) \cup (Eb^\circ \cap \bullet Ba) = \emptyset] \wedge$   
 $[(Ba^\bullet \cap Eb^\circ) \cup (Eb^\bullet \cap Ba^\circ) = \emptyset]$ .

The **interval trace alphabet** is  $(\mathcal{T}, \text{ind}_{\mathcal{C}\mathcal{N}})$ .  $\square$

The relation  $\text{ind}_{\mathcal{C}\mathcal{N}}$  is a refinement of the similar relations from [11], [15].

**Definition 17.** Let  $x = \alpha_1 \dots \alpha_n$  be an interval firing sequence of  $\mathcal{C}\mathcal{N}$ . The interval trace  $[x]_{\text{ind}_{\mathcal{C}\mathcal{N}}}$  is the **interval trace of  $\mathcal{C}\mathcal{N}$  generated by  $x$** .  $\square$

Proposition 2 and the result below prove the soundness of the above definition.

**Proposition 4.** If  $x$  is an interval firing sequence of  $\mathcal{C}\mathcal{N}$ ,  $\mathcal{C}_{init}[\langle x \rangle] \mathcal{C}_n$  for some  $n$ , and  $y \in [x]_{\text{ind}_{\mathcal{C}\mathcal{N}}}$ , then  $\mathcal{C}_{init}[\langle y \rangle] \mathcal{C}_n$ .

*Proof.* (sketch) It suffices to show it for  $|x| = 1$ , which can be derived from the definition of  $[\dots]$ .  $\square$

For the net  $\mathcal{C}\mathcal{N}$  of Figure 3 and  $x = BaBcEaEcBbEb$ , the content of  $[x]_{\text{ind}_{\mathcal{C}\mathcal{N}}}$  consists of ten sequences analyzed at the end of Section V and  $\prec_x^{trace}$  is the same as  $\prec_x^{proc}$  from Figures 3 and 4. Moreover  $\text{interv}^{trace}([x]_{\text{ind}_{\mathcal{C}\mathcal{N}}}) = \text{interv}^{ord}(\prec_x^{proc}) = \text{interv}^{proc}(\mathcal{P}_x) = \{\prec_2^N, \prec_3^N, \prec_4^N\}$ .

We will show this kind of relationship holds in all cases.



## IX. INTERVAL PROCESSES VS INTERVAL TRACES

We start with the formal statement of the main theoretical result of this paper.

**Theorem 3** (Equivalence of Process and Interval Traces Semantics). *Let  $\mathbb{N} = (P, T, F, I, C_{init})$  be an ENI system,  $\mathcal{CN} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{I}, C_{init})$  be its complement closed interval representation and let  $x = \alpha_1 \dots \alpha_n$  be an interval firing sequence of  $\mathcal{CN}$ . The the following equations hold:*

- 1)  $\llbracket x \rrbracket_{ind_{\mathcal{CN}}}^{trace} = \llbracket x \rrbracket_x^{proc}$
- 2)  $interv^{trace}(\llbracket x \rrbracket_{ind_{\mathcal{CN}}}) = interv^{ord}(\llbracket x \rrbracket_x^{proc}) = interv^{proc}(\mathcal{P}_x)$

*Proof.* (idea.) First note that due to Proposition 3, it is relatively easy to show that (1) $\Rightarrow$ (2), so proving (1) is enough. The proof of (1) is by induction on the length of  $x$ . The proof is long, tedious, non-trivial, and plenty of different cases have to be considered.  $\square$

The above theorem is an equivalent of similar seminal results for step sequences (i.e. stratified orders) operational semantics, comtraces and stratified orders process semantics of [11], [15]. In principle it states that the in interval process semantics and interval traces semantics are equivalent for elementary inhibitor nets.

## X. FINAL COMMENT

This paper deals with the case when all observations/system runs are represented by interval orders. This is often regarded as the most general case [10], [26]. We concentrated on elementary inhibitor Petri nets, applied the interval traces semantics of [12], [13] to these kind of nets, and showed that in this case the interval process semantics proposed in [3] and the interval traces semantics are equivalent. The results of this paper can be interpreted as an extension of the ideas of [11], [15] to interval order observations.

Since the process type semantics, which is based on the concept of system unfolding, is very natural and usually does not require justification, the results of this paper can also be interpreted as some validation of the interval order semantics for inhibitor nets.

Usually the concurrent histories involving interval or stratified orders and “not later than” phenomenon are represented by interval or stratified order structures (c.f. [3], [8], [11], [12], [13], [15] and others). For our case the interval order structures would be relevant. However this notion has not been used in this paper. The main result of [1] show that the canonical orders like  $\llbracket x \rrbracket_x^{proc}$  and  $\llbracket x \rrbracket_x^{trace}$  uniquely represent appropriate interval order structures, so we do not have to use them explicitly.

An extension to general Place/Transition nets (as [14] did to some aspects of the model of [11], [15]) is a serious future research project.

## ACKNOWLEDGMENTS

Mohammed Alqarni acknowledges the full support by Ministry of Education in Saudi Arabia through The Saudi Arabian Cultural Bureau in Canada, and Ryszard Janicki acknowledges the partial support by NSERC grant of Canada.

## REFERENCES

- [1] Abraham, U., Ben-David, S., Magidor, M.: On global-time and inter-process communication. In *Semantics for Concurrency, Workshops in Computing*, pp. 311–323, Springer (1990)
- [2] Agerwala, T., Flynn, M.: Comments on capabilities, limitations and “correctness” of Petri nets, *Computer Architecture News* 4 (2), 81–86 (1973).
- [3] Alqarni, M., Janicki, R.: On Interval Process Semantics of Petri Nets with Inhibitor Arcs, Proc. of ICATPN’2015, *Lecture Notes in Computer Science*, 2015, to appear
- [4] Baldan, P., Busi, N., Corradini, A., and Pinna, G. M.: Domain and event structure semantics for Petri nets with read and inhibitor arcs, *Theoretical Computer Science* 323, 129–189 (2004)
- [5] Diekert V., Rozenberg, G. (eds.): *The Book of Traces*. World Scientific, Singapore (1995)
- [6] Fishburn, P. C.: Intransitive indifference with unequal indifference intervals. *Journal of Mathematical Psychology* 7, 144–149 (1970).
- [7] Goltz, U. and Reisig, W.: The non-sequential behaviour of Petri nets. *Information and Control*, 57(2):125–147, 1983.
- [8] Janicki, R.: Relational Structures Model of Concurrency. *Acta Informatica* 45, 279–320 (2008)
- [9] Janicki, R., Kleijn, J., Koutny, M.: Quotient Monoids and Concurrent Behaviours. In Martin-Vide, C. (ed.), *Scientific Applications of Language Methods*, pp. 311–385, Imperial College Press, London (2010)
- [10] Janicki, R., Koutny, M.: Structure of Concurrency. *Theoretical Computer Science* 112, 5–52 (1993)
- [11] Janicki, R., Koutny, M.: Semantics of Inhibitor Nets. *Information and Computation* 123(1), 1–16 (1995)
- [12] Janicki, R., Yin, X., Zubkova, N.: Modeling Interval Order Structures with Partially Commutative Monoids. In Proc. of CONCUR’2012, *Lecture Notes in Computer Science*, vol. 7454, pp. 425–439 (2012)
- [13] Janicki, R., Yin, X.: Modeling Concurrency with Interval Orders, *Information and Computation*, submitted
- [14] Juhás, G., Lorenz, R., Mauser, S.: Complete Process Semantics for Inhibitor Nets, Proc. of ICATPN’2007 *Lecture Notes in Computer Science*, vol. 4546, pp 184–203 (2007)
- [15] Kleijn, J., Koutny, M.: Process Semantics of General Inhibitor Nets. *Information and Computation* 190, 18–69 (2004)
- [16] Kleijn, J., Koutny, M.: Formal Languages and Concurrent Behaviour. *Studies in Computational Intelligence* 113, 125–182 (2008).
- [17] Mazurkiewicz, A.: Concurrent Program Schemes and Their Interpretation. TR DAIMI PB-78, Comp. Science Depart., Aarhus University (1977)
- [18] Mazurkiewicz, A.: Introduction to Trace Theory. In [5], pp. 3–42.
- [19] Montanari, U., Rossi, F.: Contextual nets, *Acta Informatica* 32 (6), 545–596 (1995).
- [20] Murata T.: Petri nets: Properties, analysis and applications, *Proc. of IEEE* 77 (4), 541–579 (1989)
- [21] Nielsen, M., Rozenberg, G., Thiagarajan, P. S.: Behavioural Notions for Elementary Net Systems. *Distributed Computing* 4, 45–57 (1990)
- [22] Ochmański, E., Recognizable Trace Languages, in [5], pp. 167–204.
- [23] Peterson, J. L.: *Petri nets theory and the modelling of systems*, Prentice-Hall, 1981
- [24] Rozenberg, G. Engelfriet, J.: Elementary Net Systems, in *Lectures on Petri Nets I: Basic models*, *Lecture Notes in Computer Science*, vol 1492, pp. 12–121 (1998)
- [25] Szpilrajn, E.: Sur l’extension de l’ordre partiel. *Fundam. Mathematicae* 16, 386–389 (1930)
- [26] Wiener, N.: A contribution to the theory of relative position, *Proc. of the Cambridge Philosophical Society* 17, 441–449 (1914)
- [27] Zuberek, W. M.: Timed Petri nets and preliminary performance evaluation. In Proc. of the 7-th Annual Symp. on Computer Architecture, pp. 89–96, La Baule, France (1980)

# Data Structures and Algorithms for Partitioning a Set into Sets of non-Descending Cardinality

Oshani Titti, Yijie Han

School of Computing and Engineering  
University of Missouri at Kansas City  
Kansas City, MO 64110

**Abstract-** *Data structures have been around since the structured programming era. Algorithms often associate with data structures. An algorithm is a sequence of instructions that accomplishes a task in a finite time period. The algorithm receives zero or more inputs, produces at least one output, consists of clear and unambiguous instructions, terminates after a finite number of steps, and is basic enough that a person can carry out the algorithm using a pencil and paper. Algorithms for dividing objects into bins have long been invented. However, dividing objects in summation format is not received due attention. In this paper, objects are divided into  $n$  bins in such a way that the next bin will contain more than or equal number of objects than the preceding bin.*

Keywords: *Data structure, algorithms, edge partition, integer partition, non-descending order partition.*

## 1. Introduction

Computer science is often difficult to define. This is probably due to the unfortunate use of the word “computer” in the name. As you are perhaps aware, computer science is not simply the study of computers. Although computers play an important supporting role as a tool in the discipline, they are just that—tools.

Computer science is the study of problems, problem-solving, and the solutions that come out of the problem-solving process. Given a problem, a computer scientist’s goal is to develop an **algorithm**, a step-by-step list of instructions for solving any instance of the problem that might arise. Algorithms are finite processes that if followed will solve the problem. Algorithms are solutions.

Computer science emphasizes two important topics: data structures and algorithms. Those topics are important because the choices you make for a program’s data structures and algorithms affect that program’s memory usage (for data structures) and CPU time (for algorithms that interact with those data structures).

This paper initiates a two-part series that explores data structures and algorithms. When choosing a data structure or algorithm, you sometimes discover an inverse relationship

between memory usage and CPU time: the less memory a data structure uses, the more CPU time associated algorithms need to process the data structure’s *data items*, which are primitive type values or objects, via references. Also, the more memory a data structure uses, the less CPU time associated algorithms need to process the data items—and faster algorithms result. This paper begins with a presentation of basic concepts and continues with a tour of the array data structure.

## 2. Approach

Consider ‘ $m$ ’ objects which have to be divided into ‘ $n$ ’ bins in such a way that the next bin should not contain less objects than the previous bin. Suppose first bin contains 1 object then all other bins should contain at least 1 object. Similarly if the fourth bin contains 3 objects then all other bins after fourth bin should contain a minimum of 3 objects. This is the method of dividing objects in the form of steps into the bins.

The number of ways of dividing  $m$  objects in  $n$  bins is represented as  $f(m, n)$ .

The number of objects that are being divided into the bins will remain same or in increasing order which is in the form of steps but never decreases.

## 3. Method

Let there be ‘ $m$ ’ distinct objects say  $1o, 2o, 3o, 4o, \dots, mo$  and ‘ $n$ ’ distinct bins say  $1b, 2b, 3b, \dots, nb$ . Let the function of dividing ‘ $m$ ’ objects into ‘ $n$ ’ bins be  $f(m, n)$ . The ‘ $m$ ’ objects should be divided in ‘ $n$ ’ bins by satisfying the following conditions:

1. Each bin can contain any number of objects.
2. The objects should be divided in such a way so that the successor bin should

always have more than or equal number of objects than its predecessor bin.

Example: Consider  $x$  contains  $y$  objects then  $(x+1)$  should contain  $\geq y$  objects.

*Example*

Consider an example of dividing 8 objects in 3 bins. The objects can be divided in the following ways:

| b1 | b2 | b3 |
|----|----|----|
| 0  | 0  | 8  |
| 0  | 1  | 7  |
| 0  | 2  | 6  |
| 0  | 3  | 5  |
| 0  | 4  | 4  |
| 1  | 1  | 6  |
| 1  | 2  | 5  |
| 1  | 3  | 4  |
| 2  | 2  | 4  |
| 2  | 3  | 3  |

In the above example if the bin 0 is filled with one object then all other should be filled with a minimum of 0 object and not less than that. Then the function for remaining objects is represented as  $f(8, 3)$  which there are 10 ways that can be filled in 3 bins.

Similarly if first bin is filled with 1 object then all other bins should be filled with a minimum of one object. The dividing process goes by following this condition till the last bin is filled with the last object.

#### 4. Generating an algorithm for dividing objects in non-descending order

Partitioning an integer  $n$  is to divide it into its constituent parts which are all positive integers. Algorithms for enumerating all the partitions of an integer or only the partitions with a restriction have long been invented [1,2]. Consider  $f(8,3)$  i.e dividing 8 objects into 3 bins. If bin1 contains 0 objects then the function for dividing the other objects is  $f(8,2)$  which means 8 objects should be divided in 2 bins. Similarly if bin1 contains 1 object then the function for dividing the other objects is  $f(5,2)$  which means 5 objects should be divided in 2 bins because if first bin is filled with one object then other two bins should also be filled with a minimum of one object. So the remaining objects to be filled are  $8-3=5$ . Similarly if bin1 and bin2 contains 2 object then the function for dividing the other objects is  $f(2,1)$  which means 1 objects should be divided in 1 bin because if first and second bean is filled with two objects then other beans should also be filled with a minimum of two objects. So the remaining objects to be filled are  $8-6=2$ .

Let  $f(m,n)$  be the number of ways of dividing  $m$  objects into  $n$  bins with non-descending cardinality. Function  $f(m,n)$  for dividing  $m$  objects into  $n$  bins in this particular format is shown

$$\begin{aligned}
 f(m,n) &= f(m, n-1) \quad // \text{first bin contains 0 objects} \\
 &+ f(m-n, n-1) // \text{first bin contains 1 object} \\
 &+ f(m-2n, n-1) // \text{first bin contains 2 objects} \\
 &+ f(m-3n, n-1) // \text{first bin contains 3 objects} \\
 &\dots
 \end{aligned}$$

$$\begin{aligned}
 &\dots \\
 &+ f(m- \lfloor m/n \rfloor *n, n-1) \quad // \text{first bin contains} \\
 &\lfloor m/n \rfloor \text{ objects.}
 \end{aligned}$$

#### 5. Partition Diagram

Algorithms for enumerating all the partitions of an integer or only the partitions with a restriction have been extensively studied [4], [5].

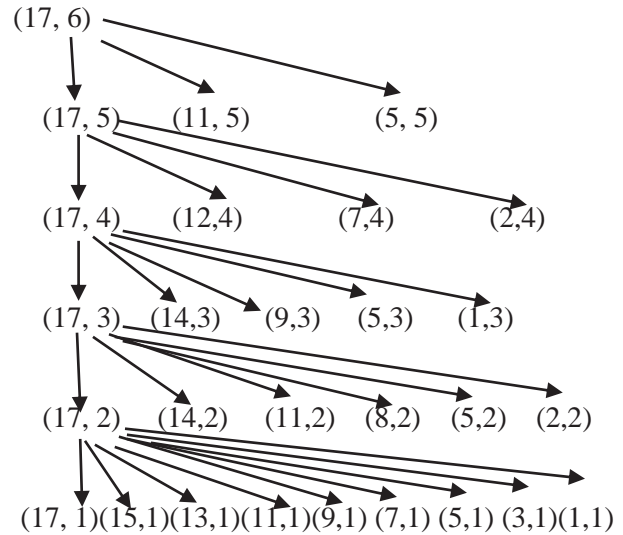


Fig.1: Patition diagram needed to divide 17 objects into 6 bins. The diagram represents a directed acyclic graph.

A data structure called partition diagram for storing all the partitions of an integer is proposed in [3]. In Merca [6], [7] improvements are proposed which, to date, are the most adequate data structures for generating integer partitions. We use the data structure proposed by Merca to present an efficient algorithm for generating ascending compositions of an integer  $n$  in  $m$  parts.

The partition diagram is a directed acyclic graph. Anode in the partition diagram is denoted by  $(m,n)$  where  $m$  is the number of objects and  $n$  denotes the number of bins. A node  $(m,n)$  that has no predecessor is called *anchored node (root node)* in a partition diagram. A node  $(m,n)$  which has no successor is called a *terminal node*. For example in the Fig. 1 the node  $(17,6)$  is an *anchored node* and also *internal node*, whereas node  $(2,1)$  is a *terminal node (leaf node)*.

Given a partition diagram, a path from an anchored node to terminal node defines a unique partition in which  $m$  objects are divided into  $n$  bins.

For example in Fig. 1 the path  $(17, 6) (17, 5) (7, 4) (5, 3) (2, 2)$  defines a partition.

If the number of objects in the first bin is ' $a$ ' then all bins should have at least ' $a$ ' objects. If we allocate ' $a$ ' objects to

every bin then we have  $(m-na)$  objects left to be distributed in  $(n-1)$  bins. So the process of dividing should continue till the value of  $m-n$  is greater than 0.

When we format the algorithm we can assume two situations

- *The first bin is empty*  
In this case  $m$  objects are to be distributed in  $(n-1)$  bins.
- *The first bin contains at least one object*  
In this case we allocate one object to every bin. Thus we have  $(m-n)$  objects left to be distributed in  $n$  bins.

```

Algorithm

function  $f(m, n)$ 
{
  if  $(m \geq n)$ 
  {
    if  $(n \equiv 1)$  return 1;
    else return  $(f(m, n-1) + f(m-n, n))$ ;
  }
  else return  $f(m, n-1)$ ;
}
    
```

*Lemma 1:*  $f(m, n) \leq xf(m-1, n) \leq nf(m-1, n)$ , where  $x$  is the number of steps explained below.

Consider  $m$  objects to be placed into  $n$  bins. Let  $x$  be the number of steps that are formed while arranging the objects.

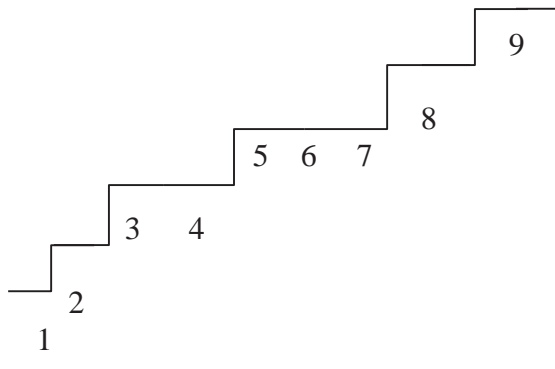


Fig. 2: Different ways of arranging the objects  
If the successor bin contains more objects than the preceding bin then the objects are arranged in the form of steps in between the bins. For example in Fig. 2 shows a configuration in which bin 2 contains more number of objects than bin 1 so they are arranged as a step. Similarly bin 2 and bin 3, bin 4 and bin 5, bin 7 and bin 8, bin 8 and bin 9. If the successor bin contains the same number of objects as in the preceding bin then the arrangement is not formed as step. It remains at same level as they contain same

number of objects. For example bin 3 and bin 4 contains same number of objects so they remain at same level. Similarly bin 5, bin 6 and bin 7 contain same number of objects.

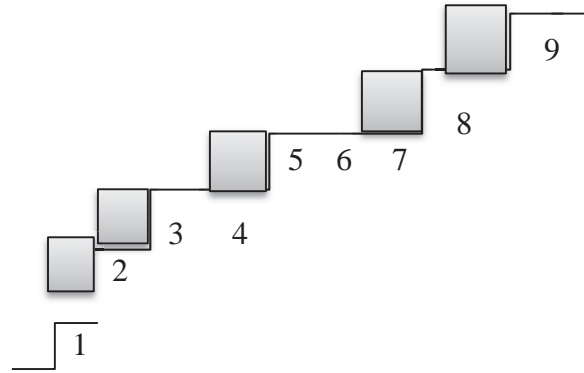


Fig.3: Different ways of adding an object in the above arrangement

If we want to add an object in the configuration shown in Fig.2, the added object should not affect the configuration i.e. the object is to be added in non-decreasing cardinality. As shown in Fig.3, the added object can only be placed in the shaded portion. Therefore we can say that the number of arranging the objects in the predecessor bin is always less than or equal to the number of way of arranging the objects in the successor bin in the form of steps. When we have a configuration for  $f(m-1, n)$  then we can generate at most  $xf(m-1, n)$  configurations when we add an object, where 'x' is the number of steps in the configuration for  $f(m-1, n)$ . Hence  $f(m, n) \leq xf(m-1, n)$ . Because  $x \leq n$  thus  $f(m, n) \leq xf(m-1, n) \leq nf(m-1, n)$ . □

*Lemma 2:*  $f(m, n) \leq \sqrt{2m} f(m-1, n)$

From the above lemma we know that  $f(m, n) \leq xf(m-1, n)$  where  $x$  is the number of steps.



Fig. 3: Arrangement of objects to achieve maximum steps

Maximum number of steps can be obtained by placing the objects in the following way. Let there be many number of objects with many number of bins. Then the objects are placed in increasing way to obtain the maximum steps in the following way

- 0 objects in bin 1
- 1 object in bin 2
- 2 objects in bin 3
- 3 objects in bin 4
- .
- .
- .
- t objects in bin t+1.

Let m be the total number of objects, then

$$m = [t(t+1)] / 2$$

By solving this we get  $t = \sqrt{2m}$

The maximum number of steps is  $\sqrt{2m}$

$$\text{Therefore } f(m, n) \leq \sqrt{2m} f(m-1, n) \quad \square$$

Algorithms which efficiently built these kind of integer partition combinations have long been studied, a survey can be found in Knuth [8]. Although the space and time needed to store either the partition diagram or the set of kernel strings is quadratic [9], our approach creates the most efficient data structure with space and time complexity  $O(mn)$ . The space and time complexity is low enough to make possible for storing all the partitions of an integer up to several ten thousands. In [3]  $O(n^2)$  storage is used for storing the partitions. Our algorithm uses less space when m is smaller than n. The implication of this result is that, in practical applications, we can efficiently recover subsets for a given path graph as kernel strings are the base to generate combined strings.

### 6. Second Approach

Consider a case in which  $x_0$  contains 0 objects,  $x_1$  contains 1 object,  $x_2$  contains 2 objects,  $x_3$  contains 3 objects..... $x_a$  contains a objects.

Let there be 'm' objects. Then we can say  $x_0 0 + x_1 1 + x_2 2 + x_3 3 + x_4 4 + \dots + x_a a = m$

Example: Consider an example of dividing 4 objects into 3 bins such that  $x_1 1 + x_2 2 + x_3 3 = m$

The only case we have is

$$x_1 = 1, x_2 = 0, x_3 = 1$$

This method of partitioning the objects contains less number of ways than the method of partitioning objects in traditional method i.e  $x_0 + x_1 + x_2 + x_3 + x_4 + \dots + x_a = m$ .

Now consider the same example of dividing 4 objects into 3 bins such that  $x_1 + x_2 + x_3 = m$

The possible cases are follows

- $x_1 = 0, x_2 = 0, x_3 = 4$
- $x_1 = 0, x_2 = 1, x_3 = 3$
- $x_1 = 0, x_2 = 2, x_3 = 2$
- $x_1 = 0, x_2 = 3, x_3 = 1$
- $x_1 = 0, x_2 = 4, x_3 = 0$
- $x_1 = 1, x_2 = 1, x_3 = 2$
- $x_1 = 1, x_2 = 2, x_3 = 1$
- $x_1 = 1, x_2 = 0, x_3 = 3$
- $x_1 = 1, x_2 = 3, x_3 = 0$
- $x_1 = 2, x_2 = 1, x_3 = 1$
- $x_1 = 2, x_2 = 0, x_3 = 2$
- $x_1 = 2, x_2 = 2, x_3 = 0$
- $x_1 = 3, x_2 = 0, x_3 = 1$
- $x_1 = 3, x_2 = 1, x_3 = 0$
- $x_1 = 4, x_2 = 0, x_3 = 0$

### 7. Generating an algorithm for partitioning objects

Initially consider there are m+n-1 positions out of which choose n-1 positions then you will be left with m objects to be partitioned into n bins.

Example:

Consider the following 4 objects to be partitioned into 3 bins.

1. Initially consider m+n-1 positions i.e. 4+3-1=6



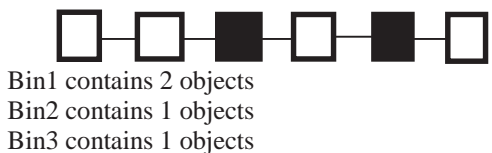
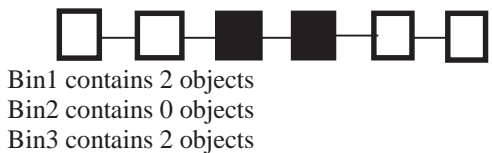
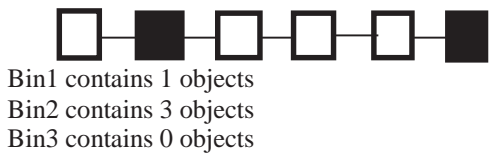
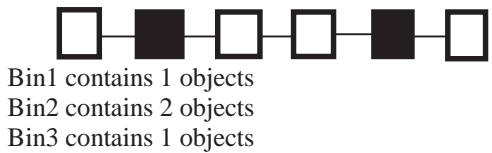
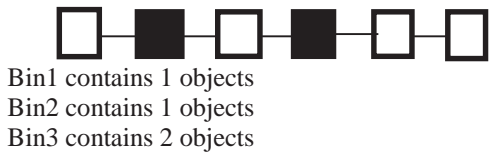
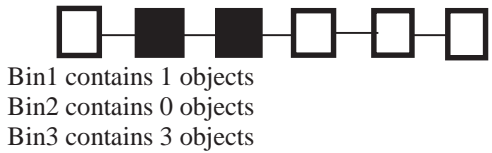
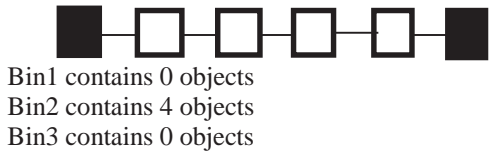
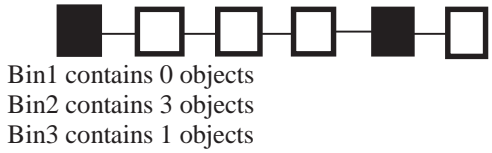
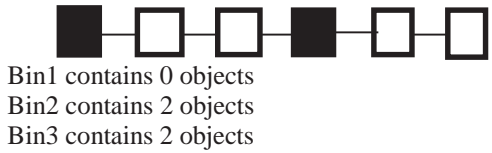
2. Choose n-1 positions i.e. 3-1=2. The two positions can be taken in the following ways.



- Bin1 contains 0 objects
- Bin2 contains 0 objects
- Bin3 contains 4 objects



- Bin1 contains 0 objects
- Bin2 contains 1 objects
- Bin3 contains 3 objects



Bin1 contains 2 objects  
Bin2 contains 2 objects  
Bin3 contains 0 objects



Bin1 contains 3 objects  
Bin2 contains 1 objects  
Bin3 contains 0 objects



Bin1 contains 3 objects  
Bin2 contains 0 objects  
Bin3 contains 1 objects



Bin1 contains 4 objects  
Bin2 contains 0 objects  
Bin3 contains 0 objects

Partitioning  $m$  objects into  $a$  bins using the method  $x_0 \cdot 0 + x_1 \cdot 1 + x_2 \cdot 2 + x_3 \cdot 3 + \dots + x_a \cdot a = m$  has definitely less number of steps than partitioning the objects in  $x_1 + x_2 + x_3 + x_4 + \dots + x_a = m$ . The space and time complexity for creating a linear structure or a partition tree is definitely less while compared to the traditional method. The ways of partitioning with  $x_0 \cdot 0 + x_1 \cdot 1 + x_2 \cdot 2 + x_3 \cdot 3 + \dots + x_a \cdot a = m$  is always a subset of ways of partitioning objects using  $x_1 + x_2 + x_3 + x_4 + \dots + x_a = m$

Now we consider partition of ' $m$ ' objects into bins such that a bin can contain atmost ' $a$ ' objects. The formula for this is  $1x_1 + 2x_2 + 3x_3 + \dots + ax_a = m$  which is less than the number of ways  $m$  can be partitioned into  $a$  bins by

$$x_1 + x_2 + x_3 + \dots + x_a = m \text{ which is } \leq \binom{m+a-1}{a-1} \quad (1)$$

Then the equation will be

$$x_{a+1}(a+1) + x_{a+2}(a+2) + \dots + x_b b = m$$

Now consider the ways  $m$  objects can be partitioned into bins in such a way that each bin contains at least  $a$  objects. The number of bins is now restricted by  $m/(a+1)$ .

We first choose  $m/(a+1)$  positions among  $m$  positions with

$$\binom{m}{m/(a+1)} \text{ ways} \quad (2)$$

and then partition  $m$  objects in non-descending as

$$x_{a_1} a_1 + x_{a_2} a_2 + \dots + x_{a_{m/(a+1)}} a_{m/(a+1)} = m$$

where  $a_1 < a_2 < a_3 < \dots$ . Compare this case with the method of partitioning  $m$  objects into  $m/(a+1)$  bins *i.e.*

$$x_1 + x_2 + \dots + x_{m/(a+1)} = m$$

The number of ways of dividing  $m$  objects using the method  $x_1 + x_2 + \dots + x_{m/a+1} = m$  is

$$\binom{m + (m/(a+1)) - 1}{m/(a+1) - 1} \quad (3)$$

The number of ways of dividing  $m$  objects using the method  $x_{a_1} a_1 + x_{a_2} a_2 + \dots + x_{a_{m/(a+1)}} a_{m/(a+1)} = m$  is  $\leq$

$$\binom{m + (m/(a+1)) - 1}{m/(a+1) - 1}$$

So the total number of ways of partitioning ' $m$ ' objects into ' $m$ ' bins is less than the product of (1), (2) and (3), that is

$$\leq \binom{m+a-1}{a-1} \cdot \binom{m}{m/(a+1)} \cdot \binom{m + (m/(a+1)) - 1}{m/(a+1) - 1}$$

$$\approx (m/a)^a a^{m/a} a^{m/a} \leq \max[(m/a)^{3a}, a^{3m/a}]$$

Therefore we let

$$[m/a]^a = a^{m/a}$$

$$a \log(m/a) = (m/a) \log a$$

$$[a^2(\log m - \log a)] / \log a = m$$

$$a^2 = m \log a / \log m$$

$$a = \sqrt{m \log a / \log m}$$

$$\approx \sqrt{m}$$

This gives about  $m^{\sqrt{m}}$  ways, while the number of ways of partitioning objects using the formula  $x_1 + x_2 + x_3 + x_4 + \dots + x_m = m$

$$\text{gives } \binom{m+m-1}{m-1} \approx 2^{2m}$$

### 8. Conclusion and Future work

In this paper we have studied about partitioning a set into non-descending cardinality. The space and time complexity for creating a linear structure or a partition tree is proportional to the number of partitions whereas the complexity for creating a partition diagram is only  $O(mn)$ . This complexity allows us to create a partition diagram that can store all the partitions of an integer up to several ten thousands.

### 9. Reference

[1] D. Stanton and D. White. *Constructive Combinatorics*. Springer-Verlag, Berlin, 1986.  
 [2] C. L. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill College, 1968.  
 [3] R.-B. Lin, "Efficient data structure for storing the

partitions of integers," The 22nd Workshop on Combinatorics and Computation Theory, pp. 349–354, 2005.  
 [4] D. Stanton and D. White, "Constructive combinatorics," SpringerVerlang, Berling, 1986.  
 [5] C. L. Liu, "Introduction to combinatorial mathematics," MacGraw-Hill College, 1986.  
 [6] M. Merca, "Binary diagrams for storing ascending compositions," The Computer Journal Advance Access, 2012.  
 [7] —, "Fast algorithm for generating ascending compositions," Journal of Mathematical Modelling and Algorithms, vol. 11, pp. 89–104, 2012. [Online]. Available: <http://dx.doi.org/10.1007/s10852-011-9168-y>  
 [8] D. E. Knuth, The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1. Addison Wesley, 2011.  
 [9] J. Raymundo Marcial-Romero, J. A. Hernández Vianney Muñoz-Jiménez and Héctor A. Montes-Venegas Generating edge covers of path graphs. In Pecoceedings of 2013 WORLDCOMP.

# Multiprocessor MMIO Tracing via Memory Protection and a Shadow Page Table

Myoungjae Kim<sup>1</sup>, Hyunmin Yoon<sup>2</sup>, Minkwan Choi<sup>1</sup>, Shakaiba Majeed<sup>1</sup>, and Minsoo Ryu<sup>1\*</sup>

<sup>1</sup>Department of Computer Science and Engineering, Hanyang University, Seoul, Korea

<sup>2</sup>Department of Electronics Computer Engineering, Hanyang University, Seoul, Korea  
{mjkim, hmyoon, mkchoi, shakaiba}@rtcc.hanyang.ac.kr, msryu@hanyang.ac.kr

**Abstract** – *Memory-mapped I/O (MMIO) tracing provides an effective means for analyzing and debugging I/O related functions since it allows us to observe and track the interplay between processors and I/O devices [1]. However, existing MMIO tracing techniques have a serious drawback in multicore systems. Current MMIO techniques commonly use a memory protection mechanism to detect access to an MMIO address area under consideration. Unfortunately, this approach may miss some I/O events and even lead to a data race condition due to inappropriate management of concurrent accesses to the MMIO address area. In this paper, we describe a novel MMIO tracing approach introducing the notion of shadow page table. We use a shadow page table to allow only one processor to have access to a MMIO address area while forbidding other processors' access to the same MMIO address area. We show how the shadow page table approach can be efficiently implemented on a multiprocessor platform with dual core ARM Cortex A15 CPU.*

**Keywords:** Memory Mapped I/O (MMIO) Trace, Memory Protection, Page Fault, Shadow Page Table.

## 1 Introduction

Memory-mapped I/O (MMIO) tracing provides an effective means for analyzing and debugging I/O related functions since it allows us to observe and track the interplay between processors and I/O devices. For example, to analyze and debug failures in device drivers, developers must be able to find out what data is sent to or received from the device. MMIO tracing can collect detailed information about I/O operations conducted between a processor and I/O devices, thus enabling us to track down the source of failures.

However, existing MMIO tracing techniques have a serious drawback in multicore systems. Current MMIO tracing techniques commonly use a system-wide address translation table, i.e. page table in processors with paging support, to set the MMIO address area under consideration as invalid and rely on memory access exceptions to detect any processor's access to the protected MMIO address area. When an exception is generated by a read/write instruction, a specially designed exception handler collects information

about the I/O access, enables access permission for the MMIO address area, re-executes the faulting memory access instruction, and sets the access permission back to invalid. Unfortunately, in multicore hardware, this may lead to missing some I/O events and even a data race condition since other processors can make writes simultaneously to the same address area during the time interval where the access to MMIO address area is enabled.

In this paper, we present a novel MMIO tracing method introducing the notion of shadow page table. When a page fault occurs on a certain processor, we replace the page table seen by the exception handling processor with a shadow page table, while leaving other processors referencing the original page table. The shadow page allows only the exception handling processor to access the MMIO address area, but other processors' access to the MMIO area is prohibited through the original page table. Therefore, this approach allows us to avoid the problem of missing I/O events and race conditions. We describe how the shadow page table approach can be efficiently implemented on a multiprocessor platform with dual core ARM Cortex A15 CPU.

This paper is organized as follows. Section 2 describes existing MMIO tracing techniques. Section 3 presents our shadow page table approach and Section 4 concludes this paper.

## 2 Background of MMIO Tracing

### 2.1 Memory-mapped I/O (MMIO)

MMIO requires a section of memory to allow a processor to communicate with I/O controllers. A processor with MMIO support reserves some part of its address space for a special I/O address range where I/O controllers' registers are mapped to specific addresses in the designated I/O address range. Programs can access I/O registers through memory access instructions such as load and store, which is no different from read/write access to normal memory addresses [3].

MMIO tracing can be efficiently implemented using a page table. A page table contains the mapping between virtual addresses and physical addresses and some additional information associated with each page table entry. One important piece of information is the access permission for



each page. By manipulating the access permission for each MMIO page, we can allow or prohibit the processor's access to specific MMIO pages. MMIO tracing initially disables access permission for MMIO pages using the page table. Whenever a processor attempts to access a protected MMIO page, a page fault exception occurs. A special page fault handler then collects information about the I/O access, enables the access permission for the MMIO page, re-executes the faulting memory access instruction, and re-disables the access permission.

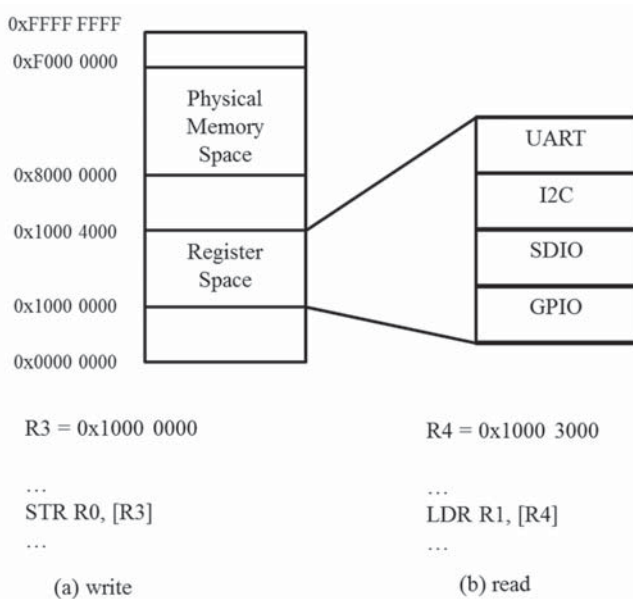


Figure 1. Address space of a Processor using MMIO.

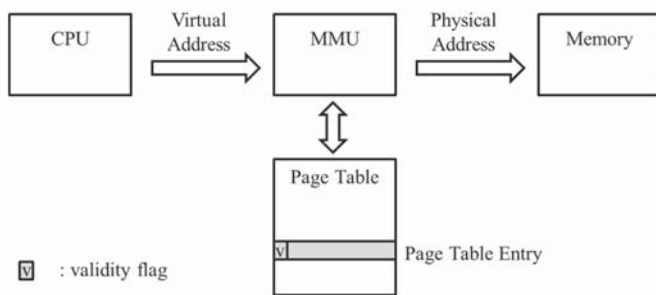


Figure 2. Paging and translation scheme.

## 2.2 MMIO Tracing in Linux

The Linux MMIO tracing tool uses a validity attribute associated with each page table entry to force page fault to occur when a processor accesses a memory mapped I/O region even if the region exists in a valid page [5]. The tool

records the MMIO accesses in the following way: First, the MMIO pages are marked as invalid. When a fault occurs due to an access to these pages, the page fault handler emulates the faulting instruction by changing the attribute of the page as valid and starts logging the events. After the emulation and logging the page fault handler again marks the page as invalid. Finally, the interrupted kernel code takes control again and executes the next instruction to the faulting instruction.

While the page fault handler is emulating the faulting instruction, the other processors can freely access the page containing the data which the faulting instruction wanted to access because that page is marked valid during this interval. In such situation, other processor's access does not create a page fault which leads to event missing without notice.

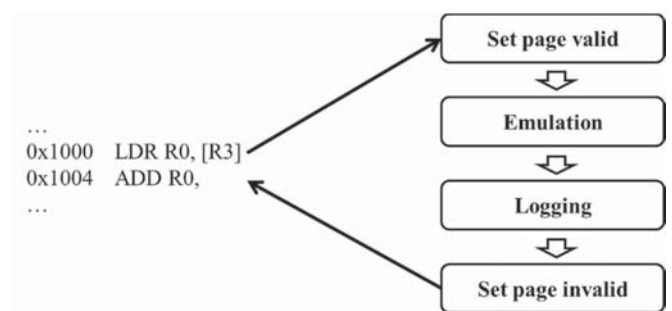


Figure 3. Tracing control flow.

## 3 MMIO Tracing with a Shadow Page Table

As mentioned earlier, existing MMIO tracing techniques based on a memory protection mechanism may fail to capture some concurrent I/O events on multiprocessors. The problem is that other processors can make references to the same MMIO address area during the interval the memory access is allowed. Those accesses cannot be detected as they do not trigger page fault exceptions and may even lead to data race conditions.

A plausible solution is freezing other processors during the page fault handling. When a page fault happens, we may stop other processors' execution by sending a special inter-processor interrupt (IPI) to other processors. This would prevent other processors from accessing the MMIO address area. However, sending and receiving IPIs also requires access to the interrupt controller's MMIO addresses, which would entail the same problem.

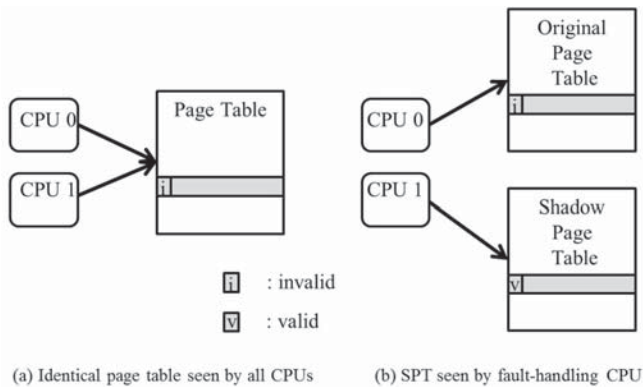


Figure 4. Shadow Page Table.

In order to address the above problem, we propose the use of a shadow page table (SPT). When a page fault occurs, a shadow page table replaces the kernel’s original page table used by the fault handling processor. The use of shadow page table allows us to enable the access permission of the fault handling processor while other processors’ memory access is prohibited by the original kernel’s page table. Therefore, this approach can overcome the problem of missing I/O events and race conditions.

The shadow page table can be efficiently implemented in many operating systems that support paging-based memory management. We replicate the original kernel’s page table and modify the access rights to the MMIO address areas in the replicated shadow page table to enable access permission. When a page fault occurs, we change the page table base register of the processor so that it can refer to the shadow page table during the page fault handling. Since other processors still refer to the original page table, they are not allowed to make access to the MMIO address areas. Once logging MMIO I/O access information is done, we change the page table base register to point to the original page table. Afterwards, all the processors use the original page table. There is a possibility that two more processors try to write access on a same MMIO address almost at the same time. It also leads to data race condition as two processors re-execute the faulting memory access instructions. To prevent this problem, we need to protect fault handling as a critical section with a synchronization method such as spin lock.

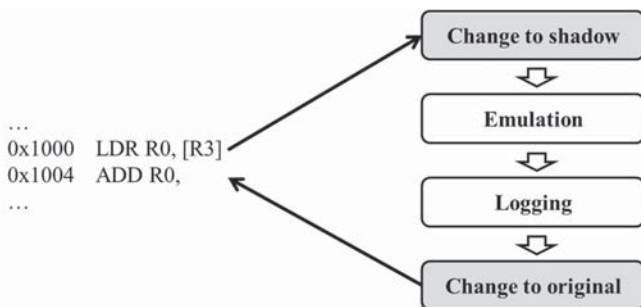


Figure 5. Tracing control flow with SPT.

## 4 Conclusion

In this paper, we have presented a novel MMIO tracing method introducing the notion of shadow page table. Letting a processor refer to shadow page table while it conducts MMIO tracing, we can solve a problem of missing another MMIO event by other processors as well as data race condition under multiprocessor platform.

## 5 Acknowledgment

This work was supported partly by Seoul Creative Human Development Program (HM120006), and partly by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (NRF-2011-0015997), and partly the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the C-ITRC(Convergence Information Technology Research Center) (IITP-2015-H8601-15-1005) supervised by the IITP(Institute for Information & communications Technology Promotion).

## 6 References

- [1] Wikipedia, “Memory-mapped I/O,” [Online]. Available: [http://en.wikipedia.org/wiki/Memory-mapped\\_I/O](http://en.wikipedia.org/wiki/Memory-mapped_I/O)
- [2] A. Kadav and M. M. Swift, "Understanding modern device drivers," *ACM SIGARCH Computer Architecture News*, vol. 40, pp. 87-98, 2012.
- [3] D. P. Bovet and M. Cesati, *Understanding the Linux kernel*: " O'Reilly Media, Inc.", 2005
- [4] Wikipedia, “Virtual memory,” [Online]. Available: [http://en.wikipedia.org/wiki/Virtual\\_memory](http://en.wikipedia.org/wiki/Virtual_memory)
- [5] LWN, “Tracing memory-mapped I/O operations,” [Online]. Available: <https://lwn.net/Articles/270939/>

## **SESSION**

# **GRAPH AND NETWORK BASED ALGORITHMS + FORMAL METHODS AND APPLICATIONS + QUANTUM COMPUTING AND RELATED ISSUES**

**Chair(s)**

**TBA**



# Beyond the Solution to 2-CSC: $O(n^2)$ Algorithm to Discover Corresponding Chain Sub-graph Covers

Wei-Da Hao<sup>1</sup> and Lin-Yu Tseng<sup>2</sup>

<sup>1</sup> Electrical Engineering and Computer Science, Texas A&M University – Kingsville, Kingsville, TX 78363, USA

<sup>2</sup> Computer Science and Communication Engineering, Providence University, Taichung 43301, ROC

**Abstract**— *The algorithm proposed in this paper accepts as input an arbitrary bipartite  $G$  and responds with corresponding chain sub-graph covers, if  $G$  is recognized as coverable by two or less chain sub-graphs and responds “no” otherwise. The component ideas of a solution to the 2-CSC problem by Tze-Heng Ma and Jeremy Spinrad in 1994 are compiled into applicable steps and assembled to construct the proposed algorithm. The time complexity is  $O(n^2)$ , where  $n$  is the number of vertices of  $G$ . Related problems that can find this research useful are discussed in the conclusion.*

**Keywords**—2-CSC; chain; sub-graph; bipartite; recognition

## 1. Introduction

In this paper, we design an effective algorithm, which has time complexity  $O(n^2)$ , for obtaining more informative outcome from a previous solution to 2-chain subgraph cover (2-CSC) problem [1]. The input to the 2-CSC problem is any bipartite graph  $G$ , and the expected output is a “yes” or “no” answer that responds to the question if  $G$  can be covered by two or one chain sub-graph.

In many occasions, a “yes” output generated from the 2-CSC problem is not informative enough, and we usually want to know the corresponding chain sub-graph covers as well. While most of the supporting properties and theorems enabling the discovering of the desired chain sub-graph covers are implied in the solution to the 2-CSC problem, an applicable algorithm has yet been designed.

Thus, we look into the solution to the 2-CSC problem in detail. Inspect supporting theorems and properties to each step for useful information to develop the algorithm discovering chain sub-graph covers. At the beginning of the solution to 2-CSC problem, a partial order  $P$  is derived from the input bipartite graph  $G$ . Then, the algorithm examining if the dimension of a partial order is two is applied to  $P$ . Since, by theorem, the dimension of  $P$  derived from  $G$  is the same as the minimum number of chain sub-graph covering of  $G$  [1], we can get the “yes” or “no” answer to a 2-CSC problem from the outcome of previous algorithm. This paper intends to go beyond a “yes” answer and substantializes the useful information implied in the above computing process for the algorithmic steps to discover the corresponding chain sub-graph covers.

## 2. Definitions

### 1. Graph

Graph  $G$  is composed of set of vertices  $V$  and set of edges  $E$ , expressed in  $G = (V, E)$ . Assume  $u, v \in V$ . For directed graph, edge  $(u, v) \in E$  represents the connection from  $u$  to  $v$ , which is a directed edge. For undirected graph, edge  $uv \in E$  or  $vu \in E$  represents the connection between  $u$  and  $v$ , which is an undirected edge.

### 2. Bipartite graph

Let  $G = (V, E)$  represent a bipartite graph. What's different from general graph is that  $V$  can be partitioned to two sets  $X$  and  $Y$ , and edge only exists between  $X$  and  $Y$ . To distinguish from general graph, we use  $B = (X, Y, E)$  to represent a bipartite graph, and every edge  $uv \in E$  (for directed graph,  $(u, v) \in E$ ), either  $u \in X$  and  $v \in Y$ , or  $v \in X$  and  $u \in Y$ .

### 3. Chain graph

Chain graph is an undirected bipartite graph that its edge set doesn't contain  $2K_2$ , which is two undirected edges  $uv$  and  $wx$ , that either  $uw \notin E$  and  $vx \notin E$ , or  $ux \notin E$  and  $vw \notin E$ .

### 4. K-CSC and 2-CSC problem

K-CSC is abbreviation of “K-Chain Sub-graph Cover”. K-CSC problem is a problem about the question if an input bipartite graph can be covered by K chain sub-graphs. It has been proved that when  $K \geq 3$ , K-CSC problem is NP-complete. When  $K = 2$ , K-CSC problem is a 2-CSC problem.

### 5. $ch(G)$

$ch(G)$  is the smallest number K such that bipartite graph  $G$  is K-chain sub-graph coverable.

### 6. Partial order [2]

Partial order  $P$  defines the dominant relationship  $R$  between some pairs of elements in a group of elements  $X$ , expressed by  $P = (X, R)$ .  $R$  is transitive and non-reflexive binary relationship on  $X$ . For  $x, y \in X$ ,  $(x, y) \in R$  indicates  $x$  dominates  $y$ , expressed by  $x < y$ . If either  $x < y$  or  $x > y$ ,  $x$  and  $y$  are comparable. If neither  $x < y$  nor  $x > y$ ,  $x$  and  $y$  are non-comparable. For  $x, y, z \in X$ , if  $(x, y) \in R$  and  $(y, z) \in R$ , it can be concluded, through the fact that  $R$  is transitive, that  $x < z$ , but  $(x, z) \in R$  may not be true.

### 7. Extension of a partial order

For a partial order  $P = (X, R)$ , its extension is another partial order  $P' = (X, R')$ , such that  $R \subseteq R'$ . If every pair of elements of  $X$  is comparable in  $P'$ ,  $P'$  is a linear extension of  $P$ .

### 8. Linear order(total order)

If a partial order has no incomparable pair, it is called a linear order.

### 9. Dimension of Partial Order

Dushnik and Miller defined the dimension of a partial order  $P$  [2],  $\dim(P)$ , as "the minimum number of total orders whose intersection defines the partial order".

### 10. 2-Dimensional partial order problem

This is a problem to determine if a given partial order whose dimension is two or less than two.

### 11. Modular Decomposition of a DAG (Directed Acyclic Graph)

DAG  $G = (V, E)$  is used to represent a partial order  $P = (V, R)$ , such that for  $x, y \in V$ :  $(x, y) \in E$  if and only if  $(x, y) \in R$ .  $x, y$  are related in DAG  $G$ , if  $(x, y) \in E$  or  $(y, x) \in E$ . Otherwise,  $x, y$  are not related in DAG  $G$ . Since  $R$  is transitive, DAG  $G$  is a transitive graph.

1) Representation of a DAG: The linear orders on  $V$  whose intersection is  $P$  construct the representation of the DAG  $G$ .

2) Listing and Non-separating listing: Each linear order of the representation for a DAG  $G$  is called a listing. Between two comparable elements  $u, v$  of  $P$  in a listing, if there is no element that is not comparable to both  $u$  and  $v$ , the listing is a non-separating listing.

3) Two dimensional partial order of a DAG  $G = (V, E)$ : Each linear order of the representation for a DAG  $G$  is called a listing. Between two comparable elements  $u, v$  of  $P$  in a listing, if there is no element that is not comparable to both  $u$  and  $v$ , the listing is a non-separating listing.

4) Module  $M$ : Module  $M$  is a subset of  $V$  with the property that for  $v \in V - M$ , either  $v$  is related to every vertex in  $V$  or not at all.

5)  $M_C$ :  $M_C$  is the undirected graph that has  $M$  as vertex set, and for  $u, v \in M$ ,  $uv$  is an edge in  $M_C$ , if  $u, v$  are related in DAG  $G$ .

6)  $M_{CC}$ :  $M_{CC}$ , of is the undirected graph that has  $M$  as vertex set, and for  $u, v \in M$ ,  $uv$  is an edge in  $M_{CC}$ , if  $u, v$  are not related in DAG  $G$ .  $M_C$  and  $M_{CC}$  are complementary to each other.

7) Maximal submodule: A module  $M$  is said to be a maximal submodule of another module  $N$ , if  $M \subset N$  and no proper submodule of  $N$  contains  $M$ .

8) Parallel module: If  $M_C$  is not connected,  $M$  is a parallel module. The vertices of a parallel module can be partitioned into two subsets, so that none of the vertex in one subset is related to any vertex of the other.

9) Series module: If  $M_{CC}$  is not connected,  $M$  is a series module. The vertices of a series module can be partitioned into two subsets, so that any of the vertexes in one subset is related to any vertex of the other.

10) Neighborhood module: If both  $M_C$  and  $M_{CC}$  are connected,  $M$  is a neighborhood module.

## 3. Properties, algorithms and theorems

Given a bipartite graph  $G = (X, Y, E)$ ,  $X = \{x_1, x_2, \dots, x_l\}$ ,  $Y = \{y_1, y_2, \dots, y_m\}$ , a partial order  $P = (X \cup Y, R)$  can be generated from  $G$  through the following steps [1] in sequence:

Algorithm 1 Convert Bipartite Graph to Partial Order

INPUT:

$G = (X, Y, E)$ ,  $X = \{x_1, x_2, \dots, x_l\}$ ,  $Y = \{y_1, y_2, \dots, y_m\}$

OUTPUT:

A partial order  $P = (X \cup Y, R)$

BEGIN

1. Compute neighborhood of  $v$ ,  $N(v)$ , where  $v \in X \cup Y$ .
2. For those vertices having the same neighborhood, keep only one for the rest of steps.
3. If  $N(x_i) \supset N(x_j)$ , add  $(x_i, x_j) \in R$ .
4. If  $N(y_i) \supset N(y_j)$ , add  $(y_j, y_i) \in R$ .
5. If  $x_i y_j \notin E$ , add  $(y_j, x_i)$  to  $R$ .
6. For  $v \in X \cup Y$ , let  $U_X(v) = \{u | u \in X \text{ and } (v, u) \in R\}$ . If  $U_X(y_j) \subseteq U_X(x_i)$ , add  $(x_i, y_j) \in R$ .

END

And, an undirected bipartite graph  $B(P) = (X \cup Y, X' \cup Y', E_{B(P)})$ , where  $X' = \{x' | x \in X\}$  and  $Y' = \{y' | y \in Y\}$ , can be generated from  $P = (X \cup Y, R)$  above through the following steps [1]:

Algorithm 2 Generate  $B(P)$  from  $P$

INPUT:  $P = (X \cup Y, R)$

OUTPUT:  $B(P) = (X \cup Y, X' \cup Y', E_{B(P)})$

BEGIN

1. For  $a, b \in X \cup Y$  and  $a \neq b$ , if  $a < b$  in  $P$ , let  $ab' \in E_{B(P)}$ .
2. For  $a, b \in X \cup Y$  and  $a \neq b$ , if  $a \sim b$  in  $P$ , let  $ab' \in E_{B(P)}$ .

END

Lemma 1 [1]

$ch(G) = \dim(P) = ch(B(P))$

Lemma 2 [1]

There is one-to-one mapping between the linear order of  $P$  and maximal chain sub-graph of  $B(P)$ .

By lemma 1, if  $\dim(P) = 2$ ,  $ch(G) = 2$ . By lemma 2, a maximal chain sub-graph of  $B(P) = (X \cup Y, X' \cup Y', E_{B(P)})$  can be obtained from a linear order of  $P$ . We realize this fact by looking into the supporting logic to lemma 2 and design the following steps to compute the desired maximal chain sub-graph.

Algorithm 3 Map Linear Order to Maximal Bipartite Graph  
BEGIN

1. Given a linear order  $L$  on  $X \cup Y$  of  $P = (X \cup Y, R)$
  2. Define the corresponding maximal bipartite graph  $H = (X \cup Y, X' \cup Y', E)$ , where  $ab' \in E$ , if  $a < b$  in  $L$ .
- END

Lemma 3 [1]

$G = (X, Y, E)$  is a sub-graph of  $B(P) = (X \cup Y, X' \cup Y', E_{B(P)})$  induced by  $X$  and  $X'$ .

Lemma 4

There is one-to-one mapping between the maximal chain sub-graph of  $G$  and the maximal chain sub-graph of  $B(P)$ .

Proof:

Assume  $ch(B(P)) = k$ . There are  $k$  distinct maximal chain sub-graphs,  $MaxChainSubB_i, i = 1 \dots k$ , that can cover the edges and vertices of  $B(P)$ . By lemma 3,  $G$  is a sub-graph of  $B(P)$  induced by  $X$  and  $X'$ , so  $G$  can be covered by the sub-graphs of  $MaxChainSubB_i, i = 1 \dots k$ , induced by  $X$  and  $X'$ , too. And, all of the induced maximal chain sub-graphs are different, otherwise,  $ch(G) < k$ , which is a contradiction to lemma 1. Q.E.D.

By lemma 4, the maximal chain sub-graphs of  $G$  can be obtained from the maximal chain sub-graphs of  $B(P)$ . We realize this fact by looking into the supporting logic to lemma 4 and design the following steps to compute the desired maximal chain sub-graphs for  $G$ .

Algorithm 4 Compute Maximal Chain Sub-graph  
BEGIN

1. Given  $B(P)$  with  $ch(B(P)) = k$ , assume the maximal chain sub-graphs that cover  $B(P)$  are  $MaxChainSubB_i, i = 1 \dots k$ .
  2. The maximal chain sub-graphs  $MaxChainSubG_i, i = 1 \dots k$ , that cover  $G$  can be obtained by generating sub-graph induced by  $X$  and  $X'$  on each of  $MaxChainSubB_i, i = 1 \dots k$ .
- END

Lemma 5 [3]

$O(n^2)$  algorithm exists for checking if  $\dim(P) = 2$  is true, and when  $\dim(P) = 2$  producing two linear orders whose intersection is  $P$ .

The contribution in [3] is an algorithm for recognizing two dimensional partial orders. The input is a partial order  $P$ . The output is "yes" and two linear orders with intersection as  $P$ , if  $P$  is two dimensional partial order, or "no", if  $P$  is not a two dimensional partial order. The steps of the recognition algorithm are listed in the appendix.

Theorem

Let  $G$  be a bipartite graph and  $P$  is a partial order generated by Algorithm 1. If  $\dim(P) = 2$ ,  $O(n^2)$  algorithm exists for discovering two maximal chain sub-graphs of  $G$  corresponding to the 2-CSC problem.

Proof:

The solution to 2-CSC [1] uses  $O(n^2)$  time and two linear orders are generated, if  $P$  is two dimensional partial order. Let  $L_1, L_2$  be the two linear orders generated. From lemma 1,  $ch(G) = \dim(P) = ch(B(P)) = 2$ . From lemma 2, there is one-to-one mapping between the linear order of  $P$  and maximal chain sub-graph of  $B(P)$ , which is generated by Algorithm 2. Thus, two maximal chain sub-graph covers of  $B(P)$  can be obtained by Algorithm 3. By lemma 4, there is one-to-one mapping between the maximal chain sub-graph of  $G$  and the maximal chain sub-graph of  $B(P)$ . Thus, by Algorithm 4, two maximal chain sub-graphs of  $G$  can be discovered. The complexity of Algorithms 2 – 4 are  $(n^2)$ . As a result, the assertion of the theorem,  $O(n^2)$  algorithm exists for discovering two maximal chain sub-graphs of  $G$  corresponding to the 2-CSC problem, is proved. Q.E.D.

Based on the above algorithms and theorem, we can design the algorithm listed in the following to compute the chain sub-graph covers for 2-CSC problem. The major steps in the algorithm are: First, the bipartite graph,  $G$ , as input to 2-CSC is converted to a partial order set,  $P$ ; Second,  $P$  is used as input to a two dimensional POSET problem. If  $P$  is two dimensional partial order set, two total orders are generated; Third, two maximal chain sub-graphs are derived from those two total orders.

Algorithm 5 2-CSC and Chain Sub-graph Covers

INPUT: Bipartite Graph  $G = (X, Y, E)$ ,  $X = \{x_1, x_2, \dots, x_l\}$ ,  
 $Y = \{y_1, y_2, \dots, y_m\}$

OUTPUT: Yes,  $ch(G) = 2$  and corresponding chain sub-graph covering, or No,  $ch(G)$  is not 2.

BEGIN

1.  $G$  is converted to a partial order set,  $P$  by applying Algorithm 1.
2.  $B(P)$  is generated from  $P$  by applying Algorithm 2.
3.  $P$  is used as input to a two dimensional POSET problem by applying Algorithm 6, which is listed in the appendix.
4. IF  $P$  is two dimensional partial order set. THEN Two total orders are generated. ELSE Output "ch(G) is not 2." and Exit.
5. Two maximal chain sub-graphs of  $B(P)$  are derived from those two total orders obtained in step 4 by applying Algorithm 3.
6. Compute the maximal chain sub-graphs of  $G$  from the maximal chain sub-graphs of  $B(P)$  obtained in Step 5 by applying Algorithm 4. Output "Yes, ch(G) is 2," and corresponding chain sub-graph coverings.

END

Steps 1-4 use Algorithms 1, 2 and 6, which depict the component activities in the  $O(n^2)$  process to resolve 2-CSC problem [1]. So, the complexity of steps 1-4 is  $O(n^2)$ . Step 5 uses Algorithm 3, which takes  $O(n^2)$  steps to identify  $E$  of the maximal bipartite graph  $H$  according to linear order  $L$  on  $X \cup Y$  of  $P = (X \cup Y, R)$ . Step 6 uses Algorithm 4 to compute each maximal chain sub-graph of  $G$  as a subset of a corresponding

maximal chain sub-graph of  $B(P)$ , which takes  $O(n^2)$  steps. As a result, the complexity of this algorithm (Algorithm 5) is  $O(n^2)$ .

### 4. Example of Algorithm 5

Step 1:

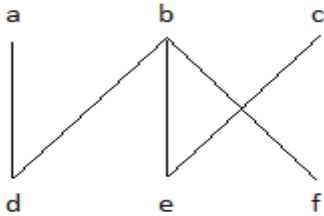


Fig. 1. Bipartite graph  $G$

$P=(X, R)$   
 $X = \{a, b, c, d, e, f\}$ ,  $R = \{(b,f), (f,d), (d,c), (f,c), (c,a)\}$

Step 2:

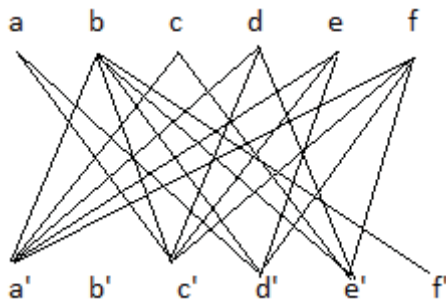


Fig. 2.  $B(P)$

Step 3-4:

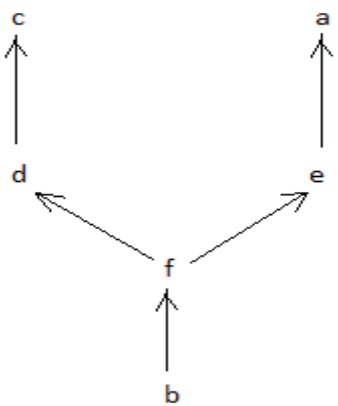


Fig. 3. DAG of  $P$

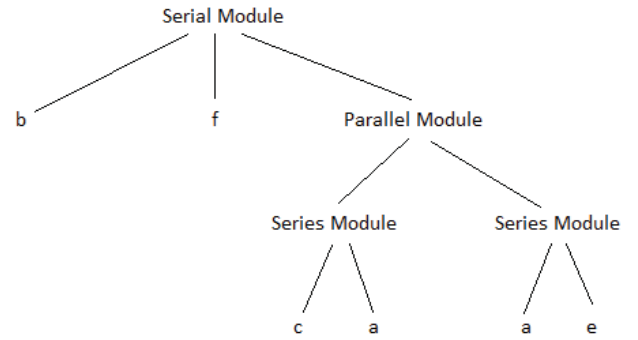


Fig. 4. Modular representation of the DAG of  $P$

Two total order L1, L2

- L1     b, f, d, c, e, a
- L2     b, f, e, a, d, c

Step 5:

Maximal Chain Sub-graphs for  $B(P)$ :  $C_1, C_2$

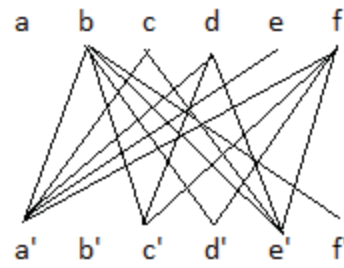


Fig. 5  $C_1$ , mapped from L1

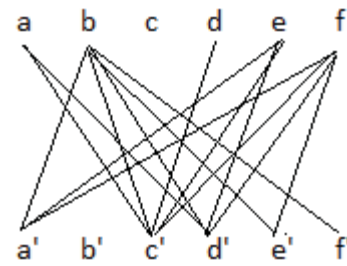


Fig. 6.  $C_2$ , mapped from L 2

Step 6:

Maximal chain sub-graphs for  $G$



Fig. 7. Extracted from  $C_1$



Fig. 8. Extracted from  $C_2$ 

## 5. Conclusion and comparison with related work

### CONCLUSION AND COMPARISON WITH RELATED WORK

This paper relates the component ideas in the solution to 2-CSC problem and shows the understanding that a more useful solution can be obtained. After compiling the ideas into interrelated knowledge, we find the evidence to support a more exact solution beyond a “yes” answer to 2-CSC problem. The identified evidence is examined and utilized to design executable steps formulating the algorithm proposed in this paper.

The major steps in the algorithm are: First, the bipartite graph,  $G$ , as input to 2-CSC is converted to a partial order set,

$P$ ; Second,  $P$  is used as input to a two dimensional POSET problem. If  $P$  is two dimensional partial order set, two total orders are generated and  $ch(G) = 2$ ; Otherwise, the algorithm outputs “No” and stop. Third, two maximal chain sub-graphs for a bipartite graph  $B(P)$  are derived from those two total orders. Fourth, maximal chain sub-graphs of  $G$ , are derived from those of  $B(P)$ . The proposed algorithm runs  $O(n^2)$  in time, where  $n$  is the number of vertices of  $G$ .

Once the chain sub-graphs can be identified, some problems reducible to 2-CSC can produce more exact answer from identified chain sub-graphs as a result in the same time complexity, too. Those problems include: threshold number 2 on split graphs, Ferrers dimension 2, biorder dimension 2 and interval dimension 2. We can acquire, respectively, two threshold graphs covering the split graph, two Ferrer graphs covering the directed graph, two biorders with intersection being the given biorder, two interval orders with intersection being the given interval order.

## 6. References

- [1] Tze-Heng Ma and Jeremy P. Spinrad, “On the 2-Chain Subgraph Cover and Related Problems”, Journal of Algorithm 17, pp 251 – 268, 1994.
- [2] B. Dushnik, E.W. Miller, “Partially Ordered Sets”, American Journal of Mathematics, vol. 63, pp. 600-610, 1941.
- [3] J. Spinrad, Two Dimensional Partial Orders, Ph.D. Thesis, Department of Electrical Engineering and Computer Sciences, Princeton University, 1982.

## Appendix

### List of Algorithm for the Recognition of Two Dimensional Partial Order [3]

Algorithm 6 Recognition of Two Dimensional Partial Order

Step 0. Given a partial order  $P = (X \cup Y, R)$ , represent it using a DAG,  $G$ , with vertex set  $X \cup Y$ , and an edge between two vertices  $u, v$  if  $uRv$ .

Step 1. Construct a modular representation for  $G$ .

- a. Consider the whole graph  $G$  as a module,  $M$ .

Recursive( $M$ )

- b. Choose any vertex  $u$  of  $M$

If  $u$  is related to every other vertex (no edge between  $u$  and  $X \cup Y - \{u\}$  in  $M_{CC}$ ),

$M$  is a series module,  $S$ , and partition  $M$  into three components:

1. vertices dominate  $u$ ,  $m_0$ . Call Recursive( $m_0$ ).
2. vertices dominated by  $u$ ,  $m_1$ . Call Recursive( $m_1$ ).
3.  $u$  itself,  $m_3$ . An end node in the tree.

else//Some  $v$  exists which is unrelated to  $u$ .

Select a vertex  $v \in M - \{u\}$  that is not related to  $u$ .

Find  $M_{u,v}$ , the smallest module that contains  $u$  and  $v$ .

Find  $M_u$ , the set of vertices which are related to  $u$  and unrelated to  $v$ .

Grow  $M_u$  by repeating the following process.

Repeat until no more addition to  $M_u$  is possible

Select a vertex  $w \in M_u$  and add to  $M_u$  the vertices that are

1. related to  $v$  and unrelated to  $w$
2. unrelated to  $v$  and related to  $w$

End

If  $v \notin M_u$ , grow  $M_v$  the same way as we do for  $M_u$ .

Find  $M_v$ , the set of vertices which are related to  $v$  and unrelated to  $u$ .

Grow  $M_v$  by repeating the following process.

Repeat until no more addition to  $M_v$  is possible

Select a vertex  $w \in M_v$  and add to  $M_v$  the vertices that are

1. related to  $u$  and unrelated to  $w$
2. unrelated to  $u$  and related to  $w$

End

If  $v \notin M_u$  and  $u \notin M_v$ ,

1.  $M_{u,v} = M_u \cup M_v$  and modular representation of  $M_{u,v}$ :

Root labeled  $P$  with  $M_u$  and  $M_v$  as children

2. Recursive( $M_u$ ), Recursive( $M_v$ )

Decompose  $M_u$

Consider vertices of  $M_{u,v}$  only

- a. Let  $x$  be a vertex brought to  $M_u$  during the last stage of its growth.
- b. Find  $M_{u,x}$  like we did for  $M_{u,v}$ , but considering vertices of  $M_{u,v}$  only. Refine  $M_u$  and  $M_x$ . Consider vertices of  $M_{u,x}$  only.
- c. Let  $y$  be a vertex brought to  $M_u$  during the last stage of its growth.
- d. Find  $M_{u,y}$  like we did for  $M_{u,v}$ , but considering vertices of  $M_{u,x}$  only. Refine  $M_u$  and  $M_y$ .
- e. ... until  $M_u$  is a single vertex

Decompose  $M_v$

... (similar)

3. Recursive( $M - M_{u,v}$ )/ $u$  represents  $M_{u,v}$  in term of adjacency

If  $v \notin M_u$  and  $u \in M_v$ , exchange the role of  $u$  and  $v$ . Continue with the else.

else//  $v \in M_u$

1.  $M_{u,v} = M_u$  and the modular representation of  $M_{u,v}$ :

2. Root labeled  $N$ (neighborhood module)

3. Decompose  $M_{u,v}$

- a. Partition  $M_{u,v}$  into two sets

$M_v$ : containing all vertices that were added to  $M_{u,v}$  at the same time as  $v$ .

$M_{u,v} - M_v$

- b. Refine  $M_v$ : so any two vertices in a partition relates in the same manner to all vertices of all other partitions. Every partition generated in this manner will be a submodule in the final decomposition (under current  $N$ )

Recursive(submodule)

- c. Recursive( $M_{u,v} - M_v$ )

4. Recursive( $M - M_{u,v}$ )

Step 2. From the modular representation, we compute a non-separating list,  $L_0$ , for  $G$ .

Begin from leaf to construct the non-separating list. Assume in obtaining the non-separating list for a module, the list of its submodule has been obtained.

If  $M$  is a parallel module, its listing is computed by concatenating the listings for all of its children in any order.

If  $M$  is a series module, the listing is obtained by concatenating the listings of its children, so that the listing for  $M_i$  precedes the listing for  $M_j$  if and only if every vertex of  $M_i$  dominates every vertex of  $M_j$ .

If  $M$  is a neighborhood module, the children of  $M$  are maximal submodules of  $M$ .

Let these submodules  $M_1, M_2, \dots, M_k$ .

Create set  $D = \{v_i | v_i \text{ is a vertex chosen arbitrarily from } M_i \text{ for } i = 1 \dots k\}$

( $v_i$  is called the representative descendant of  $M_i$ .)

Arrange the elements of  $D$  in a non-separating list called target listing.

Replace each  $v$  in the target listing by the non-separating listing for corresponding submodule.

The result is a non-separating list for  $M$ .

Step 3. From  $L_0$ , we compute a pair of listings,  $L_1, L_2$ .

$L_0$  is used as  $L_1$ .  $L_2$  is constructed in the following steps:

1. For each  $v \in V(G)$ , find the value  $n$  that is the total number of vertices  $x$  in  $G$ , such that either  $v$  dominates  $x$ , or  $x$  precedes  $v$  in  $L$  and  $x, v$  are not related.

2.  $L_2$  is constructed by sorting the vertices in  $G$  according to the value  $n$  obtained in previous step.

Step 4. Verify if  $L_1, L_2$  represent  $G$  as a two dimensional partial order DAG. If yes,  $P$  is a two dimensional partial order, and  $L_1, L_2$  are two total orders on  $X \cup Y$  with intersection being  $P$ . If no,  $P$  is not a two dimensional partial order.

# Measures/Models for Hierarchical Objects: of Quality, Quantity, Teaching, and Rubrics

H.M. Hubey, Professor

Dept. of Computer Science  
Montclair State University  
Upper Montclair, New Jersey 07043 USA  
hubeyh@mail.montclair.edu

**Abstract**—Various simple consistent, coherent models, representations, measurement methods and metrics are proposed for the CS2013 Body of Knowledge [consisting of Knowledge Areas by the Joint Task Force ACM/IEEE], and for similar hierarchical models. A short philosophical history is given and minor changes are suggested including possibility for CSIT GenEd (General Education) rubrics, as exemplars/prototypes employing CSIT concepts.

**Index Terms**—Karnaugh, Hamming, hierarchy, metric, measurement, quality, hypercube, torus, prototype, paradigm.

## I. INTRODUCTION

“The CS2013 Body of Knowledge is organized into a set of 18 Knowledge Areas (KAs), corresponding to topical areas of study in computing.” So states the The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society, 2013. [1][2] We have come a long way since the legends of Computer Science started creating of our field. Among the thoughts that created the revolutionary changes in education were the comments by D. McCracken, essentially, that all the required courses for computer science were math courses and all the computer science courses were optional. We may have tilted too far in the other direction.

## II. PHILOSOPHICAL OVERVIEW

Probably the most easily seen (or among the most easily agreeable) clustering or categorization of the concepts/clusters known as Knowledge Domains prescribed in the Ironman Draft [1][2] is the clustering in Table 1.

| Cluster          | Rationality                                  |
|------------------|--|
| AL, DS           | Theory, Mathematical Grounding               |
| AR, OS           | Theory, Not Only Mathematical                |
| IAS, PL, SDF, SE | All About Software                           |
| SF, PB, PBD      | All About Hardware                           |
| HCI, GV, NC      | Communication with Intelligent Entities      |
| CN, IS, SP, IM   | Communication with the Larger External World |

The possible “justification” for these may be as in the second column with some possibly esoteric clusters or groupings. Obviously we have to stretch some meanings to create such a clustering. Furthermore, they could have had different namings or explanations. For example Cluster 1 and 2 can be combined and simply called “Grounding” or “Theory” (using these terms loosely, since it is all theory and all realization also). That would produce Table 2 (with some classic, modern and postmodern philosophy thrown in!).

Table 2: Idealized Clusters

### Communication (with the External World)

HCI - Human-Computer Interaction--*with Humans*  
GV - Graphics and Visualization--*Another mode, another scale*  
NC - Networking and Communications--*With Intelligent Entities*  
CNSP- Computational Science- Social Issues- *Defining Self!*

### Hardware Matters

SF - Systems Fundamentals--*Integrate AR and OS*  
PD - Parallel and Distributed Computing--*Scale upwards*  
PBD - Platform-based Development-- *Partial “turnkeys”*  
IS - Intelligent Systems--(Knowledge) *What OUGHT it be?*

### Software Material/Activities

IAS - Information Assurance and Security-- *Protection/defense*  
PL - Programming Languages--*Multiculturalism (or evolution!)*  
SDFSE -Software Development & Engineering--*Here and Now*  
IM - Information Management-- *Including Inf Retr & Inf Quality*

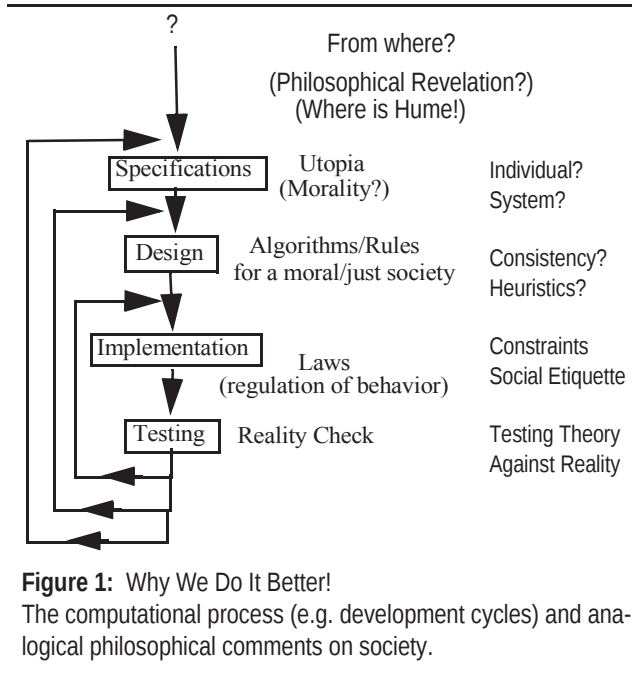
### Theory (e.g. Not Only Math --Undergirds all)

AL - Algorithms and Complexity--*Math of the Special Kind*  
DS - Discrete Structures-- *Math of a Special Kind*  
AR -Architecture and Organization--*HW (Math and CS Theory)*  
OS - Operating Systems-- *SW (more Math, and CS Theory)*

Notwithstanding the fact, that despite ostensibly nobody understanding what concepts are, we use them under different names all the time. They are apparently either (i) exemplars, (ii) prototypes (the best exemplars) or (iii) theories [3][4][5]. Obviously KA’s are “concepts”. At their simplest, they are categories/classes/sets (nouns or predicates) or clusters. They are “objects”. They are, then, hierarchically arranged since they are always complex.

How many KA’s (clusters/concepts) are there? There has to be some reasons why we might have different number of clusters. It is, after all, a “theory”. Of course this is a design problem and how many clusters (or layers since as computer scientists we must always think of hierarchy) is

itself a part of the over all analysis and synthesis. We are all familiar with the stories e.g. the OSI-ISO has 7 layers and DoD has 5. [6] The computer system itself (as we teach it) has many layers. It's as much a part of pedagogy and epistemology as it is a part of everyday reality. And we (in CSIT) apparently excel in "design".



The higher we go, and the more abstract we get, the more society (or at least the "intellectuals") will see us as modern philosophers and less as slaves in Classical Greece. It will be shown later, but first, some pseudo-history and high-level evolution of our field, either for justification or for fun.

For the most complex problems in the universe, the typical "method" of working on them is making analogies that even people who cannot read can comprehend; of course, this is in philosophy, and the ologies (specialized philosophies). Where then, does our field fit in using "better analogies" (e.g. some representations from CSIT and Math)?

It is obvious from Fig. 2, that Mathematics is "required" for the S in STEM. That is, logically speaking  $\bar{M} \rightarrow \bar{S}$ ; if there is no math in it, there is no science in it. This goes in Ologies as "structure" while we in STEM know that this is just a code-word for "math" since we all know that math is the science of patterns and structures. Where is the "quality" of any field?

Clearly, when philosophers in epistemology talk about quality vs. quantity, they are talking about *intensive* vs. *extensive* parameters of thermodynamics. Measurements of "value"(!) are of the type  $qQ$  (*quality*  $\times$  *Quantity*) so that  $d(qQ) = Qdq + qdQ$ . Terms of the latter type show up in macroscopic thermodynamics such as  $dW = pdV$  or  $dq = TdS$ . Economists measure value the same way. For

example, let  $p_j = q_j$  be the price/quality of the  $j$ th type cell phone and  $N_j$  be the number of cell phones of type  $j$ . Then the total value of a collection of cell phones is clearly given by  $\sum_j p_j N_j = \sum_j q_j N_j$ . Furthermore let the average quality of the collection be  $q$  and the total number be  $N$ . Suppose now that this collection is arbitrarily split into two collections. The quality of each subcollection and the number in each collection obey  $q_1 = q_2 = q$  and  $N_1 + N_2 = N$ . Clearly, these are the definitions of intensive and extensive parameters in physics. For example, for a given volume (system/collection) of gas, if we change the extensive parameter  $V$  by splitting the system into two, the changes are:  $V_1 + V_2 = V$  and  $T_1 = T_2 = T$  and  $p_1 = p_2 = p$ .

Computer science is as much about computers as astronomy is about telescopes" (Edsger Dijkstra)

|             |              |                              |                            |                                   |
|-------------|--------------|------------------------------|----------------------------|-----------------------------------|
|             |              | Users                        | Users                      | Masses                            |
|             |              | Information Technocracy      | Technocracy (technicians?) |                                   |
| Realization | Practice     | Information Technology       | Technology                 | Business Administration           |
|             | Design       | SWE Informatics Informatique | Engineering                | Business Administration Economics |
|             | Less Certain | Computing Science            | Physics                    | Economics                         |
|             | More Certain | Theory                       |                            | Math                              |

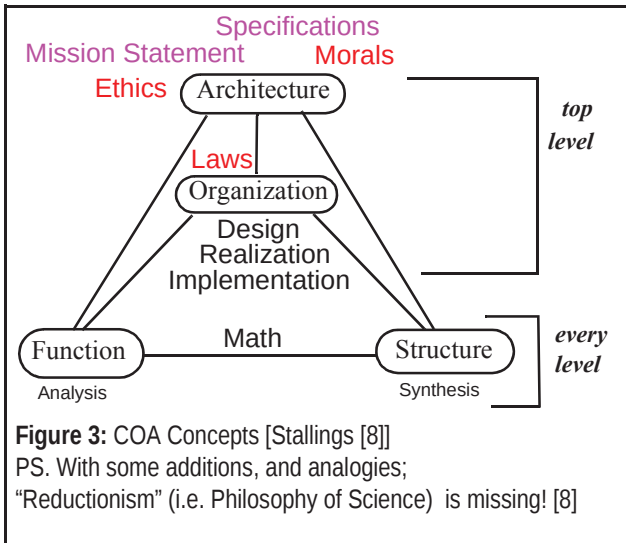
**Figure 2: Comparison of CSIT to Other Domains**  
Some of the words are used in deliberately elastic ways. See the quote above. One might notice that the the word "computer" is missing as is the phrase "computing machinery".

Since it is practically guaranteed that almost anything designed in the future by engineers will have plenty of computation in it, one also might stick CSIT as a part of engineering as an alternative e.g. a special kind of engineering.

While we are in the measurement philosophy mood, it seems a mistake to leave CSIT to the mercy of the Education Department rubrics that could have been written by Plato or Aristotle [7]. The concept of "structure" happens to be the one that is sorely lacking outside STEM. Of all the STEM fields, CSIT is the one that deals with data, information, and knowledge in a very human-centered way.

A rubric constructed (Fig. 3), for example using the COA [8] as a prototype (or at least one of the exemplars), especially for GenEd courses in CSIT would go a long way

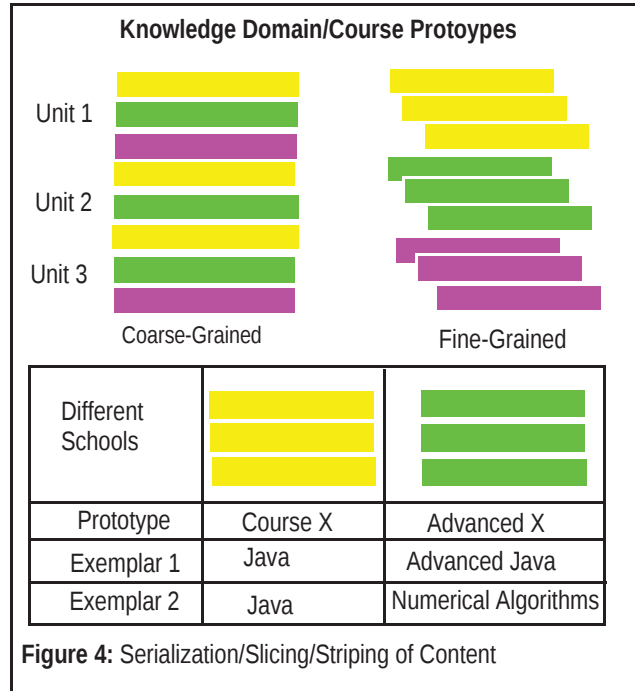
towards providing ammunition against possible university political machinations from medieval metaphysicians who invaded, occupied and colonized the university system, and eradicated science and scientists from the curriculum, and human history over the last 300 years. And the result: “One half of college bound high school students taking the SAT exams do not understand fractions and decimals” [9]. Nor is the professoriate better, where “only 8% clearly differentiated between an assumption and an inference, and only 4% differentiated between an inference and an implication” [16] [17]. Now that “computation” is rumored to be required in K12 a prototype of a STEM rubric might be a good idea [12]. How long can the tail be allowed to wag the dog?



It seems obvious that every “CSIT” course possesses (implicitly or explicitly) some programming (implicit as in only the algorithm [in NL or in pseudocode]), or explicit (as in written in some computer language).

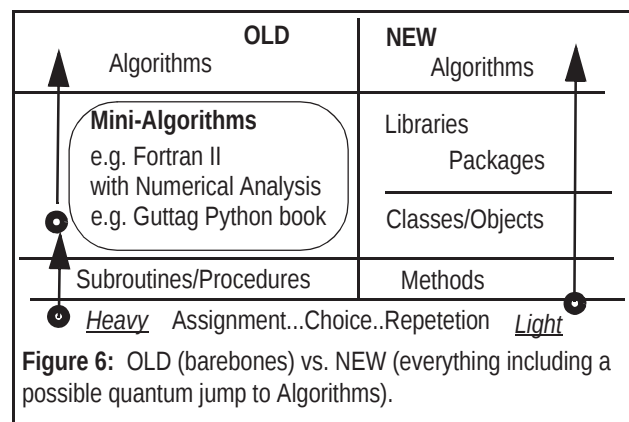
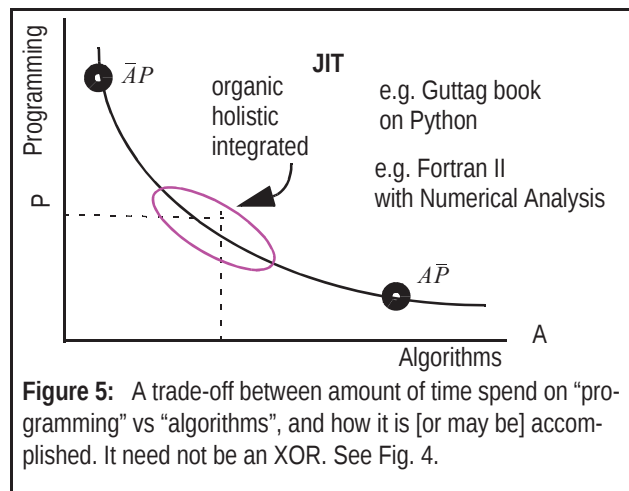
It can be seen that there is a kind of an XOR (or a trade-off, Fig. 5) between the various concepts. It seems just as obvious that every program consists of an algorithm including very trivial ones. And algorithms are [the new] math since it is obvious that they represent structure (of the program/process) in-time instead of in-space (Fig 15). The concept is explicit in a simple way in the representation of a macro (expansion in space) vs. subroutine (expansion in time) which will be very familiar with those who taught C. At one time the argument was between those who wanted a “production language” (like C) vs. those who wanted an “academic language” like Pascal. As we know C won out, and then got “fixed” by computer scientists until it was wiped out by Java. We may be experiencing a similar stage right now.

Alternatively, the trade-off as in Fig. 5 may be that of Breadth (abscissa) vs. Depth (ordinate). Then we could have the classical/conventional *broad* × *shallow* (e.g. liberal arts) vs. *narrow* × *deep* education of STEM [Fig. 16]. This is the old canard “An expert is one who knows everything about nothing, and a generalist is one who knows nothing about everything.”

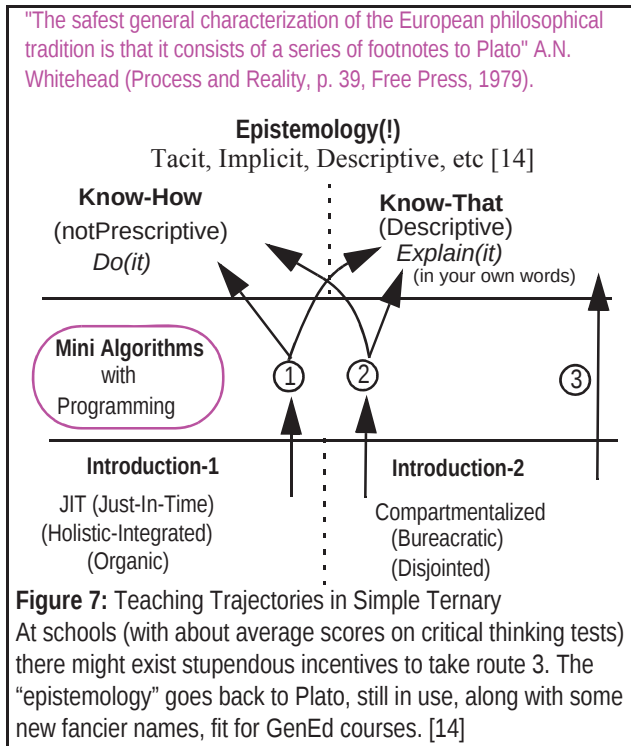


**Figure 4: Serialization/Slicing/Stripping of Content**

There are obviously elements of truth in these and it seems that we (all or some of us in CSIT) might have taken a path over the last N years which could be coming back to bite us. See Fig. 6



Without the middle layer of connecting the Discrete Math and the High School math to programming and to algorithms, it is difficult for students to jump to higher levels. The algorithms course(s) seem to have become pointers and trees. In the real world, our K12 system seems to be producing students who can barely comprehend fractions or decimals thus the course in Discrete Math itself is a serious problem as is the abstraction in OOPs.

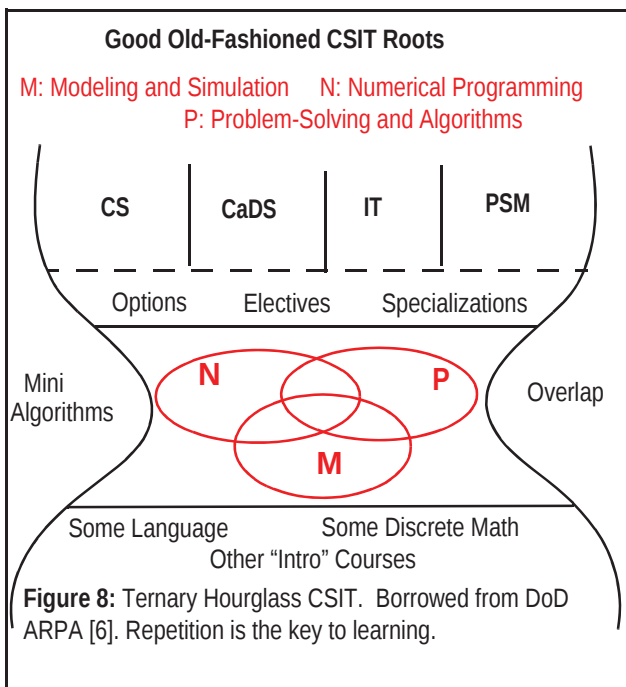


An example of an organic holistic first layer book is the Python book (Gutttag) used at MIT. There are many examples of bureaucratic/compartmentalized introductory books. Some schools might be tempted to skip the middle layer and go to upper layer courses which consist mostly of "explain in your own words" kinds of courses e.g. descriptive courses as in the Ologies. See Fig. 7.

The good news is that our domain of CSIT which may have forgotten the middle layer is now being forced into regain/recall it via the CaDS (Computational and Data Sciences) requirements which seems to have forced upon our field the original problems of Space Constraints [6]. A re-introduction of this missing "middle layer" would result in something like Fig. 8. We can bring it back and tie the complexity of upper layers in an organic way to the introductory courses of the undergirding.

III. HOLISTIC MEASUREMENT

We can now "revisit" Table I and Table II after this brief excursion. In order to introduce the specific methods of measurement and organization a 4-cluster model (with each cluster consisting of 4-KA's) is introduced. SD and SDF have been joined together to form a single KA since they are both about software development. Ditto for CN and SP since they are both about communication with the external world in different ways. Of course there are many other ways to attempt to categorize complex things. One simple way is to try to envision CSIT in a hierarchy in analogy with other fields (preferably scientific fields). See Fig. 2.



|  |   |   |   |
|--|---|---|---|
| <b>Specifications</b><br>FUTURE-POINTING                   | ⓕ<br>NC - HCI-GV-CNSP   |   | suprastructure<br>infrastucture<br>exostructure   |
| <b>Realization</b><br>HERE AND NOW<br>(Proximate Research) | ⓗ<br>PBD - PD-SF-IS<br><i>Hardware (Brain)</i><br><i>(Material/Solid)</i> | Ⓢ<br>IAS - PL-SDFSE-IM<br><i>Software (Mind)</i><br><i>(Ephemereal)</i> | superstructure<br>endostructure<br>interstructure |
| <b>Theory</b><br>CLASSICAL<br>(Basics)                     | Ⓣ<br>DS - AL - OS - AR  |   | foundation<br>infrastucture                       |

**Figure 9: Four Component Three-Level Model**

One of the things that distinguishes STEM from the Ologies is that there are large changes in q (quality) which represents the intensity of mathematics used). The Ologies are of the type where q=constant but we get massive quantities of literature at the same level of thought as Socrates. Indeed,

the great philosopher/mathematician A.N. Whitehead said so explicitly. The simple metrics here using Karnaugh maps and Hamming distances are a small step towards creating measurement metrics for measuring sets of categorical variables that is not statistics based. Indeed such work has already been done for Information Quality [15]. There are many other ways to measure such things, in addition, to the simple Hamming distance given here. Research is continuing along these lines [15].

What will be shown and used is a brief outline of:

- 1) A Complete Graph,  $C_4$ , which is planar and on which we could show the edges representing effects; Fig 10.
- 2) set of differential/difference equations to simulate the evolution of the system
- 3) A Hypercube,  $Q_4$ , to represent the KA's. Fig. 11.
- 4) Obviously the Karnaugh map and the Hypercube are intimately related and hence A Karnaugh Map for measurement of the performance and for Hamming Distances. Fig. 12.
- 5) A representation of the topological space for such measurements. Fig. 13

The digraph of  $C_4$  is self-explanatory. The edges show the

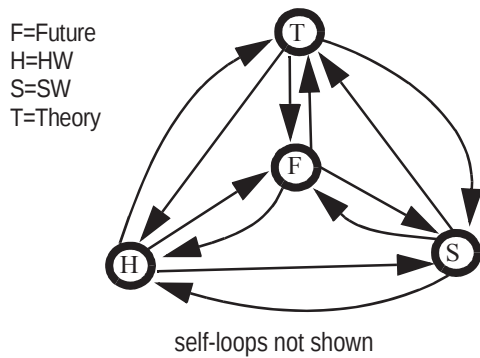


Figure 10: Complete Graph of 4

effects of the four KAs on each other. One can easily create dynamic models for such relationships, for example a set of linear differential equations such as  $\frac{d\hat{x}}{dt} = A\hat{x}$ . Some macro-economics models are of this type [18].

Unfortunately, for visualization of high dimensional data, the hypercubes are difficult to draw or to imagine in higher dimensions. It can now be seen that the numbering scheme for the hypercube was already using the reflected. It is because of this that the “adjacent” cells on a Karnaugh map are distance one apart using a Hamming metric. The hypercube can also display clusters in datamining and can be used to display the existence (or lack) of given dimensions/attributes [i.e. pass/fail] as shown in Fig. 11. The hypercube low dimensional spaces (i.e.  $n \leq 4$ ), can be displayed in two-dimensions as a table (Karnaugh map) and there exists a natural distance (Hamming) metric defined on the K-map.

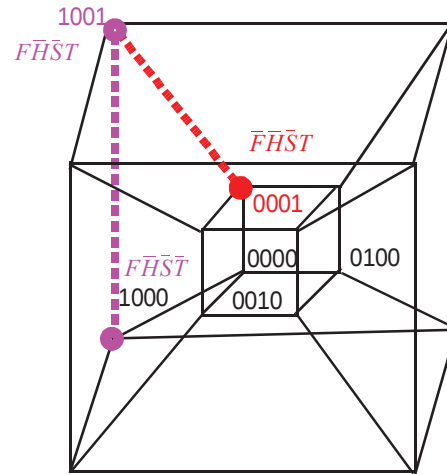


Figure 11: CSIT Hypercube:

The colored-dotted lines show a cluster  $\bar{H}\bar{S}T + F\bar{H}\bar{S}$  which might be thought of as “nonlinear”.

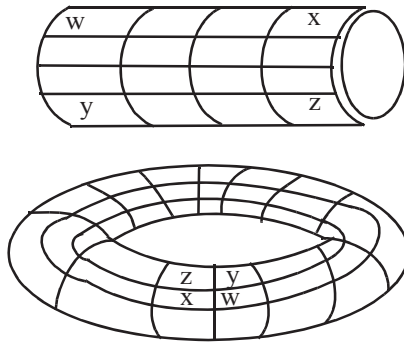
It is reminiscent of the city block metric used in data mining procedures. After we obtain scores (as in Fig. 12) for T, H, S, F, we can then combine all of them in another Karnaugh map for the whole CSIT program. Finally, wrapping the Karnaugh map on a torus shows us the natural topological space of such phenomena and such metrics. We wrap the table on a cylinder such that the top edge is adjacent to the bottom edge.

| AD | RS |    |    |    |
|----|----|----|----|----|
|    | 00 | 01 | 11 | 10 |
| 00 | W  |    |    | X  |
| 01 |    |    |    |    |
| 11 |    |    |    |    |
| 10 | Y  |    |    | Z  |

Theory  
A=AL  
D=DS  
R=AR  
S=OS

Figure 12: Karnaugh Map for T (Theory) (See Fig. 9)

Therefore the cell w would be adjacent to y and the cell x would be adjacent to z. Then we bend the cylinder so that the ends come together to form the torus. Now the left and right edges of the table are adjacent resulting in Figure 13 torus. Of course, there would only be 16 or less cells on the whole table. Obviously bigger tables can be wrapped on the torus and all cells that are adjacent on the torus are distance of 1 from each other, but not all cells that are distance 1 from each other will be adjacent on the torus.



**Figure 13:** Karnagh Map Topological Space for CSIT

#### IV. SUMMARY/EPILOGUE

At a time when half of college-bound high school seniors do not understand fractions or decimals [9], and math teachers do not understand why one must multiply base times height to obtain the area of a rectangle [13], perhaps changing the curriculum slightly to connect (i) algorithms, (ii) programming [language(s)] and (iii) high school math in organic, holistic and integrated ways might be an idea to bring back, and forget our amnesia. It will be better preparation for the upper levels than what we have now, at least for the schools who are not getting the top students. Obviously, those universities who are getting students who already had these courses, there is no need for the MNP sequence.

While we are in the asking mood, it seems a mistake to leave CSIT to the mercy of Education Department rubrics produced by what seem like premedieval metaphysicians that could have been written by Plato or Aristotle. Of all the STEM fields, CSIT is the one that deals with {data, information, knowledge, intelligence}[i.e. epistemology] in a very human-centered way, and could provide a prototype for other such CSIT rubrics or for other STEM fields. Modern epistemology is the Urheimat of Computing Sciences, (CS) and we should reclaim our territory. We do not operate machines. It seems that CSIT model is the one everyone has been looking for all these centuries! It is probably a good idea to express some of these in CSIT courses. These rubrics and the description of the program could be the basis of a new “Philosophy of Science”.

After all, a just society has to be “designed”, and CS design is now accessible to all. Of course, things are still missing from this picture. They have to be; it is too difficult to express so easily. It is obvious that things have to be tested and to accomplish the task one needs to measure. And these measurements also have to come from CS. And the other is that we have to expand this model to improve the measurement aspects by extending from the simple Boolean model that uses the Hamming metric. Research is continuing along these lines.

#### REFERENCES

- [1] <https://www.acm.org/education/CS2013-final-report.pdf>
- [2] <http://ai.stanford.edu/users/sahami/CS2013/ironman-draft/cs2013-ironman-v0.8.pdf>
- [3] Margolis, Eric, and S. Laurence (ed), Concepts: Core Readings, MIT Press (Bradford Books), 1999
- [4] Carey, Susan, Origin of Concepts (Oxford Series in Cognitive Development), Oxford University Press, 2011.
- [5] Machery, Edouard, Doing Without Concepts, Oxford University Press, 2011.
- [6] DoD HourGlass Model  
[http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-02-introduction-to-eecs-ii-digital-communication-systems-fall-2012/lecture-slides/MIT6\\_02F12\\_lec23.pdf](http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-02-introduction-to-eecs-ii-digital-communication-systems-fall-2012/lecture-slides/MIT6_02F12_lec23.pdf)
- [7] Stevens, D, and A. Levi, Introduction to Rubrics: An Assessment Tool to Save Grading Time, Convey Effective Feedback and Promote Student Learning, Stylus Publishing LLC, Sterling, VA, 2005.
- [8] Stallings, Wm, Computer Organization and Architecture: Designing for Performance, Pearson, 2012.
- [9] Susan Carey, Origin of Concepts, Behavioral and Brain Sciences / Volume 34 / Issue 03 / June 2011, pp 113-124  
<http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3489495/>
- [10a] Kean XSEDE Workshop  
<http://preview.tinyurl.com/o7lfy4m>
- [10b] Kean XSEDE Workshop  
<http://preview.tinyurl.com/onemzoy>
- [11a] Integrating Computational Science into the Curriculum – Steven Gordon, Ohio Supercomputer Center, Education Lead, XSEDE Project  
<http://preview.tinyurl.com/nucn4s6>
- [11b] Ohio CaDS <http://preview.tinyurl.com/p9po8q9>
- [12] Mims, Christopher, Wall Street Journal, April 26, 2015  
<http://www.wsj.com/articles/why-coding-is-your-childs-key-to-unlocking-the-future-1430080118>
- [13] <http://www.ams.org/notices/200502/fea-kenschaft.pdf>
- [14] (See for example) <http://preview.tinyurl.com/p9moq2z>
- [15] Krupapaben Patel (advisee), MS Project, Montclair State University [2014]
- [16] <http://files.eric.ed.gov/fulltext/ED410276.pdf>
- [17] <http://preview.tinyurl.com/nwod5tw>  
[Criticalthinking.org]
- [18] Hubey, H.M. Equations of Motion of Macroeconomics, Proceedings of International Symposium on Economic Modeling at the London School of Economics, July 9-11, 1991.



APPENDIX

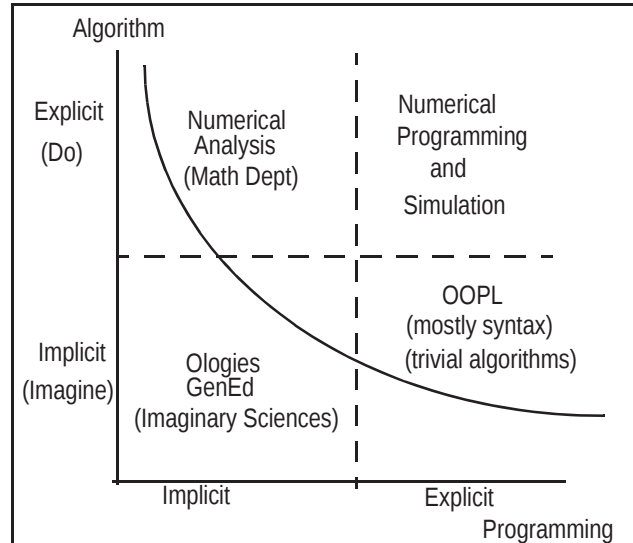
**Table 3**

|                                 |                                   |  |
|---------------------------------|-----------------------------------|--|
| Comte<br>(Evolution of Society) | Whitehead<br>(Education of Youth) | <i>Computation</i><br>(CSIT View)                                |
| Scientific                      | Generalization<br>(Science)       | <u>Knowledge</u><br>(Being worked on--AGI)                       |
| Social<br>Physics               | Precision<br>(facts, factoids)    | <u>Information</u><br>(Data meaningful to an intelligent entity) |
| Metaphysical<br>Theological     | Romance<br>(narratives)           | <u>Data</u><br>(Raw physical stuff)                              |

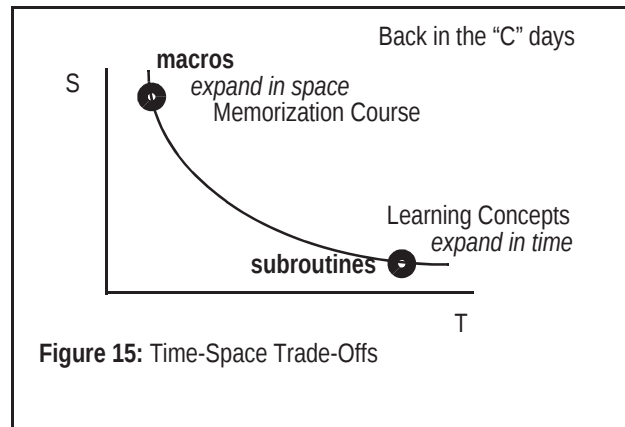
Philosophical stages of learning of humanity.

**Table 4**

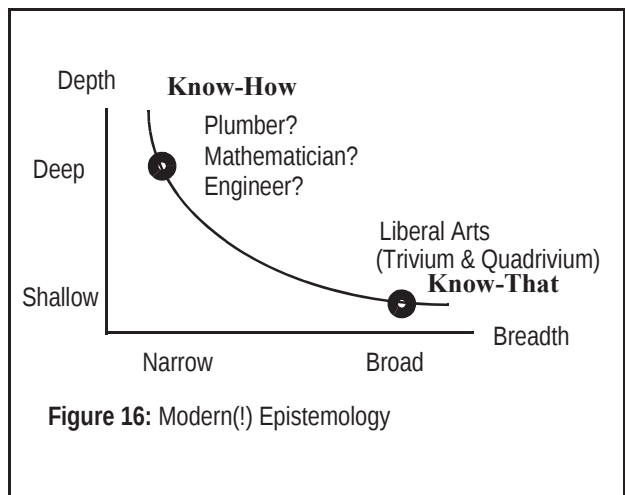
| CC2001 Computer Science Final Report Processes   |                                   |                                  |
|--|-----------------------------------|----------------------------------|
| Theory Paradigm<br>(Math)  | Science Paradigm<br>(Abstraction) | Engineering Paradigm<br>(Design) |
| Definitions<br>Axioms  | Data,<br>Hypothesis               | Requirements                     |
| Theorems   | Modeling,<br>Prediction           | Specifications                   |
| Proofs   | Experiment, Design                | Design,<br>Implementation        |
| Interpretation<br>(Semantics)  | Analysis<br>(Semantics)           | Testing,<br>Analysis             |
| Processes and Themes from CC 1991<br><a href="http://www.acm.org/education/curric_vols/cc2001.pdf">http://www.acm.org/education/curric_vols/cc2001.pdf</a> |                                   |                                  |



**Figure 14:** Illustration of Some Divisions



**Figure 15:** Time-Space Trade-Offs



**Figure 16:** Modern(!) Epistemology

# An Algorithm for Counting the Number of Edge Covers on Acyclic Graphs

J. Raymundo Marcial-Romero<sup>1</sup>, Guillermo De Ita<sup>2</sup>, J. A. Hernández<sup>1</sup> and R. M. Valdovinos<sup>1</sup>

<sup>1</sup>Facultad de Ingeniería, UAEM, Toluca, México

<sup>2</sup>Facultad de Ciencias de la Computación, BUAP, Puebla, México

**Abstract**—Counting the number of edge covers on graphs, denoted as the #Edge\_Covers problem, is well known to be #P-complete. In this paper, we present an algorithm that compute the number of edge covers in polynomial time if and only if the graph is acyclic. Our algorithm is based on a post-order traversal of the spanning tree of the original graph.

**Keywords:** Counting the number of edge covers, efficient counting algorithms.

## 1. Introduction

Counting problems besides of being theoretically interesting, they also have a wide range of applicability on different areas. As a matter of example, it can be mentioned that if a propositional formula needs to be probabilistically tested to be true or given a graph, estimates the probability that it remains connected, in the case of a probability of failure of an edge, the estimation of such probabilities becomes a counting problem. Counting problems also arise naturally in Artificial Intelligence Research, when some methods are used in reasoning areas, such as computing the ‘degree of belief’ and ‘Bayesian belief networks’, which are computationally equivalent to counting the number of models to a propositional formula [4], [5], [11], [13].

Counting has become an important area in theoretical computer science, although it has received less attention than decision problems. There are few counting problems in graph theory that can be solved exactly in polynomial time, indeed an important line of research is to determine low-exponential upper bounds for the time complexity of hard counting problems.

An *edge cover* set of a graph  $G$  is a subset of edges covering all nodes of  $G$ . The problem of counting the number of edge cover sets of a graph, denoted as #Edge\_Covers, is a #P complete problem which has been proved via the reduction from #Twice-SAT to #Edge\_Covers [1].

Although the time complexity to compute exactly the edge cover sets on a graph is a hard problem, it is relevant to categorize the class of instances where counting the number of edge covers can be done in polynomial time. There is a scarce literature about the design of procedures for computing edge covers, and as far as the authors are aware, it is not known which is the largest polynomial class of graphs for the #Edge\_Covers problem.

In this work, the computation of #Edge\_Covers based on the topological structure of the graph will be addressed. A method

for counting edge covers for acyclic graphs is presented via a post-order traversal strategy.

In Section 2, it is briefly discussed the preliminaries of the paper. In Section 3, the basic topologies of a graph are presented, for which efficient procedures for solving the #Edge\_Covers problem have been designed. In this direction, it is shown that the #Edge\_Covers problem is solved in linear time over the size of a graph when the graph does not have cycles or it is acyclic. Those are topological cases for which a bound can be estimated from the branch and bound algorithm.

In Section 4, an algorithm to compute edge covers for acyclic graphs is described. A spanning tree of the original algorithm is built followed by a post-order traversal. Finally, in Section 5 the conclusions of the paper are presented.

## 2. Preliminaries

Let  $G = (V, E)$  be a simple graph (i.e. finite, undirected, loop-less and without multiple edges).  $V(G)$  and  $E(G)$  are also used to denote the set of vertices and edges, respectively, of graph  $G$ . A vertex and an incident edge are said to *cover* each other. The cardinality of a set  $A$  will be as usual denoted by  $|A|$ .

The neighbourhood of a vertex  $v \in V$  is the set  $N(v) = \{w \in V : \{v, w\} \in E(G)\}$ , and the closure neighbourhood of  $v$  is  $N[v] = N(v) \cup \{v\}$ . The degree of a node  $v$ , denoted by  $\delta(v)$ , is the number of neighbours it contains, that is  $\delta(v) = |N(v)|$ . A vertex  $v$  is said to be *pendant* if its neighbourhood consists of exactly one vertex; analogously an edge  $e$  is said to be *pendant* if one of its endpoints is a pendant vertex [10]. The degree of a graph  $G$  is  $\Delta(G) = \max_{x \in V(G)} \{\delta(x)\}$ .

Let  $G = (V, E)$  be a graph then  $S = (V', E')$  is a subgraph of  $G$  if  $V' \subseteq V$  and  $E'$  contain edges  $\{v, w\} \in E$  such that  $v, w \in V'$ . If  $E'$  contains every edge  $\{v, w\} \in E$  where  $v, w \in V'$ , then  $S$  is called the *subgraph of  $G$  induced by  $S$*  and is denoted by  $G||S$ . Let  $S$  be any subgraph,  $G - S$  will denote the induced graph  $G|| (V - V')$ . In the same way,  $G - v$  for  $v \in V(G)$  denotes the induced subgraph  $G|| (V - \{v\})$ , and  $G - e$  for  $e \in E(G)$  will denote the subgraph of  $G$  formed by  $V(G)$  and  $E(G) - \{e\}$ .

*Definition 1:* Let  $G$  be a graph then  $G$  is said to be connected if for each pair  $v, w \in V(G)$  there exists a path from  $u$  to  $v$ . The path may consist of more than one edge  $e \in E(G)$ . A *connected component* of  $G$  is a maximal induced subgraph of  $G$ , that is, a connected component is not a proper subgraph of any other connected subgraph of  $G$ .

For example, a tree graph is an acyclic connected graph. Let us denote a complete graph, a simple path and a simple cycle by  $K_n, P_n$  and  $C_n$  respectively, where  $n$  represents the number of nodes in the graph.

*Definition 2:* A vertex cover for a graph  $G = (V, E)$  is a subset of nodes  $U \subseteq V(G)$  that covers every edge of  $G$ ; that is, every edge has at least one endpoint in  $U$ . An edge cover for a graph  $G = (V, E)$  is a subset of edges  $\mathcal{E} \subseteq E(G)$  that cover all node of  $G$ , that is, for each  $u \in V(G)$  there is a  $v \in V(G)$  such that  $e = \{u, v\} \in \mathcal{E}$ .

### 2.1 Statement of the problem

The statement of the problem that this paper is concerned about can be established as follows: Let us consider a graph  $G = (V, E)$  and let  $C\mathcal{E}(G) = \{\mathcal{E} \subseteq E(G) : \mathcal{E} \text{ is an edge cover of } G\}$  be the set of edge covers for  $G$ . Let us also consider  $NE(G) = |C\mathcal{E}(G)|$  to be the number of edge covers of  $G$ , in different words  $NE(G)$  is the cardinality of the set  $C\mathcal{E}(G)$ . The problem of computing the number  $NE(G)$  for any graph will be called the #Edge.Covers problem.

### 3. Linear time Procedures for Counting Edge Covers

$NE(G)$  for any graph  $G$ , including the case when  $G$  is a disconnected graph, is computed as:  $NE(G) = \prod_{i=1}^k NE(G_i)$ , where  $k$  is the cardinality of the set of connected components of  $G$  and each  $G_i$  represents an element of this set. The set of connected components of  $G$  can be computed in linear time [1].

The edges of  $G$  appearing in all edge cover sets are called *fixed edges*. When an edge cover  $\mathcal{E}$  of  $G$  is being built, we distinguish between two different states of a node  $u$ ; we say that  $u$  is *free* when it has not still been covered by any edge of  $\mathcal{E}$ , otherwise the node is *covered*. We begin designing procedures for counting edge covers, considering the most common topologies of a network.

#### Case A: The Bus Topology

Let  $P_n = G = (V, E)$  be a linear bus (a path graph). We assume an order between vertices and edges in  $P_n$ , i.e. let  $V = \{v_0, v_1, \dots, v_n\}$  be the set of  $n + 1$  vertices and let  $e_i = \{v_{i-1}, v_i\}, 1 \leq i \leq n$  be the  $n$  edges of  $P_n$ .

Let  $G_i = (V_i, E_i), i = 0, \dots, n$  be the subgraphs induced by the first  $i$  nodes of  $V$ , i.e.  $G_0 = (\{v_0\}, \emptyset), G_1 = (\{v_0, v_1\}, \{e_1\}), \dots, G_n = P_n$ .  $G_i, i = 0, \dots, n$  is the family of induced subgraphs of  $G$  formed by the first  $i$  nodes of  $V$ . Let  $C\mathcal{E}(G_i)$  be the set of edge covers of each subgraph  $G_i, i = 0, \dots, n$ .

Each edge  $e_i, i = 1, \dots, n$  in the bus has associated an ordered pair  $(\alpha_i, \beta_i)$  of integer numbers where  $\alpha_i$  carries the number of edge cover sets of  $C\mathcal{E}(G_i)$  where the edge  $e_i$  appears in order to cover the node  $v_{i-1}$ , while  $\beta_i$  conveys the number of edge cover sets in  $C\mathcal{E}(G_i)$  where the edge  $e_i$  does not appear.

By traversing  $P_n$  in depth-first search, each pair  $(\alpha_i, \beta_i), i = 1, \dots, n$  is computed in accordance with the type of edge that  $e_i$  is.  $P_n$  has two fixed edges:  $e_1$  and  $e_n$ . The pair  $(1,0)$  is assigned to  $(\alpha_1, \beta_1)$  because  $e_1$  has to appear in all edge cover of  $P_n$ .

If we know the values  $(\alpha_{i-1}, \beta_{i-1})$  for any  $0 < i < n$ , then we know the number of times where the edge  $e_{i-1}$  appears or does not appear into the set of edge covers of  $G_i$ . When the edge  $e_i$  is being visited, the vertex  $v_{i-1}$  has to be covered considering its two incident edges:  $e_{i-1}$  and  $e_i$ . Any edge cover of  $C\mathcal{E}(G_i)$  containing the edge  $e_{i-1}$  ( $\alpha_{i-1}$  cases) has already covered  $v_{i-1}$  then the occurrence of  $e_i$  is optional. But for the edge covers where  $e_{i-1}$  does not appear ( $\beta_{i-1}$  cases)  $e_i$  must appear in order to cover  $v_{i-1}$ . This simple analysis shows that the number of edge covers where  $e_i$  appears is  $\alpha_{i-1} + \beta_{i-1}$  and that just in  $\alpha_{i-1}$  edge covers the edge  $e_i$  does not appear. Thus, we compute  $(\alpha_i, \beta_i)$  associated with the edge  $e_i$ , applying the Fibonacci recurrence relation.

$$\alpha_i = \alpha_{i-1} + \beta_{i-1}; \quad \beta_i = \alpha_{i-1} \tag{1}$$

When the search arrives to the last edge  $e_n$  of the linear bus, we have already computed the pair  $(\alpha_{n-1}, \beta_{n-1})$ ; since  $e_n$  is a fixed edge, it has to appear in all edge covers of  $P_n$ . We call  $\alpha_n = \alpha_{n-1} + \beta_{n-1}$  and  $\beta_n = 0$  the *recurrence for processing fixed edges (RPFE)*.

The pair associated with  $e_n$  is  $(\alpha_n, \beta_n) = (\alpha_{n-1} + \beta_{n-1}, 0)$ . The sum of the elements of this pair  $(\alpha_n, \beta_n)$  yields the number of edge covers:  $NE(P_n) = \alpha_n + \beta_n$ . Notice that  $NE(P_n)$  is computed in linear time over the number of edges in  $P_n$ . In figure 1 we present an example where  $\rightarrow$  denotes the application of recurrence (1), and  $\mapsto$  denotes the application of RPFE.

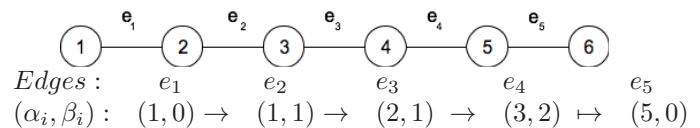


Fig. 1

COUNTING EDGE COVERS ON A LINEAR BUS

Recall that each Fibonacci number  $F_i$  can be bounded from above and from below by  $\phi^{i-2} \geq F_i \geq \phi^{i-1}, i \geq 1$ , where  $\phi = \frac{1}{2} \cdot (1 + \sqrt{5})$ .

*Theorem 3:* The number of edge cover sets of a path of  $n$  edges, is:

$$F_n = \text{ClosestInteger} \left[ \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^n \right].$$

*Proof:* The series  $(\alpha_i, \beta_i), i = 1, \dots, n$  used for computing  $NE(P_n)$ , coincides with the Fibonacci numbers:  $(F_1, F_0) \rightarrow (F_2, F_1) \rightarrow (F_3, F_2) \rightarrow \dots \rightarrow (F_{n-1}, F_{n-2}) \mapsto (F_n, 0)$ . Then, we infer that  $(\alpha_i, \beta_i) = (F_i, F_{i-1})$  for  $i = 1, \dots, n - 1$  and  $\alpha_n = F_n, \beta_n = 0$ . Thus,  $NE(P_n) = \alpha_n + \beta_n = F_n$ . ■

### Case B: The Tree Topology

Let  $T = (V, E)$  be a rooted tree. *Root-edges* in  $T$  are the edges with one endpoint in the root node; *leaf-edges* in  $T$  are the edges with one endpoint in a leaf node of  $T$ . Given any intermediate node  $v \in V$ , we call a *child-edge* of  $v$  to the edge connecting  $v$  with any of its children nodes, and the edge connecting  $v$  with its father node is called the *father-edge* of  $v$ .  $NE(T)$  is computed by traversing  $T$  in post-order and associating  $(\alpha_e, \beta_e)$  with each edge  $e \in E$ , except for the leaf edges.

#### Algorithm #Edge\_Covers\_for\_Trees( $T$ )

- 1) Reduce the input tree  $T$  to another tree  $T'$  by pruning all leaf nodes and leaf-edges from  $T$ , and by labeling as *covered* all father nodes of the original leaf nodes of  $T$  (see figure 3).
- 2) Traverse  $T'$  in post-order and associate a pair  $(\alpha_e, \beta_e)$  with each edge  $e$  in  $T'$ . Such pairs are computed in the following way:
  - a)  $(\alpha_e, \beta_e) = (1, 1)$  if  $e$  is a leaf-edge of  $T'$ , since its children nodes have been covered.
  - b) if an internal node  $v$  is being visited and it has a set of child-edges  $u_1, u_2, \dots, u_k$ , then each pair  $(\alpha_{u_j}, \beta_{u_j})$ ,  $j = 1, \dots, k$  has already been computed. Assume  $\alpha_u$  carries the number of different combinations of the child-edges of  $v$  for covering  $v$ , while  $\beta_u$  gives the number of combinations among the child-edges of  $v$  which do not cover  $v$ . The pair  $(\alpha_u, \beta_u)$ , which we assume represents an imaginary child-edge  $e_u$  of  $v$ , is computed as:

$$\alpha_u = \prod_{j=1}^k (\alpha_{u_j} + \beta_{u_j}) - \prod_{j=1}^k \beta_{u_j}; \quad \beta_u = \prod_{j=1}^k \beta_{u_j} \quad (2)$$

The pair associated to the father-edge

$e_v$  of  $v$  is computed as:

$$(\alpha_v, \beta_v) = \begin{cases} (\alpha_u + \beta_u, \alpha_u) & \text{if } v \text{ is free,} \\ (\alpha_u + \beta_u, \alpha_u + \beta_u) & \text{otw} \end{cases}$$

This step is iterated until it computes the pairs  $(\alpha_e, \beta_e)$  for all edge  $e \in T'$ . If there are more than one root-edges then one extra iteration of this step is applied in order to obtain a final pair  $(\alpha_{e_r}, \beta_{e_r})$  associated with just one root-edge  $e_r$ .

- 3)  $NE(T)$  is computed in accordance with the status of the root node  $v_r$  of  $T$ ;  $NE(T) = \alpha_{e_r} + \beta_{e_r}$  if  $v_r$  is a covered node, otherwise  $NE(T) = \alpha_{e_r}$ .

The above procedure returns  $NE(T)$  in time  $O(n + m)$  which is the necessary time for traversing  $T$  in post-order. Notice that this case includes the star topology network.

*Example 4:* Let  $T$  be the tree of figure 3a.  $T'$  is the reduced tree from  $T$  where its covered nodes are marked by a black point inside of the nodes (figure 3b). When  $T'$  is traversed in post-order a pair  $(\alpha_e, \beta_e)$  is associated with each edge. The pairs for the child-edges of  $v_r$ , are: (1,1), (4,3) and (6,3). Those three edges are combined in only one edge  $e_r$  applying

recurrence (2):  $\alpha_{e_r} = (1+1)*(4+3)*(6+3) - 1*3*3 = 117$  and  $\beta_{e_r} = 1*3*3 = 9$ . Since  $v_r$  is the root node and it is free, then  $NE(T) = \alpha_{e_r} = 117$ .

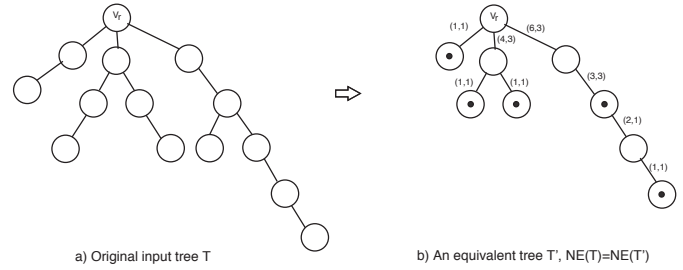


Fig. 2

COMPUTING THE NUMBER OF EDGE COVERS FOR A TREE

### Case C: The Ring Topology

Let  $C_n = (V, E)$  be a simple ring with  $n$  edges. We assume an order over the nodes and edges of  $C_n$  given by  $V = \{v_1, \dots, v_n\}$  and  $E = \{e_1, \dots, e_n\}$ ,  $e_i = \{v_i, v_{i+1}\}$ ,  $i = 1, \dots, n-1$ ,  $e_n = \{v_n, v_1\}$ . We call a *computing thread* or just a *thread* to the series  $(\alpha_1, \beta_1) \rightarrow (\alpha_2, \beta_2) \rightarrow \dots \rightarrow (\alpha_k, \beta_k)$  obtained by counting in an incremental way, applying the recurrence (1), the number of edge covers of a path with  $k$  edges.

Let  $L_p$  be the thread used for computing the series of pairs associated to the  $n$  edges of  $C_n$ . The pair  $(\alpha_1, \beta_1) = (1, 1)$  is associated with  $e_1$  since  $C_n$  has not fixed edges. Traversing in depth first search, the new pairs in  $L_p$  are computed applying the Fibonacci recurrence (1) since all nodes in  $C_n$  have degree two and they are free. After  $n$  applications of recurrence (1), the pair  $(\alpha_n, \beta_n) = (F_{n+1}, F_n)$  is obtained,  $F_i$  being the  $i$ -th Fibonacci number.

Let  $NC_n$  be the number of edge sets counted by  $L_p$ , i.e.  $NC_n = \alpha_n + \beta_n = F_{n+2}$ .  $L_p$  counted the edge sets where neither  $e_1$  nor  $e_n$  appear, since  $\beta_1 = 1$  and  $\beta_n > 0$ . Due to  $e_1$  or  $e_n$  or both have to be included in the edge cover sets of  $C_n$ , in order to cover  $v_1$ , we have to subtract from  $NC_n$  the number of sets which does not cover  $v_1$ .

Let  $Y$  be the number of edge sets which cover all nodes of  $C_n$  except  $v_1$ , then  $NE(C_n) = NC_n - Y$ . In order to compute  $Y$  a new thread  $L'_p = (\alpha'_1, \beta'_1) \rightarrow \dots \rightarrow (\alpha'_n, \beta'_n)$  is computed.  $L'_p$  begins with the pair  $(\alpha'_1, \beta'_1) = (0, 1)$ , i.e. it begins counting the edge sets where  $e_1$  does not appear. After  $n$  applications of recurrence (1) the last pair  $(\alpha'_n, \beta'_n)$  of  $L'_p$  obtains  $(F_{n-1}, F_{n-2})$ .

The number of edge sets where neither  $e_1$  nor  $e_n$  appear is  $\beta'_n = F_{n-2}$ , hence  $Y = F_{n-2}$ . Finally,  $NE(C_n) = NC_n - Y = F_{n+2} - F_{n-2}$ . Then, we deduce the following theorem.

*Theorem 5:* The number of edge cover sets of a simple cycle  $C_n$  with  $n$  edges, expressed in terms of Fibonacci numbers, is:  $NE(C_n) = F_{n+2} - F_{n-2}$ .

With  $\frown$  we denote the binary operation  $(\alpha_n, \beta_n - \beta'_n)$  between two pairs, and the result is associated with the last edge  $e_n$  of the ring  $C_n$  (fig. 4). Notice that the computation of

$NE(C_n)$  is the order  $O(n)$  since we compute the two threads:  $L_p$  and  $L'_p$  in parallel while the depth-first search is applied.

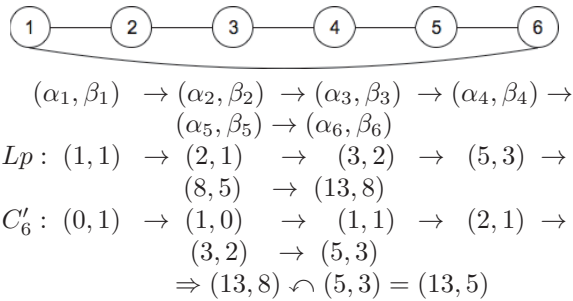


Fig. 3  
COUNTING THE NUMBER OF EDGES COVERS FOR A RING

*Example 6:* Let  $C_6$  be the ring illustrated in figure 4. Applying theorem (4.3), we have that  $NE(C_6) = F_{6+2} - F_{6-2} = F_8 - F_4 = 21 - 3 = 18$ .

The graphs which hold the topologies of the above cases (A) to (C) englobe the most common topologies of a communication network. The linear time procedures designed here can be included into a branch and bound algorithm which processes any kind of topology of a network.

### 4. Counting edge covers for acyclic graphs

In [6] methods to compute the number of edge covers for acyclic graphs and simple cycle graphs were presented. In this section we present an algorithm which combine those methods to compute the number of edge covers for graph without intersecting cycles, e.g. acyclic or with independent cycles graphs. The complexity of the method is polynomial with respect to the number of vertices of the graph.

Let  $G$  be a graph a directed depth first search graph  $T_G$  of  $G$  is built as follows:

- 1) Built a depth first search graph  $G'$  of  $G$  (it is well known that the edges of  $T_G$  are tree edges or back edges).
- 2) For each tree edge  $e = (u, v) \in G'$ , add the directed edge  $e = u \rightarrow v$  to  $T_G$  if  $u$  is a child of  $v$  in  $G'$ .
- 3) For each back edge  $e = (u, v) \in G'$  add the directed edge  $e = u \rightarrow v$  to  $T_G$  if  $u$  is a descendant of  $v$  in  $G'$ .

*Example 7:* Consider the graph of figure 4. It can be notice that it does not have intersecting cycles, just two independent cycles. A depth first search of  $G$  is shown at the left of figure 6. The dotted edges denote back edges and the solid edges denote tree edges. At the right hand side of figure 6 the directed depth first search graph of  $G$  is shown.

*Definition 8:* Let  $T_G$  be a directed depth first search graph and  $v \in V(T_G)$  we define:

$$input(u) = |\{v \mid v \rightarrow u \in E(T_G)\}|$$

$$output(u) = |\{v \mid u \rightarrow v \in E(T_G)\}|$$

*Definition 9:* Let  $G$  be a directed depth first search graph, and  $e = u \rightarrow v \in E(G)$ . A tuple  $(\alpha_e, \beta_e)$  is associated to  $e$

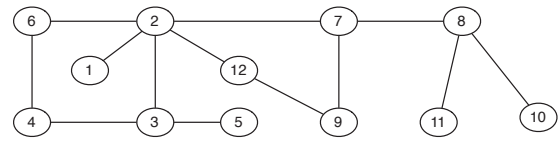


Fig. 4

A GRAPH WHICH DOES NOT HAVE INTERSECTING CYCLES

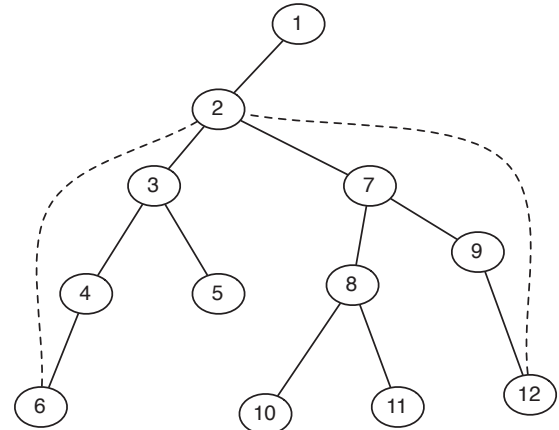


Fig. 5

A DEEP FIRST SEARCH OF THE GRAPH OF FIGURE 4. DOTTED EDGES REPRESENT BACK EDGES AND SOLID EDGES REPRESENT TREE EDGES.

such that  $\alpha_e$  represents the number of edge covers of  $G$  where  $e$  is considered to cover  $u$ . The number  $\beta_e$  represents the edge covers of  $G$  where  $e$  is not considered to cover  $u$ .

Algorithm 1 computes the number of edge covers of a directed depth first search graph  $T_G$  which does not have intersecting cycles. Lines 1 and 2 states the required inputs and output respectively. Line 3 states that a postorder traversal of the tree nodes of  $T_G$  is required. Line 4 states that a computation of a pair  $(\alpha_e, \beta_e)$  for each edge  $e$  during the traversal is required (see Definition 9). From line 6 to 24 the different types of edges in non-intersecting graphs are

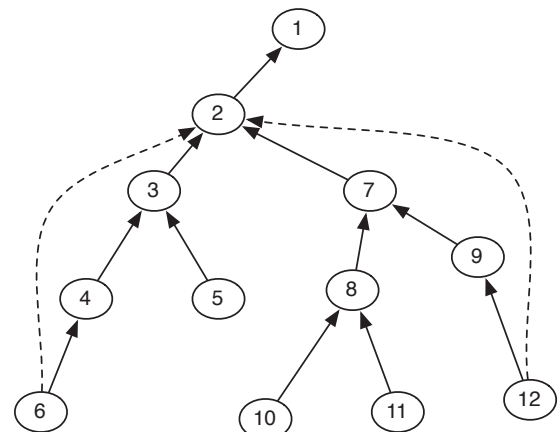


Fig. 6

A DIRECTED DEEP FIRST SEARCH OF THE GRAPH OF FIGURE 4

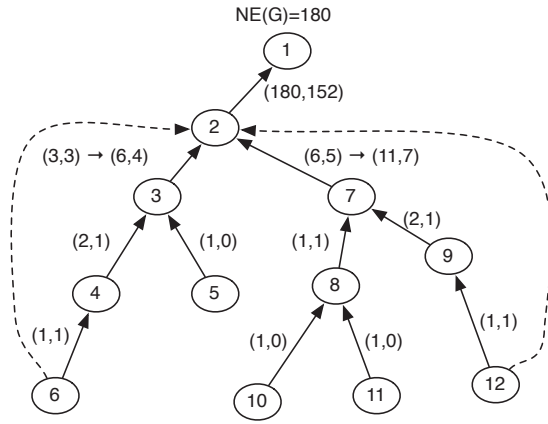


Fig. 7

COUNTING THE NUMBER OF EDGE COVERS ON THE DIRECTED DEPTH FIRST SEARCH OF THE GRAPH OF FIGURE 4.

considered. Line 6 represents the case where the edge  $e = u \rightarrow v$  must be present in each edge cover of  $T_G$  due to  $u$  is a leaf and there is not another edge to cover  $u$ , so a pair (1,0) is associated to these kind of edges. Line 9 represents the case where there are two output edges from  $u$ , let say  $e = u \rightarrow v$  and  $e_1 = u \rightarrow w$ . It can be notice that in a non-intersecting-cycle graph  $G$ ,  $output(v)$  is either 0, 1 or 2 for each node  $v \in G$ . That  $output(v) = 2$  means that one is a tree edge and the other a back edge (there are not two fathers for a child in a tree). In this case a pair (1,1) can be associated to  $e$  since removing  $e$  from an edge cover is valid iff  $e_1$  is contained. From line 12 to 15, it is said how to compute  $(\alpha_e, \beta_e)$  when besides that  $e = u \rightarrow v$  there is a back edge  $e_1 = x \rightarrow v$  (in other words a symple cycle is reached). A detail explanation is shown in [6]. From line 16 to 19 the formula to compute the pair  $(\alpha_e, \beta_e)$  for  $e = u \rightarrow v$  taking into account the pairs  $(\alpha_{e_i}, \beta_{e_i})$  where  $e_i = x_i \rightarrow u$  is described. Finally, lines 20 to 22 it is said how to compute edge covers when the root node is reached. Figure 7 shows the application of algorithm 1 to the directed depth first search graph of figure 6.

## 5. Conclusions

Sound and correct algorithms have been presented to compute the number of edge covers for graphs. It has been shown that if a graph  $G$  has simple topologies: paths, trees and simple cycles; then the number of edge covers can be computed in linear time over the graph size.

With respect to cyclic graphs that have intersecting cycles, a branch and bound procedure has been presented, it reduces the number of intersecting cycles until basic graphs are produced.

It has been determined a pair of recurrence relations that establish a bound on the time to compute the number of edge covers on intersecting cycle graphs. It was also designed the first "low-exponential" algorithm for the #Edge-Covers problem whose upper bound in the worst case is  $O(1.465571^{(m-n)} * (m+n))$ ,  $m$  and  $n$  being the number of

### Algorithm 1 Procedure edge cover for $G$ without intersecting cycles

- 1: Input:  $T_G$ : a directed depth-first search graph of  $G$  which do not contain intersecting cycles.
- 2: Output:  $NE(G)$ : the number of edge covers of  $G$ .
- 3: Traverse the nodes of  $G$  from the leaves to the root.
- 4: for each tree edge  $e_i = u \rightarrow v$  compute a pair  $(\alpha_{e_i}, \beta_{e_i})$  {back edges do not have a pair}
- 5: **switch** ( $e_i = u \rightarrow v$ )
- 6: **case**  $u$  is a leaf node and  $output(v) == 1$ :
- 7:  $(\alpha_{e_i}, \beta_{e_i}) = (1, 0)$ ; {there is no back edge from  $r$ }
- 8: **break**;
- 9: **case**  $u$  is a leaf node and  $output(v) == 2$ :
- 10:  $(\alpha_{e_i}, \beta_{e_i}) = (1, 1)$ ; {there is a back edge from  $r$ }
- 11: **break**;
- 12: **default**:
- 13: **if**  $e_j = x \rightarrow v$  is a back edge and  $path(e_i, e_j)$  is a cycle **then**
- 14:  $(\alpha_j, \beta_j) = (\alpha_j + \beta_j, \alpha_j + 1)$
- 15: **end if**
- 16: let  $A = \{e_1, e_2 \dots e_j\}$  be the set of edges such that  $e_k = x \rightarrow u$  for any  $x$ ,
- 17: **if**  $output(v) > 0$  and each par  $(\alpha_{e_j}, \beta_{e_j})$  for each edge of  $A$  has been computed **then**
- 18:  $(\alpha_{e_i}, \beta_{e_i}) = \left( \prod_{r=1}^j (\alpha_{e_r} + \beta_{e_r}), T \right)$ ; where  $T = \prod_{r=1}^j (\alpha_{e_r} + \beta_{e_r}) - \prod_{r=1}^j \beta_{e_r}$  { $s$  is not the root node}
- 19: **end if**
- 20: **if**  $output(v) == 0$  and and each par  $(\alpha_{e_j}, \beta_{e_j})$  for each edge of  $A$  has been computed **then**
- 21:  $(\alpha_i, \beta_i) = \left( \prod_{r=1}^j (\alpha_r + \beta_r) - \prod_{r=1}^j \beta_r, \prod_{r=1}^j \beta_r \right)$ ; { $s$  is the root node}
- 22: **return**  $\alpha_i$ ;
- 23: **end if**
- 24: **end switch**

edges and nodes of the input graph respectively. In terms only of the number of edges, our algorithm has an upper bound of  $O((1.324718)^m * (m+n))$ .

## References

- [1] Bubley R., Dyer M., Graph Orientations with No Sink and an Approximation for a Hard Case of #SAT, *Proc. of the Eight Annual ACM-SIAM Symp. on Discrete Algorithms*, 1997, pp. 248-257.
- [2] Bubley R., Dyer M., Greenhill C., Jerrum M., On approximately counting colourings of small degree graphs, *SIAM Jour. on Computing*, 29, (1999), pp. 387-400.
- [3] Bubley R., Randomized Algorithms: Approximation, Generation, and Counting, Distinguished dissertations Springer, 2001.
- [4] Darwiche Adnan, On the Tractability of Counting Theory Models and its Application to Belief Revision and Truth Maintenance, *Jour. of Applied Non-classical Logics*, 11 (1-2), (2001), 11-34.
- [5] Dahllöf V., Jonsson P., Wahlström M., Counting Satisfying Assignments in 2-SAT and 3-SAT, *LNCS 2387*, pp. 535-543, 2002.
- [6] De Ita G., Marcial-Romero J. Raymundo, Montes-Venegas Héctor., Counting the number of edge covers on common network topologies, *Electronic Notes in Discrete Mathematics*, Vol. 36, pp. 247-254, 2010.
- [7] Dyer M., Greenhill C., Some #P-completeness Proofs for Colourings and Independent Sets, Research Report Series, University of Leeds, 1997.

- [8] Garey M., Johnson D., *Computers and Intractability a Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.
- [9] Greenhill Catherine, "The complexity of counting colourings and independent sets in sparse graphs and hypergraphs", *Computational Complexity*, 9(1): 52-72, 2000.
- [10] Levit V.E., Mandrescu E., *The independence polynomial of a graph - a survey*, To appear, Holon Academic Inst. of Technology, 2005.
- [11] Roth D., "On the hardness of approximate reasoning", *Artificial Intelligence* 82, (1996), pp. 273-302.
- [12] Tarjan R., "Depth-First Search and Linear Graph Algorithms", *SIAM Journal on Computing*, Vol. 1, pp.146-160, 1972.
- [13] Vadhan Salil P., "The Complexity of Counting in Sparse, Regular, and Planar Graphs", *SIAM Journal on Computing*, Vol. 31, No.2, pp. 398-427, 2001.

# Feature Based Cryptanalytic Technique for Digital Forensics Analysis of Visual Cryptographic Digital Image Data Based on Formal Concept Analysis

Quist-Aphetsi Kester<sup>1,2,3</sup>, Laurent Nana<sup>1</sup>, Anca Christine Pascu<sup>1</sup>, Sophie Gire<sup>1</sup>, Jojo M. Eghan<sup>3</sup>, Nii Narku Quaynor<sup>3</sup>

<sup>1</sup> Lab-STICC (UMR CNRS 6285), European University of Brittany, University of Brest, France  
Kester.quist-aphetsi@univ-bret.fr / kquist@ieee.org

<sup>2</sup> Faculty of Informatics, Ghana Technology University College

<sup>3</sup> Department of Computer Science and Information Technology, University of Cape Coast

**Abstract** - *Lossless pixel value encrypted images still maintains the some properties of their respective original plain images. Ciphered Images that maintain the properties of their plain images of a given domain are very useful in certain applications where the conservation of pixel values but visual concealment is of a paramount concern. Medical images that have a fully reversible and recoverable process are of key importance in Medicine. Hence visually ciphered images stored or transmitted over secured or unsecured networks can also be analyzed in a forensic investigation to determine possible plain image equivalence. Digital Forensics processes have played crucial role in fighting crime both in society and cyberspace. In this paper, feature based cryptanalytic technique for digital forensics analysis of visual cryptographic digital image data based on formal concept analysis was proposed. Different techniques of visual cryptographic approaches were engaged in ciphering the plain image and our proposed approach was engaged in the cryptanalysis of the plain image after feature extractions from both the plain and the ciphered images. A lattice was generated which was then used authenticate and match the ciphered images to their respective ciphered plain images. At the end, the Galois lattice of both ciphered and plain image remained the same.*

**Keywords:** formal concept analysis, cryptanalysis, digital forensics, lattices, digital image, feature extraction, pixels

## 1 Introduction

The high increase in multimedia image usage for data communications over secured and unsecured network was due to the digitization of processes such as digital filing of documents, video conferencing, social media activities etc[1-3]. Secret communications between two parties using multimedia can also involve communications of encrypted image [4]. The rise in crime and availability of approaches to securing data to the general public has created avenues for people to implement cryptographic approaches in securing and concealing image contents. These approaches hinder

criminal investigative procedures and prevent easy analysis of digital evidences [5-7]. Cryptanalysis is an effective way of analyzing ciphers and encrypted data with the high hopes of decrypting the data or breaking the cipher. These approaches are very crucial in solving a range of issues in military communication applications and digital forensics toolkits. Cryptographers overtime have device the means of securing messages as well breaking codes [8-9].

Security in multimedia supplications is critical for the future. In this paper, we proposed a feature based cryptanalytic technique for digital forensics analysis of visual cryptographic digital image data based on formal concept analysis was proposed. Features were extracted from both plain and ciphered images and then lattices were built to help match plain images to their respected ciphered images. At the end, the Galois lattice of both ciphered and plain image remained the same. The paper has the following structure; section II Related works, section III is Methodology, section IV Results and analysis, and section V concluded the paper.

## 2 Literature Review

Forensics approaches cannot be effective in the presence of anti forensics procedures such as altering of content data during recovery process, incomplete evidence, encrypted data etc And as society has become increasingly reliant upon digital images to communicate visual information, a number of forensic techniques have been developed to verify the authenticity of digital images. Hence the digital forensics community requires new tools and strategies for the rapid turnaround of large forensic targets [10-13]. Alin C in their work described several statistical techniques for detecting traces of digital tampering in the absence of any digital watermark or signature. They quantify statistical correlations that result from specific forms of digital tampering, and devise detection schemes to reveal these correlations [14]. Dehnie, S proposed a digital image forensics for identifying computer generated and digital camera images [15]. Formal Concept analysis Formal is a field of applied mathematics based on the



mathematization of concept and conceptual hierarchy. It thereby activates mathematical thinking for conceptual data analysis and knowledge processing [16]. Its applications in forensics are normally in the domain of computer aided investigations. Where the data collected on crime re being analyzed using the approach[17-18]. In our approach we engaged feature based cryptanalytic technique for digital forensics analysis of visual cryptographic digital image data based on formal concept analysis was proposed. Different techniques of visual cryptographic approaches were engaged in ciphering the plain image and our proposed approach was engaged in the cryptanalysis of the plain image after feature extractions from both the plain and the ciphered images.

### 3 Methodology

Our method employed a cryptanalytic procedure by using features generated from digital images which were then used to construct a Galois lattice. The features were extracted in such a way that a change in pixel value can cause a change in concept of the lattice. This means that if there is no pixel expansion in the ciphering process of the image, a perfect match of its plain image can be obtained by using our proposed method. The overall process is indicated in figure 1 below.

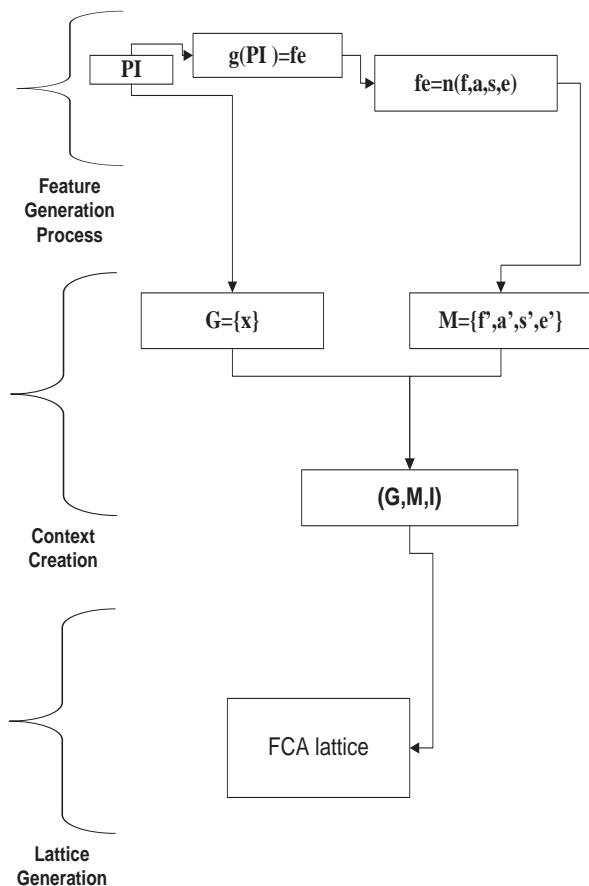


Figure 1: Summary of the Entire process

From figure 1:

PI=Plain image

g(PI)=function that operated on the plain image to produce the features

n(f,a,s,e)=function of the features

fe= the feature results

f=sum of all frequency of each pixel in the image

a=arithmetic mean of all the pixel values in the image

s=standard deviation all the pixel value in the image

e=entropy of all the pixel value the image

x=a distinct chosen pixel value number

x'=frequency of x

f'=x'/f, a'=x'/a, s'=x'/s and e'=x'/e

G =set objects extracted from the image

M=sets attributes obtained from the image

Concepts obtained are (G,M,I)

r(G,M,I)=the function that operated on G,M and I concept to produce K

ImC=the image encryption algorithm that operated on K and Pi to produce CI

#### 3.1 The Feature Extraction

Let I= an image=f (R, G, B)

I is a color image of m x n x 3 arrays

$$\begin{pmatrix} R & G & B \\ r_{i1} & g_{i2} & b_{i3} \\ \vdots & \vdots & \vdots \\ r_{n1} & g_{n2} & b_{n3} \end{pmatrix}$$

(R, G, B) = m x n and R, G, B ∈ I

(R o G) ij = (R) ij . (G) ij

where r<sub>i1</sub> = first value of R

r = [ri1] (i=1, 2... m) and

x ∈ r<sub>i1</sub> : [a, b]= {x ∈ I: a ≤ x ≤ b}

a=0, b=255 and R= r= I (m, n, 1)

where g<sub>i2</sub> = first value of G

g = [gi2] (i=1, 2... m) and

x ∈ g<sub>i1</sub> : [a, b]= {x ∈ I: a ≤ x ≤ b}

a=0, b=255 and G= g= I (m, n, 1)

and b<sub>i3</sub> = first value of B

g = [bi3] (i=1, 2... m) and

x ∈ b<sub>i1</sub> : [a, b]= {x ∈ I: a ≤ x ≤ b}

a=0, b=255 and B=b= I (m, n, 1)

Such that R= r= I (m, n, 1)

Let X=freq(x) which is the number of times x occurred in r,g and b

$$f = \sum_{i=m}^n X_i$$

$$a = \frac{\sum_{n=1}^k x_n}{k}$$

Where  $x \in b\_i1 : [a, b] = \{x \in I : a \leq x \leq b\}$

$$s = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}, \text{ where } \mu = \frac{1}{N} \sum_{i=1}^N x_i.$$

Entropy is defined as

$$e = -\sum_{\eta=0}^{\varepsilon-1} \Psi(\eta) \cdot \log_2(\Psi(\eta))$$

Where:

$\delta$  = Entropy of image

$\varepsilon$  = Gray value of an input image (0-255).

$\Psi(\eta)$  = Probability of the occurrence of symbol  $\eta$

### 3.2 The Feature Classification using FCA

Formal concept analysis (FCA) is a method of data analysis with growing popularity across various domains. FCA analyzes data which describe relationship between a particular set of objects and a particular set of attributes.

If  $g \in A$  and  $m \in B$  then  $(g, m) \in I$ , or  $gIm$ .

A formal context is a triple  $(G, M, I)$ , where

- $G$  is a set of objects,
- $M$  is a set of attributes
- and  $I$  is a relation between  $G$  and  $M$ .
- $(g, m) \in I$  is read as „object  $g$  has attribute  $m$ “.

For  $A \subseteq G$ , we define

$$A' := \{m \in M \mid \forall g \in A: (g, m) \in I\}.$$

For  $B \subseteq M$ , we define dually

$$B' := \{g \in G \mid \forall m \in B: (g, m) \in I\}.$$

For  $A, A1, A2 \subseteq G$  holds:

- $A1 \subseteq A2 \Rightarrow A'2 \subseteq A'1$
- $A \subseteq A''$
- $A' = A'''$

For  $B, B1, B2 \subseteq M$  holds:

- $B1 \subseteq B2 \Rightarrow B'2 \subseteq B'1$
- $B \subseteq B''$
- $B' = B'''$

A formal concept is a pair  $(A, B)$  where

- $A$  is a set of objects (the extent of the concept),
- $B$  is a set of attributes (the intent of the concept),
- $A' = B$  and  $B' = A$ .

The concept lattice of a formal context  $(G, M, I)$  is the set of all formal concepts of  $(G, M, I)$ , together with the partial Order  $(A1, B1) \leq (A2, B2): \Leftrightarrow A1 \subseteq A2 (\Leftrightarrow B1 \supseteq B2)$  (Priss, U, 1997).

The concept lattice is denoted by  $\mathfrak{B}(G, M, I)$ .

• Theorem: The concept lattice is a lattice, i.e. for two concepts

$(A1, B1)$  and  $(A2, B2)$ , there is always

- a greatest common sub-concept:  $(A1 \cap A2, (B1 \cup B2)')$
- and a least common super-concept:  $((A1 \cup A2)', B1 \cap B2)$

More general, it is even a complete lattice, i.e. the greatest common sub-concept and the least common super-concept exist for all (finite and infinite) sets of concepts.

Corollary: The set of all concept intents of a formal context is a closure system. The corresponding closure operator is  $h(X) := X''$ .

An implication  $X \rightarrow Y$  holds in a context, if every object having all attributes in  $X$  also has all attributes in  $Y$ .

Def.: Let  $X \subseteq M$ . The attributes in  $X$  are independent, if there are no trivial dependencies between them

|       | $y_1$ | $y_2$ | $y_3$ | ... |
|-------|-------|-------|-------|-----|
| $x_1$ | ×     | ×     | ×     |     |
| $x_2$ | ×     | ×     |       | ⋮   |
| $x_3$ |       | ×     | ×     |     |
| ⋮     |       | ...   |       | ⋮   |

Figure 2: A table of attributes and properties

The table above represents logical attributes represented by a triplet  $(X, Y, I)$ , where  $I$  is a binary relation between  $X$  and  $Y$ . The elements of  $X$  are called objects and correspond to table rows, elements of  $Y$  are called attributes and correspond to table columns, and for  $x \in X$  and  $y \in Y$ ,  $(x, y) \in I$  indicates that object  $x$  has attribute  $y$  while  $(x, y) \in I$ .

From the image we chose our objects as a classified range of values of  $x$ , where  $x \in b\_i1 : [a, b] = \{x \in I : a \leq x \leq b\}$  and  $a=0, b=255$ .  $G = \{0-25, 26-50, 51-75, 76-100, 101-125, 126-150, 151-175, 176-200, 201-225, 256-255\}$ .

$$f' = j/f$$

$$a' = j/a$$

$$s' = j/s$$

$$e' = j/e$$

where  $j$  is the sum of all the frequencies of all numbers that fall within the range of each object.

Where  $B = \{f, a', s', e'\}$  Major attributes and  $f'$  has  $\{B\_1, B\_2, B\_3$  and  $B\_4\}$  as sub attributes. Therefore,  $a', s'$  and  $e'$  have the same sub attributes as  $f'$ . But  $\{B\_1, B\_2, B\_3, B\_4\}$  maps directly and exactly on at least one element of  $\{0-0.25, 0.26-0.50, 0.51-0.75, 0.76-1.0\}$ .

## 4 Analysis and Results

We chose a 24 bit depth image jpg of dimension 960 pixels by 720 pixels with a horizontal resolution of 96 dpi and a vertical resolution of 96 dpi.



Figure 3: Plain image

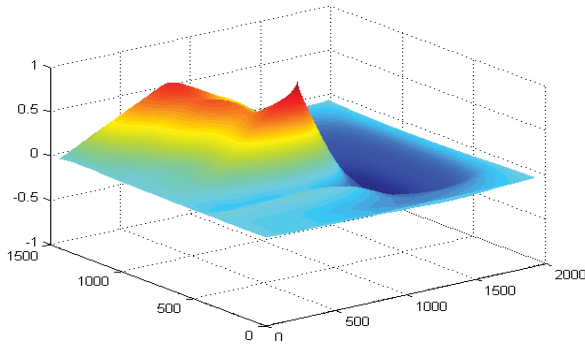


Figure 4: The graph of the normalized cross-correlation of the matrices of the plain image

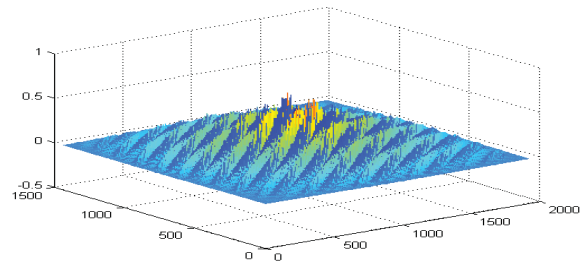


Figure 8: The graph of the normalized cross-correlation of the matrices of the ciphered image

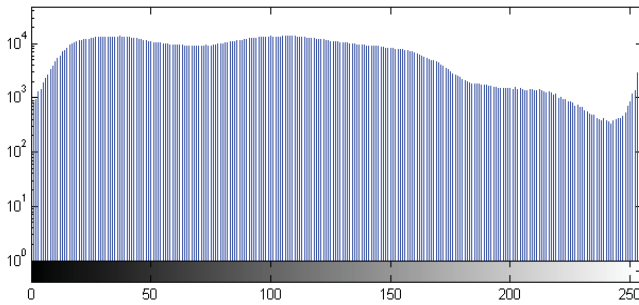


Figure 5: A graph of frequency of pixel values

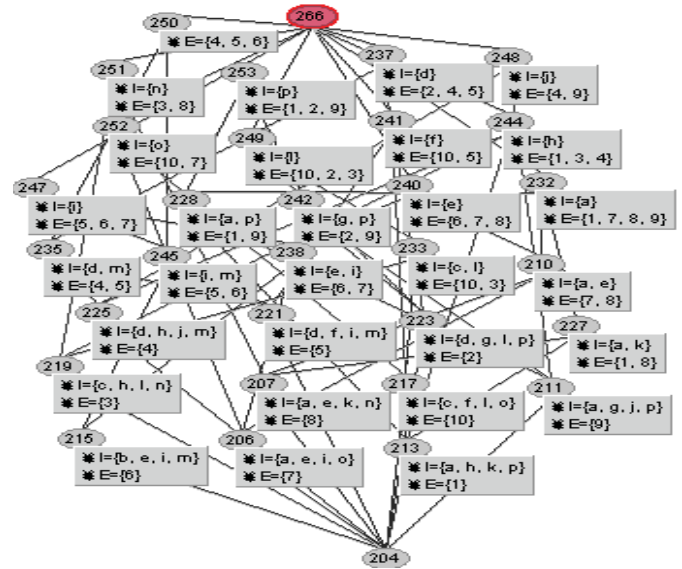


Figure 9: A Galois lattice generated from the ciphered image

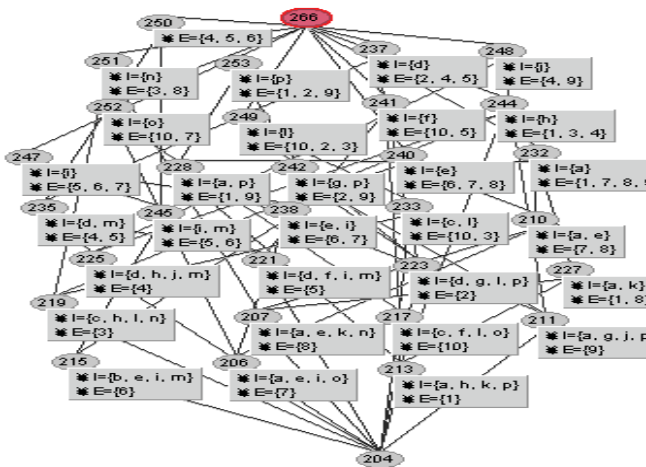


Figure 6: A Galois lattice generated from the plain image

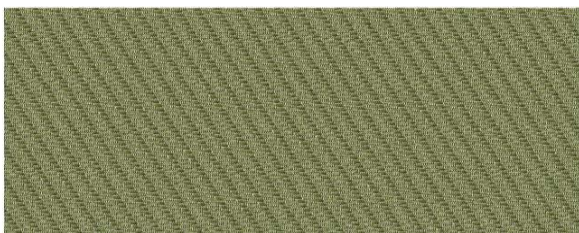


Figure 7: The ciphered image

Table 1: Table Objects X and attributes

| G       | j          | f'       | a'       | s'       |
|---------|------------|----------|----------|----------|
| 0-25    | 16993<br>8 | 0.081953 | 1502.998 | 2343.901 |
| 26-50   | 31651<br>7 | 0.152641 | 2799.4   | 4365.619 |
| 51-75   | 23576<br>4 | 0.113698 | 2085.189 | 3251.819 |
| 76-100  | 28914<br>9 | 0.139443 | 2557.347 | 3988.141 |
| 101-125 | 31944<br>6 | 0.154054 | 2825.306 | 4406.018 |
| 126-150 | 23575<br>8 | 0.113695 | 2085.136 | 3251.736 |
| 151-175 | 14600<br>1 | 0.070409 | 1291.29  | 2013.746 |
| 176-200 | 46113      | 0.022238 | 407.8414 | 636.0221 |
| 201-225 | 31294      | 0.015092 | 276.7764 | 431.6283 |
| 226-255 | 28362<br>0 | 0.136777 | 2508.446 | 3911.881 |

The graph of the normalized cross-correlation of the matrices of the plain image in figure 3 was plotted as shown in figure 4. The features  $f=2073600$ ,  $a=113.066$ ,  $s=72.5022$  and  $e=7.1945$  were extracted from the plain images. The frequencies of the pixel values of the plain image were plotted as shown in figure 5 and a Galois lattice was generated from the features extracted from the plain image based on table 1. Table two below showed six different techniques of visual cryptography applies to the image. The results for the features were constant even though the visual states of the encrypted images differ for the various approaches engaged.

Table 2: Table Objects X and attributes

| Approaches | f       | a       | s       | e      |
|------------|---------|---------|---------|--------|
| 1          | 2073600 | 113.066 | 72.5022 | 7.1945 |
| 2          | 2073600 | 113.066 | 72.5022 | 7.1945 |
| 3          | 2073600 | 113.066 | 72.5022 | 7.1945 |
| 5          | 2073600 | 113.066 | 72.5022 | 7.1945 |
| 6          | 2073600 | 113.066 | 72.5022 | 7.1945 |

At the end all the graphs plotted and the concept lattices were the same for both the ciphered and the plain images. A set of encrypted images were tested against their corresponding ciphered images and the results were effective. This means that a plain image can be mapped directly to its corresponding ciphered image without decrypting the ciphered image for a given data set.

## 5 Conclusion

Based on the extracted features from both the plain and the ciphered images, a Galois lattice was constructed. We have realized that the Galois lattice generated from the plain image as well as the features extracted from the plain image was the same as that of the ciphered image irrespective of pixel displacement that occurred. This was as a result of conservation of pixel values. This makes our approach a suitable forensics analysis of encrypted images based on visual cryptography or pixel displacement. Our results were very effective for different kind of approaches that engaged a non pixel expansion technique in ciphering the image. Our proposed method can help also in the indexing of images based on the extracted features and can help in evidence analysis of ciphered images based on visual cryptographic techniques that conserves pixel values.

**Acknowledgments.** This work was supported by Lab-STICC (UMR CNRS 6285) at UBO France, AWBC Canada, Ambassade de France-Institut Français-Ghana and the DCSIT-UCC, and also Dominique Sotteau (formerly directeur de recherche, Centre national de la recherche scientifique (CNRS) in France and head of international relations, Institut national de recherche en informatique et automatique, INRIA) and currently the Scientific counselor of AWBC.

## References

- [1] Yadav, R., Gupta, R. K., & Singh, A. P. (2015). A Survey of Image Compression using Neural Network and Wavelet Transformation. *International Journal of Research*, 2(1), 301-305.
- [2] Yen, N. Y., Zhang, C., Waluyo, A. B., & Park, J. J. (2015). *Social Media Services and Technologies Towards Web 3.0. Multimedia Tools and Applications*, 1-7.
- [3] Roy, S. D., & Zeng, W. (2015). Revelations from Social Multimedia Data. In *Social Multimedia Signals* (pp. 135-142). Springer International Publishing.
- [4] Nguyen, K. T., Laurent, M., & Oualha, N. (2015). Survey on secure communication protocols for the Internet of Things. *Ad Hoc Networks*.
- [5] Hashem, F. S., & Sulong, G. (2015). Passive Approaches for Detecting Image Tampering: A Review. *Jurnal Teknologi*, 73(2).
- [6] Cao, Y., Gao, T., Sheng, G., Fan, L., & Gao, L. (2015). A New Anti - forensic Scheme- Hiding the Single JPEG Compression Trace for Digital Image. *Journal of forensic sciences*, 60(1), 197-205.
- [7] Wang, X. G. (2015, January). Research on digital forensics and its relevant problems. In *Electronics, Information Technology and Intellectualization: Proceedings of the International Conference EITI 2014, Shenzhen, 16-17 August 2014* (p. 43). CRC Press.
- [8] Yap, W. S., Phan, R. C. W., Yau, W. C., & Heng, S. H. (2015). Cryptanalysis of a new image alternate encryption algorithm based on chaotic map. *Nonlinear Dynamics*, 80(3), 1483-1491.
- [9] Ahmad, M., Khan, I. R., & Alam, S. (2015, January). Cryptanalysis of Image Encryption Algorithm Based on Fractional-Order Lorenz-Like Chaotic System. In *Emerging ICT for Bridging the Future-Proceedings of the 49th Annual Convention of the Computer Society of India CSI Volume 2* (pp. 381-388). Springer International Publishing.
- [10] Kessler, G. C. (2007, March). Anti-forensics and the digital investigator. In *Australian Digital Forensics Conference* (p. 1).
- [11] Harris, R. (2006). Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. *digital investigation*, 3, 44-49.
- [12] Stamm, M. C., & Liu, K. R. (2011). Anti-forensics of digital image compression. *Information Forensics and Security, IEEE Transactions on*, 6(3), 1050-1065.
- [13] Richard III, G. G., & Roussev, V. (2006). Next-generation digital forensics. *Communications of the ACM*, 49(2), 76-80.
- [14] Popescu, A. C., & Farid, H. (2005, January). Statistical tools for digital forensics. In *Information Hiding* (pp. 128-147). Springer Berlin Heidelberg.
- [15] Dehnie, S. (2006, October). Digital image forensics for identifying computer generated and digital camera images. In *Image Processing, 2006 IEEE International Conference on* (pp. 2313-2316). IEEE.
- [16] Ganter, B., & Wille, R. (2012). *Formal concept analysis: mathematical foundations*. Springer Science & Business Media.
- [17] Kester, Q. A. (2013). *Criminal Geographical Profiling: Using FCA for Visualization and Analysis of Crime Data*. arXiv preprint arXiv:1310.0864.
- [18] Kester, Q. A. (2013). Visualization and analysis of geographical crime patterns using formal concept analysis. arXiv preprint arXiv:1307.8112.
- [19] Poelmans, J., Elzinga, P., Dedene, G., Viaene, S., & Kuznetsov, S. O. (2011). A concept discovery approach for fighting human trafficking and forced prostitution. In *Conceptual Structures for Discovering Knowledge* (pp. 201-214). Springer Berlin Heidelberg.

# Quantum codes derived from generalized Hadamard matrices

Cekad Sarami

Department of Mathematics and Computer Science, Fayetteville State University, Fayetteville, NC, USA

**Abstract**—This paper discusses constructions of quaternary Hermitian self-orthogonal codes and related quantum codes obtained from Kronecker products of generalized Hadamard matrices over to elementary Abelian group of order 4. We have shown that the codes generated by these matrices yeild large single error-correcting and single error-detecting quantum codes.

## 1. Introduction

A generalized Hadamard matrix  $GH(\mu, G) = (h_{ij})$  over a group  $G$  of order  $g$  is a  $g\mu \times g\mu$  matrix with entries from  $G$  with the property that for every  $i, j, 1 \leq i < j \leq g\mu$ , the multi-set  $\{h_{is}h_{js}^{-1} | 1 \leq s \leq g\mu\}$  contains every element of  $G$  exactly  $\mu$  times. If  $H_1 = GH(\mu_1, G)$  and  $H_2 = GH(\mu_2, G)$  are generalized Hadamard matrices over the group  $G$  of order  $g$ , their Kronecker product  $H_1 \otimes H_2$  is a  $g^2\mu_1\mu_2 \times g^2\mu_1\mu_2$  generalized Hadamard matrix over  $G$ .

A Generalized Hadamard matrix  $H$  is normalized with respect to row  $i$  and column  $j$  if all entries in the  $i$ th row and  $j$ th row of  $H$  are equal to the unit element of  $G$ . Two generalized Hadamard matrices of the same order over a group  $G$  are said to be equivalent if one cab be obtained from the other by permutations of rows and columns, and multiplications of rows and columns with elements from  $G$ .

In this paper, we consider generalized Hadamard matrices over the Abelian group of order 4,

$$EA(4) = \{1, a, b, ab | a^2 = b^2 = (ab)^2 = 1, ab = ba\}.$$

For each of the orders 4, 8, and 12 there is one equivalence class of generalized Hadamard matrices [10]. In [5], Harada, Lam, and Tonchev enumerated all generalized Hadamard matrices of 16 over  $EA(4)$ , and studied the linear codes of length 15 over the finite field  $F_4 = \{0, 1, w, w^2\}$  spanned by the rows of  $15 \times 15$  matrices obtained from normalized generalized Hadamard matrices of order 16 by deleting the constant row and constant column and replacing the elements of  $EA(4)$  with the elements of the additive group of  $F_4, 1 \leftrightarrow 0, a \leftrightarrow 1, b \leftrightarrow w, ab \leftrightarrow w^2$ , and replacing multiplication in  $EA(4)$  by the addition in  $F_4$ . Many of the resulting codes turned out to be self-orthogonal with respect to the Hermitian inner product:

$$(x, y) = \sum_{i=1}^n x_i y_i^2,$$

where,  $x = (x_1, \dots, x_n)$  and  $y = (y_1, \dots, y_n) \in F_4$ , hence yield quantum error-correcting codes via a known construction due to Calderbank, Rains, Shor, and Sloane [3]:

*Proposition 1:* (Shor, Calderbank, Rains, and Sloane [3]). A linear Hermitian self-orthogonal  $F_4$ -code  $C$  of length  $n$  with dimension  $k$  and dual distance  $d^\perp$  (where  $C^\perp$  is the Hermitian dual code of  $C$ ) yields a quantum error-correcting code with parameters  $[[n, n - 2k, d^\perp]]$ .

The theory of quantum error-correcting codes is a well-known topic. We refer readers not familiar with the topic to a thorough discussion of the principles of quantum error-correcting codes given in [3]. In the aforementioned paper, many examples of quantum codes are given, together with tabulation of codes and bounds on the minimum distance for length up to 30 quantum bits. We use the notation  $[[n, k, d]]$  to refer to a quantum error-correcting code for  $n$  quantum bits having  $2^k$  codewords and minimum distance  $d$ . In 1996, Calderbandk and Shor showed existence of good quantum codes [2]. In [11], Steane extended their work and constructed quantum codes using binary BCH-codes and extended BCH-codes. Quantum codes can be build using combinatorial structures in various ways. In [9], Ruihu Li and Xueliang Li construct quantum codes using Steane construction of quantum codes [11]. In [6], Kim and Walker offer construction of nonbinary quantum codes with various length, dimensions, and minimum distances from algebraic curves. In [7], Guo, Ma, and Feng construct quantum codes from self-dual codes and maximal self-dual codes over  $F_5$ . In [4], Clark and Tonchev uses finite geometry over  $F_q$  to construct  $q$ -ary quantum codes. In [6], Fugiwara and Vandendriessche construct quantum synchronizable codes via cyclic codes and finite geometries. In this paper, we summarize the results of the computation of quaternary Hermitian self-orthogonal codes and the corresponding quantum codes derived from generalized Hadamard matrices over  $EA(4)$  of orders 64, 128, and 192 obtained from Kronecker products of the unique matrices of order 4, 8 and 12 with each matrices of order 16 of symmetric nets introduced in [5]

## 2. Kronecker products of generalized Hadamard matrices and their quantum codes

The generalized Hadamard matrices of order 4,8, and 12 over  $EA(4)$  are unique up to equivalence [10]. Classification

of generalized Hadamard matrices of order larger than 20 is infeasible for us due two large number of them. This motivated us to compute the Kronecker sum of the unique matrices of order 4, 8 and 12 by 226 generalized Hadamard matrices (computed in [5]). In the following sections, we have shown that the codes generated by these matrices yield single error-correcting and single error-detecting codes. The generalized Hadamard matrices of order 16 over EA(4), are presented in [5] by means of  $64 \times 64$  (0,1)-incidence matrices of 226 combinatorial structures known as(4,4)-nets. The incidence matrices of (4,4)-nets No.  $i$  ( $i = 1, \dots, 226$ ) are available at

<http://www.math.mtu.edu/~tonchev/Z2Z2nets>.

We can obtain 226 generalized Hadamard matrices using the following method. The  $64 \times 64$  (0,1)-incidence matrix  $A$  of a (4,4)-net with a group of automorphism being an elementary Abelian group of order four, i.e.  $EA(4)$  is a block matrix of  $4 \times 4$  submatrices

$$I = I_4 \quad A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

These matrices form a multiplicative Abelian group

$$G = (\{I, A, B, C = AB\}$$

$$A^2 = B^2 = (AB)^2 = I, AB = BA, \cdot)$$

which is isomorphic to Abelian group of the finite field  $F_4$ . we replace these these block matrices with the elements of  $F_4 = \{0, 1, w, w^2\}$ , as follows:

$$I \leftrightarrow 0, A \leftrightarrow 1, B \leftrightarrow w, C \leftrightarrow w^2$$

to get a GH(4,4). Let's denote the normalized generalized Hadamard matrix that corresponds to the net No. $i$  ( $i = 1, \dots, 226$ ) by  $T_i$ . Let  $U_i, V_i$ , and  $W_i$ , be the Kronecker sum of generalized Hadamard matrices  $H_4, H_8$ , and  $H_{12}$  with  $T_i$  ( $i = 1, \dots, 226$ ), respectively as follows:

$$U_i = H_4 \oplus T_i$$

$$V_i = H_8 \oplus T_i$$

$$W_i = H_{12} \oplus T_i$$

where  $1 \leq i \leq 226$ . Deleting the first all-zero row and column from  $U_i, V_i$ , and  $W_i$  give 63 by 63, 127 by 127, and 191 by 191 matrices  $U'_i, V'_i$  and  $W'_i$ , respectively. We denote the  $F_4$ -codes generated by the columns of the matrices  $U'_i, V'_i$  and  $W'_i$  by  $A_i, B_i$ , and  $C_i$ , respectively. In the following

Table 1: Quantum codes obtained from codes  $A_i$

| Code Number   | Parameters    |
|---|---------------|
| 1   | [[63, 53, 2]] |
| 2, 3, 5, 6, 9, 10, 20, 26, 34, 36   | [[63, 55, 2]] |
| 4, 7, 8, 11, 12, 14, 15, 16, 19, 21, 22, 23, 24, 25, 27<br>28, 30, 31, 33, 35, 37, 38, 40, 42, 43, 44, 46, 47, 49<br>83, 84, 50, 51, 52, 65, 68, 69, 75, 76, 77, 78, 79, 80, 81<br>99, 100, 82, 85, 86, 87, 88, 91, 92, 93, 94, 95, 96, 97<br>98, 103, 104, 105, 107, 111, 113, 120, 121, 127, 128<br>131, 142, 143, 144, 145, 146, 147, 148, 149, 150<br>151, 181, 182, 192, 193, 194, 195, 196, 197, 198<br>211, 212, 213, 215, 218, 219, 220, 221, 224, 225, 226 | [[63, 53, 2]] |
| 13, 17, 18, 29, 32, 39, 41, 45, 48, 53, 54, 55, 56, 57, 66<br>67, 70, 71, 72, 73, 74, 89, 90, 101, 102<br>106, 108, 109, 110, 112, 114, 115, 116, 122, 129, 130<br>137, 138, 139, 140, 141, 152, 157, 158, 167, 168, 176<br>183, 184, 185, 187, 188, 190, 203   | [[63, 51, 2]] |
| 58, 59, 60, 61, 62, 63, 64, 117, 118, 119, 123, 124, 125<br>126, 132, 133, 134, 135, 136, 153, 154, 155, 159, 160<br>161, 162, 163, 164, 165, 166, 169, 170, 171, 172, 173<br>174, 175, 177, 178, 179, 180, 189, 191, 199, 200, 201<br>202, 204, 206, 207, 208, 209, 214, 216, 217, 222, 223<br>156, 205, 210   | [[63, 49, 2]] |
|   | [[63, 47, 2]] |

subsections we present the quantum codes obtained by these  $F_4$ -codes.

## 2.1 Quantum codes obtained from $H_4 \oplus T_i$

Let codes  $A_i$  be the  $F_4$ -codes generated by the columns of the matrices  $U'_i$ . All 226 codes  $A_i$  are both self-orthogonal and Hermitian self-orthogonal. Therefore, They yield quantum codes of length  $n = 63$  and  $d = 2$  by Proposition 1. We have computed the parameters of these quantum codes using Magma[1] (Table 1).

## 2.2 Quantum codes obtained from $H_8 \oplus T_i$

Let codes  $B_i$  be the codes generated by the columns of the matrices  $V'_i$ . All 226 codes  $B_i$  are both self-orthogonal and Hermitian self-orthogonal. This is easily verified with computer. They yield quantum codes of length  $n = 127$  and  $d = 2, 3$  by Proposition 1. We have computed the parameters of these quantum codes using Magma (Table 2).

## 2.3 Quantum codes obtained from $H_{12} \oplus T_i$

Let codes  $C_i$  be the codes generated by the columns of the matrices  $W'_i$ . All 226 codes  $C_i$  are both self-orthogonal and Hermitian self-orthogonal. This is easily verified with computer. They yield quantum codes of length  $n = 191$  and  $d = 2$  by Proposition 1. We have computed the parameters of these quantum codes using Magma (Table 3).

## 2.4 Acknowledgement

This research was partially supported by NSF grant, HBCU-UP #1036257.

Table 2: Quantum codes obtained from codes  $B_i$

| Code Number  | Parameters      |
|--|-----------------|
| 1  | [[127, 117, 2]] |
| 2, 3, 5, 6, 9, 10, 20, 26, 34, 36  | [[127, 115, 2]] |
| 4, 7, 8, 11, 12, 14, 15, 16, 19, 21, 22, 23, 24, 25, 27<br>28, 30, 31, 33, 35, 37, 38, 40, 42, 43, 44, 46, 47, 49<br>50, 51, 52, 65, 68, 69, 75, 76, 77, 78, 79, 80, 81, 82,<br>83, 84, 85, 86, 87, 88, 91, 92, 93, 94, 95, 96, 97, 98                           | [[127, 113, 2]] |
| 13, 17, 18, 29, 32, 39, 41, 45, 48, 53, 54, 55, 56<br>57, 66, 67, 70, 71, 72, 73, 74, 89, 90   | [[127, 111, 2]] |
| 58, 59, 60, 61, 62, 63, 64   | [[127, 109, 2]] |
| 99, 100, 103, 104, 105, 107, 111, 113, 120, 121, 127<br>128, 131, 142, 143, 144, 145, 146, 147, 148, 149, 150<br>151, 181, 182, 192, 193, 194, 195, 196, 197, 198, 211<br>212, 213, 215, 218, 219, 220, 221, 224, 225, 226                                       | [[127, 113, 3]] |
| 101, 102, 106, 108, 109, 110, 112, 114, 115, 116<br>122, 129, 130, 137, 138, 139, 140, 141, 152, 157, 158<br>167, 168, 176, 183, 184, 185, 186, 187, 188, 190, 203   | [[127, 111, 3]] |
| 117, 118, 119, 123, 124, 125, 126, 132, 133, 134<br>135, 136, 153, 154, 155, 159, 160, 161, 162, 163<br>164, 165, 166, 169, 170, 171, 172, 173, 174, 175<br>177, 178, 179, 180, 189, 191, 199, 200, 201, 202<br>204, 206, 207, 208, 209, 214, 216, 217, 222, 223 | [[127, 109, 3]] |
| 156, 205, 210  | [[127, 107, 3]] |

Table 3: Quantum codes obtained from codes  $C_i$

| Code Number   | Parameters      |
|---|-----------------|
| 1   | [[191, 177, 2]] |
| 2, 3, 5, 6, 9, 10, 20, 26, 34, 36   | [[191, 175, 2]] |
| 4, 7, 8, 11, 12, 14, 15, 16, 19, 21, 22, 23, 24,25,27<br>28, 30, 31, 33, 35, 37, 38, 40, 42, 43, 44, 46, 47,49<br>50, 51, 52, 65, 68, 69, 75, 76, 77, 78, 79, 80, 81,82<br>83,84, 85, 86, 87, 88, 91, 92, 93, 94, 95, 96, 97, 98<br>99, 100,103, 104,105, 107, 111,113,120,121,127<br>128, 131,105, 107, 111, 113, 120, 121, 127, 128<br>131,182, 192, 193, 194, 195, 196, 197, 198,211<br>212, 213, 215, 218, 219, 220, 221, 224, 225, 226 | [[191, 173, 2]] |
| 13, 17, 18, 29, 32, 39, 41, 45, 48, 53, 54, 55, 56,57<br>66, 67, 70, 71, 72, 73, 74, 89, 90, 101, 102,106,108<br>109, 110,112, 114, 115, 116, 122, 129, 130, 137<br>138, 139, 140, 141,152, 157, 158, 167, 168<br>176, 183, 184, 185, 186, 187, 188, 190, 203   | [[191, 171, 2]] |
| 58, 59, 60, 61, 62, 63, 64, 117, 118, 119, 123  | [[191, 169, 2]] |
| 124, 125, 126, 132, 133, 134, 135, 136<br>153, 154, 155, 159, 160,161, 162, 163, 164<br>165, 166, 169, 170, 171, 172, 173, 174, 175<br>177, 178, 179, 180, 189, 191, 199, 200, 201,202<br>204, 206, 207, 208, 209, 214, 216, 217, 222, 223  |                 |
| 156, 205, 210   | [[191, 167, 2]] |

## References

- [1] W. Bosma, J. Cannon and C. Playoust. "The Magma algebra system. I. The user language", *J. Symbolic Comput.*, 24 (1997), 235–265
- [2] A. R. Calderbank, and P. W. Shor. "Good quantum error-correcting codes exist." *Physical Review A* 54.2 (1996): 1098.
- [3] A.R. Calderbank, et al. "Quantum error correction via codes over GF (4)." *Information Theory. 1997. Proceedings., 1997 IEEE International Symposium on.* IEEE.
- [4] D. Clark, and V. D. Tonchev. "Nonbinary quantum codes derived from finite geometries." *Finite Fields and Their Applications* 18.1 (2012): 63-69.
- [5] M. Harada, C. Lam, and V. D. Tonchev. "Symmetric (4, 4)-nets and generalized Hadamard matrices over groups of order 4." *Designs, Codes and Cryptography* 34.1 (2005): 71-87.
- [6] F. Fujiwara, and P. Vandendriessche. "Quantum synchronizable codes from finite geometries." *Information Theory, IEEE Transactions* 60.11(2013).
- [7] L. Guo, Y. Ma, and Y. Feng. "Quantum Codes Constructed from Self-Dual Codes and Maximal Self-Orthogonal Codes Over F5." *Procedia Engineering* 29 (2012): 3448-3453.
- [8] J. Kim and J. Walker. "Nonbinary quantum error-correcting codes from algebraic curves." *Discrete Mathematics* 308.14 (2008): 3115-3124.
- [9] R. Li, and X. Li. "Binary construction of quantum codes of minimum distances five and six." *Discrete Mathematics* 308.9 (2008): 1603-1611.
- [10] C. Sarami, "Topics in Coding Theory and Combinatorial Structures", PhD Dissertation, *Michigan Technological University*, 2004.
- [11] A. M. Steane, . "Enlargement of calderbank-shor-steane quantum codes." *Information Theory, IEEE Transactions* 45.7 (1999): 2492-2495.





## **SESSION**

# **OPTIMIZATION METHODS + SEARCH AND INDEXING ALGORITHMS + REAL-TIME SYSTEMS AND HPC + APPLICATIONS**

**Chair(s)**

**TBA**



# On the Intersection of Inverted Lists

Yangjun Chen<sup>1</sup> and Weixin Shen<sup>2</sup>

Dept. Applied Computer Science, University of Winnipeg, Canada

<sup>1</sup>y.chen@uwinnipeg.ca, <sup>2</sup>wxshen1985@gmail.com

**Abstract**— In this paper, we discuss an efficient and effective index mechanism to support set intersections, which are important to evaluation of conjunctive queries by search engines. The main idea behind it is to decompose an inverted list associated with a word into a collection of disjoint sub-lists by arranging a set of word sequences into a trie structure. Then, by using a kind of tree encoding, we can replace each inverted list with a much shorter interval sequence. In this way, we can transform the comparison of document identifiers to the checking of interval containment by associating each interval with a sub-list. More importantly, for a sorted interval sequence the binary search can also be used. With the lowest common ancestors utilized to control the search, a better theoretical time complexity than any traditional method can be achieved.

**Key words:** Search engine; inverted files; conjunctive queries; disjunctive queries.

## 1. INTRODUCTION

Indexing the Web for fast keyword search is among the most challenging applications for scalable data management. In the past several decades, different indexing methods have been developed to speed up text search, such as *inverted files* [14, 15], *signature files* and *signature trees* for indexing texts [1, 5, 6, 11, 12]; and *suffix trees* and *tries* [13] for string matching. Especially, different variants of inverted files have been used by the Web search engines to find pages satisfying conjunctive queries of the form:

$$w_1 \wedge w_2 \wedge \dots \wedge w_k.$$

A document  $D$  is an answer to such a query if it contains every  $w_i$  for  $1 \leq i \leq k$ . The algorithms developed to evaluate such a query typically use *inverted lists*, each of which comprises all those document identifiers containing a certain word. So, to find all the documents satisfying a query, set intersections have to be conducted.

There has been considerable study on this topic, such as *adaptive* algorithms [9], *melding* algorithms [2], building additional data structures like *skipping lists* [32], *treaps* (a kind of balanced trees) [4], *hash tables* over sorted lists [3, 10], and so on. All of them can improve the time complexity at most by a constant factor, but none of them is able to break through the linear time bottleneck.

In this work, we explore a different way to speed up the operation by constructing indexes, which are substantially different from any existing strategy. Concretely, our method works as follows.

- Represent each document as a word sequence, sorted decreasingly by the word appearance frequency (referred to as a *document word sequence*, or simply a *word sequence*), and then construct a trie structure over all such sequences.

- Associate each word with an interval sequence  $L$ , where each interval in  $L$  is created by applying a kind of tree encoding over the generated trie structure.
- Associate each interval, rather than a word, with a set of document identifiers. In this way, we decompose an inverted list associated with a word into a collection of disjoint sub-lists, and transform the comparison of document identifiers to the checking of interval containment.
- For each word  $w$ , instead of its interval sequence, we will construct a balanced binary tree over an even shorter interval sequence with each being an interval for a lowest common ancestor of some nodes labelled with  $w$ . The set intersection operation can then be done by searching a binary tree against a series of intervals.

Let  $\delta_x$  and  $\delta_y$  be two inverted lists associated with two words  $x$  and  $y$ , respectively. Without loss of generality, assume that  $|\delta_x| < |\delta_y|$ . Up to now, the best comparison-based algorithm for intersecting  $L_x$  and  $L_y$  requires  $O(|\delta_x| \cdot \log \frac{|\delta_y|}{|\delta_x|})$

time. In contrast, our algorithm needs  $O(|L_y| \cdot \log \frac{\lambda_x}{|L_y|})$  time,

where  $L_x$  and  $L_y$  are the interval sequences created for  $L_x$  and  $L_y$ , respectively; and  $\lambda_x$  is the size of a subset of nodes with each being a lowest common ancestor of some nodes labeled with  $x$  in the trie. Generally, we have  $|L_y| \leq |L_x| \leq |\delta_x|$  and  $\lambda_x < |\delta_x|$ . This time complexity is significantly better than the traditional methods since an interval sequence can be much shorter than its corresponding inverted list. Especially, the larger an inverted list is, the smaller its corresponding interval sequence. Only for those very short inverted lists (associated with low frequent words), may the sizes of their corresponding interval sequences be near their sizes. In fact, we only care about the cases of very long inverted lists. When all the inverted lists involved in an operation is short, all the methods work fast.

Finally, we indicate that our index structure can also be easily maintained.

## 2. NEW INDEX STRUCTURE

In this section, we mainly discuss our index structure, by which each word with high frequency will be assigned an interval sequence. We will then associate intervals, instead of words, with inverted sub-lists. To clarify this mechanism, we will first discuss interval sequences for words in 2.1. Then, in

2.2, how to associate inverted lists with intervals will be addressed.

## 2.1 Interval sequences assigned to words

Let  $D = \{D_1, \dots, D_n\}$  be a set of documents. Let  $W_i = \{w_{i1}, \dots, w_{ij}, \dots, w_{in}\}$  ( $i = 1, \dots, n$ ) be all of the words appearing in  $D_i$ , to be indexed. Denote  $W = \bigcup_{i=1}^n W_i$ , called the *vocabulary*. For each word  $w \in W$ , we will associate it with an inverted list containing all the document identifiers with each containing  $w$ . Thus, to answer a conjunctive query, a set intersection over some inverted lists has to be conducted.

For the purpose of the new index structure, we will put all the words in a sorted sequence  $\mathcal{Q} = w_1, w_2, \dots, w_m$  ( $m = |W|$ ) such that for any two words  $w$  and  $w'$  if the frequency of  $w$  is higher than  $w'$  then  $w$  appears before  $w'$  in  $\mathcal{Q}$ , denoted as  $w < w'$ . Then, each document can be represented as a subsequence of  $\mathcal{Q}$ ; and over all these subsequences a trie structure can be established as illustrated in Fig. 1.

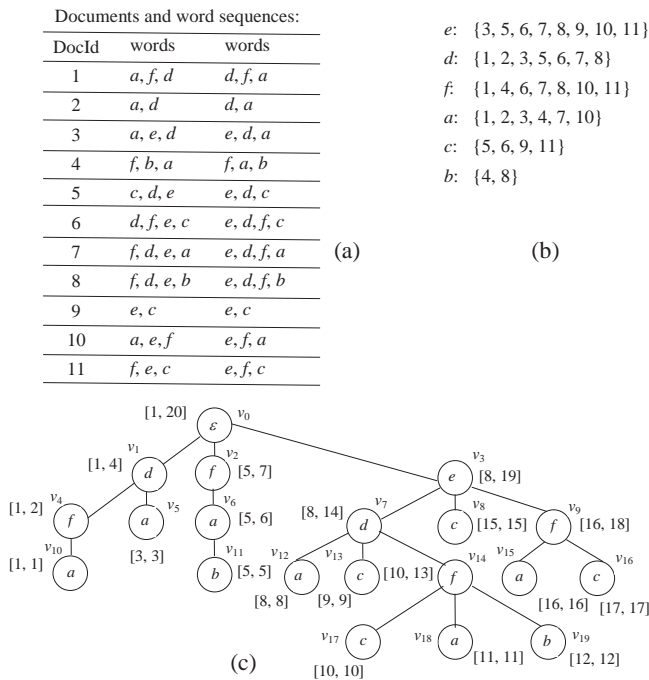


Fig. 1: A trie and a set of sorted interval sequences

In Fig. 1(a), we show a document database containing 11 documents, their words, and their sorted sequences by the word frequency, where we use a character to represent a word for simplicity. In Fig. 1(b), we show the inverted lists for all the words in the database. The trie over all the sorted sequences is shown in Fig. 1(c).

In this trie,  $v_0$  is a virtual root, labeled with an *empty* word  $\varepsilon$  while any other node is labeled with a *real* word. Therefore, all the words on a path from the root to a leaf spell a sorted word sequence for a certain document. For instance, the path from  $v_0$  to  $v_{13}$  corresponds to the sequence:  $c, f, a, p, m$ . Then, to check whether two words  $w_1$  and  $w_2$  are in the

same document, we need only to check whether there exist two nodes  $v_1$  and  $v_2$  such that  $v_1$  is labeled with  $w_1$ ,  $v_2$  with  $w_2$ , and  $v_1$  and  $v_2$  are on the same path. This shows that the *reachability* needs to be checked for this task, by which we ask whether a node  $v$  can reach another node  $u$  through a path. If it is the case, we denote it as  $v \Rightarrow u$ ; otherwise, we denote it as  $v \not\Rightarrow u$ .

The reachability problem on tries can be solved very efficiently by using a kind of tree encoding [7][8], which labels each node  $v$  in a trie with an interval  $I_v = [\alpha_v, \beta_v]$ , where  $\beta_v$  denotes the rank of  $v$  in a *post-order* traversal of the trie. Here the ranks are assumed to begin with 1, and all the children of a node are assumed to be ordered and fixed during the traversal. Furthermore,  $\alpha_v$  denotes the lowest rank for any node  $u$  in  $T[v]$  (the subtree rooted at  $v$ , including  $v$ ). Thus, for any node  $u$  in  $T[v]$ , we have  $I_u \subseteq I_v$  since the post-order traversal visits a node after all of its children have been accessed. In Fig. 1(c), we also show such a tree encoding on the trie, assuming that the children are ordered from left to right. It is easy to see that by interval containment we can check whether two nodes are on a same path. For example,  $v_3 \Rightarrow v_{19}$ , since  $I_{v_3} = [8, 19]$ ,  $I_{v_{19}} = [12, 12]$ , and  $[12, 12] \subset [8, 19]$ ; but  $v_2 \not\Rightarrow v_{18}$ , since  $I_{v_2} = [5, 7]$ ,  $I_{v_{18}} = [11, 11]$ , and  $[11, 11] \not\subset [5, 7]$ .

Let  $I = [\alpha, \beta]$  be an interval. We will refer to  $\alpha$  and  $\beta$  as  $I[1]$  and  $I[2]$ , respectively.

**Lemma 1** For any two intervals  $I$  and  $I'$  generated for two nodes in a trie, one of four relations holds:  $I \subset I'$ ,  $I' \subset I$ ,  $I[2] < I'[1]$ , or  $I'[2] < I[1]$ .

*Proof.* It is easy to prove.  $\square$

However, more than one node may be labeled with the same word, such as nodes  $v_1$ , and  $v_7$  in Fig. 1(c). Both are labeled with word  $d$ . Therefore, a word may be associated with more than one node (or say, more than one node's interval). Thus, to know whether two words are in the same document, multiple checkings may be needed. For example, to check whether  $d$  and  $b$  are in the same document, we need to check  $v_1$  and  $v_7$  each against both  $v_{11}$  and  $v_{19}$ , by using the node's intervals.

In order to minimize such checkings, we associate each word  $w$  with an interval sequence of the form:  $L_w = I_w^1, I_w^2, \dots, I_w^k$ , where  $k$  is the number of all those nodes labeled with  $w$  and each  $I_w^i = [I_w^i[1], I_w^i[2]]$  ( $1 \leq i \leq k$ ) is an interval associated with a certain node labeled with  $w$ . In addition, we can sort  $L_w$  by the interval's first value such that for  $1 \leq i < j \leq k$  we have  $I_w^i[1] < I_w^j[1]$ , which will greatly reduce the time for the reachability checking. We illustrate this in Fig. 2, in which each word in Fig. 1(a) is associated with an interval sequence.

From this figure, we can see that for any two intervals  $I$  and  $I'$  in  $L_w$  we must have  $I \not\subset I'$ , and  $I' \not\subset I$  since in any trie no two nodes on a path are labeled with the same word.

In addition, for any interval sequence  $L$ , we will use  $L[i]$  to refer to the  $i$ th interval in  $L$ , and  $L[i..j]$  to the segment from the  $i$ th to the  $j$ th interval in  $L$ .

- $e$ : [8,19]
- $d$ : [1, 4][8, 14]
- $f$ : [1, 2][5, 7][10, 13][16, 18]
- $a$ : [1, 1][3, 3][5, 6][8, 8][11, 11][16, 16]
- $c$ : [9, 9][10, 10][15, 15][17, 17]
- $b$ : [5, 5][12, 12]

Fig. 2: a set of interval sequences

### 2.2 Assignment of DocIDs to intervals

Another important component of our index is to assign document identifiers to intervals. An interval  $I$  can be considered as a representative of some words, i.e., all those words appearing on a *prefix* in the trie, which is a path  $P$  from the root to a certain node that is labeled with  $I$ . Then, the document identifiers assigned to  $I$  should be those containing all the words on  $P$ . For example, the words appearing on the prefix:  $v_0 \rightarrow v_3 \rightarrow v_7 \rightarrow v_{14}$  in the trie shown in Fig. 1(c) are words:  $\varepsilon$ ,  $e$ ,  $d$ , and  $f$ , represented by the interval [10, 13] associated with  $v_{14}$ . So, the document identifiers assigned to [10, 13] should be  $\{6, 7, 8\}$ , indicating that documents  $D_6, D_7$  and  $D_8$  all contain those three words. See the trie shown in Fig. 3 for illustration, in which each node  $v$  is assigned a set of document identifiers that is also considered to be the set assigned to the interval associated with  $v$ .

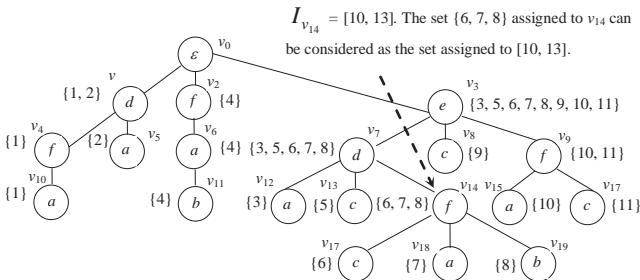


Fig. 3: Illustration for assignment of document IDs

Let  $v$  be the ending node of a prefix  $P$ , labeled with  $I$ . We will use  $\delta_P$ , interchangeably  $\delta_v$ , to represent the set of document identifiers containing the words appearing on  $P$ . Thus, we have, for example,  $\delta_{v_{14}} = \delta_{[10, 13]} = \{6, 7, 8\}$ . Concerning the decomposition of inverted lists, the following two lemmas can be easily proved.

**Lemma 1** Let  $T$  be a trie constructed over a set of word sequences (sorted by the appearance frequency) over  $\mathbf{W}$ . Then, we have  $\sum_{v \in T} \delta_v = \sum_{w \in \mathbf{W}} \delta_w$ .

*Proof.* Let  $v_1, \dots, v_{l_w}$  be all the nodes labeled with a word  $w$  in  $T$ . Then  $\delta_w = \sum_{i=1}^{l_w} \delta_{v_i}$ . Since in  $T$  no node is labeled with more

than one word, we have  $\sum_{w \in \mathbf{W}} \delta_w = \sum_{w \in \mathbf{W}} \sum_{i=1}^{l_w} \delta_{v_i} = \sum_{v \in T} \delta_v$ .  $\square$

**Lemma 2** Let  $u$  and  $v$  be two nodes in a trie  $T$ . If  $u$  and  $v$  are not on the same path in  $T$ , then  $\delta_u$  and  $\delta_v$  are disjoint, i.e.,  $\delta_u \cap \delta_v = \emptyset$ .

*Proof.* It is easy to prove.  $\square$

**Proposition 1** Assume that  $v_1, v_2, \dots, v_j$  be all the nodes labeled with the same word  $w$  in  $T$ . Then,  $\delta_w$ , the inverted list of  $w$  (i.e., the list of all the documents identifiers containing  $w$ ) is equal to  $\delta_{v_1} \cup \delta_{v_2} \cup \dots \cup \delta_{v_j}$ , where  $\cup$  represents *disjoint union* over disjoint sets that have no elements in common.

*Proof.* Obviously,  $\delta_w$  is equal to  $\delta_{v_1} \cup \delta_{v_2} \cup \dots \cup \delta_{v_j}$ . Since  $v_1, v_2, \dots, v_j$  are labeled with the same word, they definitely appear on different paths as no nodes on a path are labeled with the same word. According to Lemma 1,  $\delta_{v_1} \cup \delta_{v_2} \cup \dots \cup \delta_{v_j}$  is equal to  $\delta_{v_1} \cup \delta_{v_2} \cup \dots \cup \delta_{v_j}$ .  $\square$

As an example, see the nodes  $v_1$  and  $v_7$  in Fig. 2. Both are labeled with word  $d$ . So the inverted list of  $d$  is  $\delta_{v_1} \cup \delta_{v_7} = \{1, 2\} \cup \{3, 5, 6, 7, 8\} = \{1, 2, 3, 5, 6, 7, 8\}$ .

### 3. BASIC QUERY EVALUATION

Based on the new index structure, we design our basic algorithms.

We first consider a query containing only two words  $w \wedge w'$  with  $w < w'$ . It is easy to see that any interval in  $L_w$  cannot be contained in any interval in  $L_{w'}$ . Thus, to check whether  $w$  and  $w'$  are in the same document, we need only to check whether there exist  $I \in L_w$  and  $I' \in L_{w'}$  such that  $I \supset I'$ . Therefore, such a query can be evaluated by running a process, denoted as  $conj(L_w, L_{w'})$ , to find all those intervals in  $L_w$  with each being contained in some interval in  $L_{w'}$ , stored in a new sequence  $L$ .

1. Let  $L_w = I_w^1, I_w^2, \dots, I_w^k$ . Let  $L_{w'} = I_{w'}^1, I_{w'}^2, \dots, I_{w'}^{k'}$ .  $L \leftarrow \emptyset$ .
2. Step through  $L_w$  and  $L_{w'}$  from left to right. Let  $I_w^p$  and  $I_{w'}^q$  be the intervals currently encountered. We will do one of the following checkings:
  - i) If  $I_w^p \supset I_{w'}^q$ , append  $I_{w'}^q$  to the end of  $L$ . Move to  $I_{w'}^{q+1}$  if  $q < k'$  (then, in a next step, we will check  $I_w^p$  against  $I_{w'}^{q+1}$ ). If  $q = k'$ , stop.
  - ii) If  $I_w^p [1] > I_{w'}^q [2]$ , move to  $I_w^{p+1}$  if  $p < k$ . If  $p = k$ , stop.
  - iii) If  $I_w^p [2] < I_{w'}^q [1]$ , move to  $I_w^{p+1}$  if  $p < k$  (then, in a next step, we will check  $I_w^{p+1}$  against  $I_{w'}^q$ ). If  $p = k$ , stop.  $\square$

Assume that the result is  $L = I_1, I_2, \dots, I_l$  ( $0 \leq l \leq k'$ ). Then, for each  $1 \leq j \leq l$ , there exists an interval  $I \in L_w$  such that  $I_j \subset I$ , and we can return  $\delta_{I_1} \cup \dots \cup \delta_{I_l}$  as the answer. In Fig. 4, we illustrate the working process on  $L_d$  and  $L_b$  shown in Fig. 2.

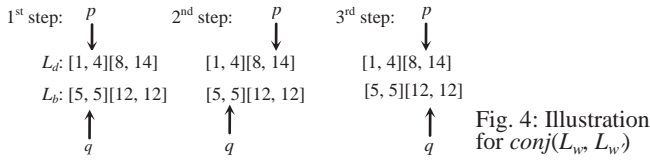


Fig. 4: Illustration for  $\text{conj}(L_w, L_w)$

In Fig. 4, we first notice that  $L_d = [1, 4][8, 14]$  and  $L_b = [5, 5][12, 12]$ . In the 1<sup>st</sup> step, we will check  $L_d^1 = [1, 4]$  against  $L_b^1 = [5, 6]$ . Since  $L_d^1[2] = 4 < L_b^1[1] = 5$ , we will check  $L_d^2 = [8, 14]$  against  $L_b^1$  in a next step, and find  $L_b^1[2] = 5 < L_d^2[1] = 8$ . So we will have to do the third step, in which we will check  $L_d^2$  against  $L_b^2 = [12, 12]$ . Since  $L_d^2 \supset L_b^2$ , we get to know that  $d$  and  $b$  are in the same document.

**Lemma 3** Let  $L = I_1, \dots, I_k$  be the result of  $\text{conj}(L_w, L_w)$ . Then, for each  $I_j$  ( $1 \leq j \leq k$ ), there must be an interval  $I \in L_w$  such that  $I \supset I_j$ . For any interval  $I' \in L_w$  but  $\notin L$ , it definitely does not belong to any interval in  $L_w$ .

*Proof.* It is easy to prove.  $\square$

Since in this process, each interval in both  $L_w$  and  $L_w$  is accessed only once, the time complexity of this process is bounded by  $O(|L_w| + |L_w|)$ . In addition, the above approach can be easily extended to evaluate general queries of the form  $Q = w_1 \wedge w_2 \wedge \dots \wedge w_l$  with  $w_1 < w_2 < \dots < w_l$  and  $l \geq 1$  based on the transitivity of intervals:  $I \supseteq I' \supseteq I'' \rightarrow I \supseteq I''$ .

What we need to do is to repeatedly apply  $\text{conj}()$  to the corresponding interval sequences associated with the query words one by one. The following is a formal description of the process.

---

#### ALGORITHM $\text{conEvaluation}(Q)$

---

**begin**

1. let  $|Q| = l$ ; assume that  $Q[1] < Q[2] < \dots < Q[l]$ ;
2.  $L := Q[1]$ ;
3. **for** ( $j = 2$  to  $l$ ) **do**
4. {  $L \leftarrow \text{conj}(L, L_{Q[j]})$  ; }
5. let  $L = I_1, \dots, I_k$ ;
6. return  $\delta_{I_1} \cup \dots \cup \delta_{I_k}$ .

**end**

---

It is easy to see that the time complexity of the algorithm is bounded by  $O(\sum_{w \in Q} |L_w|)$ .

**Proposition 2** Let  $Q = w_1 \wedge w_2 \wedge \dots \wedge w_l$  with  $w_1 < w_2 < \dots < w_l$  and  $l \geq 1$ . The answer produced by algorithm  $\text{conEvaluation}(Q)$  is correct.

*Proof.* Let  $L = I_1, \dots, I_k$  be the interval sequence produced by the main **for**-loop (line 3 – 4). Then, according to Lemma 3, for each  $I_j$  ( $1 \leq j \leq k$ ), there must exist an interval sequence  $t_1, t_2, \dots, t_{l-1}$  such that  $t_i \in L_{w_i}$  ( $1 \leq i \leq l-1$ ) and  $t_1 \supset t_2 \supset \dots \supset t_{l-1} \supset I_j$ . Next, according to Proposition 1, we know that  $\delta_{t_1} \cup \dots \cup \delta_{t_k}$  must be the correct answer.  $\square$

**Example 1** Consider Fig. 2 and 3. Let  $Q = d \wedge f \wedge a$ . Then, in the first iteration, we will get  $L = \text{conj}(L_d, L_f) = [1, 2][10, 13]$ . In the second iteration, we will get  $L = \text{conj}(L, L_a) = [1, 1][11, 11]$ . The results is then  $R = \delta_{[1, 1]} \cup \delta_{[11, 11]} = \{1\} \cup \{7\} = \{1, 7\}$ .  $\square$

## 4. IMPROVEMENTS

In this section, we discuss a new algorithm to improve the naïve method shown in the previous subsection. The main idea is to use lowest common ancestors (LCAs for short) of nodes (in  $T$ ) to control a binary search process. First, in 4.1, we discuss the binary search of an  $L_w$ . Then, we show how to use LCAs to speed up such a search in 4.2.

### 4.1 Set intersection based on binary search

Each interval sequence is sorted. So we can do the conjunction of interval sequences based on binary search.

Let  $L_o = I_o^1, I_o^2, \dots, I_o^m$  and  $L_w = I_w^1, I_w^2, \dots, I_w^n$  be two interval sequences with  $o < w$ . Then,  $m = |L_o| \leq n = |L_w|$ .

By using the binary search technique, we need to work from the end to the start of  $L_w$  to incorporate the LCAs into the process. To this end, we design an algorithm different from  $\text{conj}(L_o, L_w)$ , called  $\text{conjB}()$ , which can be mostly easily described recursively. When  $m = 0$ , there is no conjunction to be done and the result is  $\phi$ . Otherwise, we will first check

$I_o^m$  against  $L_w$ . As with [46], let  $l = \left\lfloor \lg \frac{n}{m} \right\rfloor$ . Then,  $2^l$  is the largest power of two not exceeding  $\frac{n}{m}$ . Let  $t = n - 2^l + 1$ .

Compare  $I_o^m$  and  $I_w^t$ .

1. If  $I_o^m[1] > I_w^t[2]$ , we should look for the intervals (in  $L_w$ ) covered by  $I_o^m$  somewhere to the right of  $I_w^t$ . By using the traditional binary search, we try to find an interval  $I$  covered by  $I_o^m$  with  $l$  more comparisons. Around  $I$ , we will continually (by a simple linear search) find the left-most interval  $x$  in  $L_w$ , which can be covered by  $I_o^m$ ; and then with  $l$  more comparisons, we will find the right-most interval  $y$  covered by  $I_o^m$ , in a similar way. Obviously, all the intervals between  $x$  and  $y$ , including  $x$  and  $y$ , can be covered by  $I_o^m$ . (See Fig. 5(a).) This information allows us to reduce the problem to the situation illustrated in Fig. 5(b). To complete the whole operation, it is sufficient to apply the above process to  $L_o'$  and  $L_w'$ , where  $L_o' = I_o^1, \dots, I_o^{m-1}$  and  $L_w' = I_w^1, \dots, I_w^{x-1}$ .
2. If, on the other hand,  $I_o^m[2] < I_w^t[1]$ , we should check the intervals to the left of  $I_w^t$ , and the problem immediately reduces to the checking of  $L_o' = L_o$  against  $L_w' = L_w[1 .. t-1]$ . We can complete the operation by applying the above process to  $L_o'$  and  $L_w'$ .

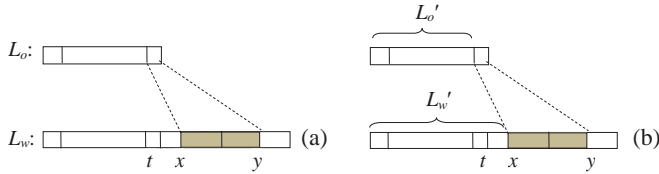
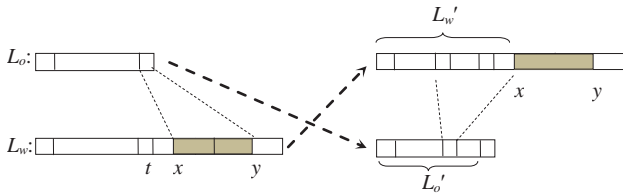


Fig. 5: First comparison during an interval intersection

However,  $L_o'$  may become larger than  $L_w'$ . So in the recursive call to  $conjB()$ , the roles of  $L_o'$  and  $L_w'$  may be reversed, by which we will check each interval  $I$  in  $L_w'$  against  $L_o'$  to find an interval  $I'$  in  $L_w'$  such that  $I' \supset$  the last interval in  $L_o'$ . See Fig. 6 for illustration. Assume that the last interval  $I_w^{x-1}$  in  $L_w'$  is covered by an interval  $I_o^j$  ( $1 \leq j \leq m-2$ ) in  $L_o'$ . Then, by the next recursive call, we will check  $L_w'' = I_w^1, \dots, I_w^{x-2}$  and  $L_o'' = I_o^1, \dots, I_o^{j-2}$ .

Fig. 6: Illustration for interchanging rolls of  $L_w'$  and  $L_o'$ 

3. If  $I_o^m \supset I_w^t$ , we will check linearly  $I_w^{t-1}, I_w^{t-2}, \dots$  until we meet a first interval  $x'$  which is to the left of  $I_w^t$  and not covered by  $I_o^m$ . Then, check  $I_w^{t+1}, I_w^{t+2}, \dots$  until a first interval  $y'$  which is to the right of  $I_w^t$  and not covered by  $I_o^m$ . All the encountered nodes, except  $x'$  and  $y'$ , must be covered by  $I_o^m$ . This reduces the problem to a checking of  $L_o' = L_o[1 .. m-1]$  against  $L_w' = L_w[1 .. x']$ .
4. If  $I_o^m \subset I_w^t$  (we may have this case due to the roll interchange), we add  $I_o^m$  to the result and the problem reduces to a checking  $L_o' = L_o[1 .. m-1]$  against  $L_w' = L_w[1 .. t]$ .

According to the above discussion, we give the following recursive algorithm, which takes three inputs:  $L_o$ ,  $L_w$ ,  $b$  with  $|L_o| \leq |L_w|$ , where  $b$  is a Boolean value used to indicate how  $I_o^m$  is checked against  $L_w$ . If  $o < w$ ,  $b = 0$ . Otherwise ( $w < o$ ),  $b = 1$ . In addition, in the Algorithm a global variable  $R$  is used to store the result.

---

**ALGORITHM**  $conjB(L_o, L_w, b)$ 


---

**begin**

1.  $m \leftarrow |L_o|$ ;  $n \leftarrow |L_w|$ ;
2. **if**  $m = 0$  **then** return;
3.  $l \leftarrow \lfloor \lg \frac{n}{m} \rfloor$ ;  $t \leftarrow n - 2^l + 1$ ;  $I \leftarrow I_o^m$ ;
4. **if**  $l[2] < I_w^t[1]$  **then**  $\{L_o' \leftarrow L_o; L_w' \leftarrow L_w[1 .. t-1]\}$

5. **if**  $l[1] > I_w^t[2]$
  6. **then if**  $b = 1$  **then**  $z \leftarrow binaryS-I(I, L_w[t+1 .. n])$
  7. **if**  $z = 0$  **then**  $\{L_o' \leftarrow L_o[1 .. m-1]; L_w' \leftarrow L_w\}$ ;
  8. **else**  $R := R \cup \{I\}$ ;
  9.  $L_o' \leftarrow L_o[1 .. m-1]$ ;
  10.  $L_w' \leftarrow L_w[1 .. t+z-1]$ ;
  11. **else**  $\langle x, y \rangle \leftarrow binaryS-2(I, L_w[t+1 .. n])$
  12. **if**  $x = 0$  **then**  $\{L_o' \leftarrow L_o[1 .. m-1]; L_w' \leftarrow L_w\}$ ;
  13. **else**  $R \leftarrow R \cup \{\text{all interval between } x \text{ and } y, \text{ including } x \text{ and } y\}$ ;
  14.  $L_o' \leftarrow L_o[1 .. m-1]$ ;  $L_w' \leftarrow L_w[1 .. x-1]$ ;
  15. **if**  $I \supset I_w^t$  **then**  $\langle x, y \rangle \leftarrow linearSearch(I, L_w, I_w^t)$
  16.  $L_o' \leftarrow L_o[1 .. m-1]$ ;  $L_w' \leftarrow L_w[1 .. x-1]$ ;
  17.  $R \leftarrow R \cup \{\text{all interval between } x \text{ and } y, \text{ including } x \text{ and } y\}$ ;
  18. **if**  $I \subset I_w^t$  **then**  $R := R \cup \{I\}$ ;
  19.  $L_o' \leftarrow L_o[1 .. m-1]$ ;  $L_w' \leftarrow L_w[1 .. t]$ ;
  20. **if**  $|L_o'| \leq |L_w'|$  **then**  $conjB(L_o', L_w', b)$
  21. **else**  $conjB(L_w', L_o', \bar{b})$ ;
  - end**
- 

The above algorithm can be divided into two parts. The first part consists of lines 1 – 10; and the second part lines 20 – 21. In the first part, we will check the last interval  $I_o^m$  in  $L_o$  against  $L_w$ . According to the above discussion, four cases are distinguished:  $I_o^m[2] < I_w^t[1]$  (line 4),  $I_o^m[1] > I_w^t[2]$  (lines 4 – 14),  $I_o^m[1] \supset I_w^t$  (lines 15 – 17), and  $I_o^m[1] \subset I_w^t$  (18 – 19). Special attention should be paid to the use of  $b$ , which indicates whether we check  $I_o^m$  to find a covering interval in  $L_w$  (by calling  $binaryS-I()$ ) or to find all those intervals that can be covered by  $I_o^m$  (by calling  $binaryS-2()$ ). In the second part (lines 20 – 21), we make a recursive call to check  $L_o'$  and  $L_w'$ , which are determined respectively from  $L_o$  and  $L_w$  during the execution of the first part. If  $|L_o'| \leq |L_w'|$ , we simply call  $conjB(L_o', L_w', b)$  (see line 14.) Otherwise, the rolls of  $L_o$  and  $L_w$  should be interchanged and we will call  $conjB(L_w', L_o', \bar{b})$ , where  $\bar{b}$  represents the negation of  $b$  (see line 21.)

In  $binaryS-I(I, L)$ , we will find, by the binary search, an interval  $I_z$  in  $L$  which covers  $I$ . If  $z = 0$ , it shows that such an interval does not exist.

---

**FUNCTION**  $binaryS-I(I, L)$ 


---

**begin**

1.  $z \leftarrow 0$ ;
  2. binary search of  $L$  to find an interval  $z$ , which covers  $I$ ;
  3. return  $z$ ;
- end**
- 

In  $binaryS-2(I, L)$ , we will first find a pair  $\langle x, y \rangle$  such that  $I_x$  is the left-most interval in  $L$ , which can be covered by  $I$ ; and  $I_y$  the right-most interval covered by  $I$ . Then,  $x = 0$  indicates that no interval in  $L$  is covered by  $I$ .

**FUNCTION** *binaryS-2(I, L)***begin**

1.  $x \leftarrow 0; y \leftarrow 0;$
2. binary search of  $L$  to find an interval  $I_z$  which is covered by  $I$ ;
3. return *linearSearch(I, L, I<sub>z</sub>)*;

**end**

In *linearSearch(I, L, I')*, we will find a pair  $\langle x, y \rangle$  such that  $I_x, I_{x+1}, \dots, I'_y, \dots, I_{y-1}, I_y$  are all the intervals that can be covered by  $I$ .

**FUNCTION** *linearSearch(I, L, I')***begin**

1. Let  $I'$  be  $I_z$ ;
2. Search  $I_{z-1}, I_{z-2}, \dots$  until  $I_x$  such that  $I_x$  is covered by  $I$ , but  $I_{x-1}$  not;
3. Search  $I_{z+1}, I_{z+2}, \dots$  until  $I_y$  such that  $I_y$  is covered by  $I$ , but  $I_{y+1}$  not;
2. return  $\langle x, y \rangle$ ;

**end**

**Example 2** Consider  $L_d = [1, 4][8, 14]$  and  $L_a = [1, 1][3, 3][5, 6][8, 8][11, 11][16, 16]$ . By calling *conjB(L<sub>f</sub>, L<sub>a</sub>, 0)*, the following operations will be conducted:

Step 1: checking  $L_d[2] = [8, 14]$  against  $L_a$ .  $l = \left\lfloor \lg \frac{6}{2} \right\rfloor = 1, t = n - 2^l + 1 = 6 - 2 + 1 = 5, L_d[5] = [11, 11]$ . Since  $[11, 11] \subset [8, 14]$ , we will call *linearSearch()* to find  $x = 4$  and  $y = 5$ .

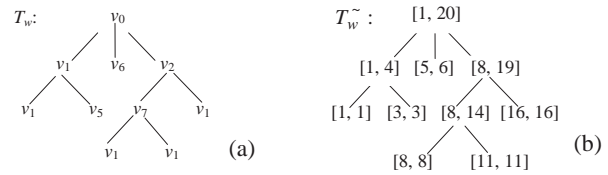
Step 2: checking  $L_d[1] = [1, 4]$  against  $L_a[1 \dots 3]$ .  $l = \left\lfloor \lg \frac{3}{1} \right\rfloor = 1, t = 3 - 2^1 + 1 = 2, L_d[2] = [3, 3]$ . Since  $[3, 3] \subset [1, 4]$ , we will call *linearSearch()* to find  $x = 1$  and  $y = 2$ .  $\square$

**4.2 Search control by using LCAs**

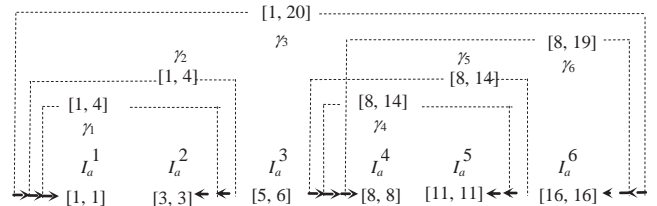
The method discussed in 4.1 can be significantly improved by using *LCAs*. Given a word  $w$ , denote by  $V_w$  all the nodes labeled with  $w$ . All the *LCAs* of the nodes in  $V_w$  (in  $T$ ), denoted as  $V_w'$ , can be efficiently recognized using a way to be discussed in Section 6. For example, for the set of nodes labeled with word  $a$ :  $V_a = \{v_{10}, v_5, v_6, v_{12}, v_{18}, v_{15}\}$ , we can find another set of nodes:  $V_a' = \{v_1, v_7, v_2, v_0\}$  with  $v_1$  being *LCA* of  $\{v_{10}, v_5\}$ ,  $v_7$  being *LCA* of  $\{v_{12}, v_{18}\}$ ,  $v_2$  being *LCA* of  $\{v_6, v_{12}, v_{18}, v_{15}\}$ , and  $v_0$  being *LCA* of  $\{v_{10}, v_5, v_6, v_{12}, v_{18}, v_{15}\}$ . Now we construct a tree structure, called an *LCA-tree* and denoted as  $T_w$ , which contains all the nodes in  $V_w \cup V_w'$ . In  $T_w$ , there is arc from  $v_1$  to  $v_2$  iff there exists a path  $P$  from  $v_1$  to  $v_2$  in  $T$  and  $P$  does not pass any other node in  $V_w \cup V_w'$ . In Fig. 7(a), we show  $T_a$  for illustration.

Replacing each node in  $T_w$  with the corresponding interval, we get another tree, denoted as  $T_w^\sim$ , in which each internal node  $v$  must be an interval that is the smallest interval covering all the intervals represented by the leaf nodes in  $T_w^\sim[v]$  (the subtree rooted at  $v$  in  $T_w^\sim$ ). See  $T_a^\sim$  shown in Fig. 7(b) for illustration. From this, we can see that  $[1, 4]$  is the

smallest interval covering  $[1, 1]$  and  $[3, 3]$ ;  $[8, 14]$  is the smallest interval covering  $[8, 8]$  and  $[11, 11]$ ; and  $[8, 19]$  is the smallest interval covering  $[8, 8]$ ,  $[11, 11]$  and  $[16, 16]$ . Finally,  $[1, 20]$  is the smallest interval covering all the intervals in  $L_a$ :  $[1, 1], [3, 3], [5, 6], [8, 8], [11, 11], [16, 16]$ .

Fig. 7: Illustration for  $T_w$  and  $T_w^\sim$ 

Here, our intention is to associate each interval  $I_w^j$  in  $L_w$  with a second interval  $\gamma_j$ , which is the parent of  $I_w^j$  in  $T_w^\sim$ , and two links, denoted as  $l_j$  and  $r_j$ , respectively pointing to two intervals in  $L_w$ , which are respectively the left-most and right-most leaf nodes in  $T_w^\sim[\gamma_j]$ . Fig. 8 helps for illustration.

Fig. 8: Illustration for links associated with intervals in  $L_w$ 

In Fig. 8,  $I_a^3 = [5, 6]$  is associated with an *LCA* interval  $\gamma_3 = [8, 14]$ , which is the parent of  $I_a^3$  in the corresponding  $T_a^\sim$  shown in Fig. 7(b). In addition,  $l_3$  is a link pointing to  $I_a^1$  and  $r_3$  is a link pointing to  $I_a^6$ . They are respectively the left-most interval and the right-most interval covered by  $\gamma_3$ . In the same way, we can check all the other intervals and links shown in Fig. 8.

In addition, we will keep a sequence  $\Gamma_w$  containing all the *LCA* intervals in the post-order of  $T_w^\sim$ . For example,  $\Gamma_a = \gamma_1\gamma_4\gamma_6\gamma_3 = [1, 4][8, 14][8, 19][1, 20]$ . With such intervals and links, the binary search of  $L_w$  against a certain interval (in  $L_o$ ) can be done much more efficiently by skipping over useless checkings. Concretely, the checking of  $I_o^m$  against  $L_w$  will be done as follows.

1. If  $I_o^m[1] > I_w^t[2]$ , compare  $I_o^m$  and  $\gamma_t$ . If  $I_o^m \not\subset \gamma_t$ , explore  $L_w[r_t + 1 \dots n]$  by the binary search. Otherwise, explore  $L_w[t + 1 \dots r_t]$ .
2. If  $I_o^m[2] < I_w^t[1]$ , compare  $I_o^m$  and  $\gamma_t$ . If  $I_o^m \not\subset \gamma_t$ , explore  $L_w[1 \dots l_t - 1]$ . Otherwise ( $I_o^m \subset \gamma_t$ ), explore  $L_w[l_t \dots t - 1]$ .
3. If  $I_o^m \supset I_w^t$ , compare  $I_o^m$  and  $\gamma_t$ . If  $\gamma_t \supset I_o^m$ ,  $I_w^t$  must be the unique interval which can be covered by  $I_o^m$ . Therefore,  $I_w^t$  is the result and the search stops. The problem reduces to a



checking of  $L_o[1 .. m - 1]$  against  $L_w[1 .. t - 1]$  with  $\Gamma_w[1 .. k]$  to be used for control, where  $k$  is the position prior to  $\gamma_t$  in  $\Gamma_w$ . If  $\gamma_t = I_o^m$ , we will return all those intervals between  $l_t$  and  $r_t$ , including both  $l_t$  and  $r_t$ . The search also stops and the problem reduces to a checking of  $L_o[1 .. m - 1]$  against  $L_w[1 .. l_t - 1]$  with  $\Gamma_w[1 .. k]$ . If  $\gamma_t \in I_o^m$ , we will search part of  $\Gamma_w$  to the right of  $\gamma_t$  to find the right-most interval  $\gamma_f$  covered by  $I_o^m$ . Then, return all the intervals between  $l_f$  and  $r_f$ , including  $l_f$  and  $r_f$ , which allows us to reduce the problem to check  $L_o[1 .. m - 1]$  against  $L_w[1 .. l_f - 1]$  with  $\Gamma_w[1 .. g]$ , where  $g$  is the position prior to  $\gamma_f$  in  $\Gamma_w$ .

4. If  $I_o^m \subset I_w^t$ , the above data structure cannot be utilized to speed up the search. Thus, this case will be handled in the same way as described for *conjB*( ).

**Example 3** To see how the LCAs can be used to skip over useless checkings, we check several single intervals against  $L_a$  in Fig. 8 to show the working process.

1. Assume that  $I = [5, 7]$  is compared with  $I_5 = [11, 11]$  in  $L_a$ . Since  $[5, 7]$  is to the left of  $[11, 11]$ , we will compare  $[5, 7]$  with  $\gamma_5 = [8, 14]$  and  $[5, 7] \not\subset [8, 14]$ . So we will check  $[5, 7]$  against  $L_a[1 .. l_5 - 1] = L_a[1 .. 3]$  in a next step, instead of the sequence containing all the intervals to the left of  $I_5$ .
2. Assume that  $I = [10, 13]$  is compared with  $I_4 = [8, 8]$  in  $L_a$ . Since  $[10, 13]$  is to the right of  $[8, 8]$ ,  $[10, 13]$  and  $\gamma_4 = [8, 14]$  will be compared and  $[10, 13] \subset [8, 14]$ . So, in the next step, we will check  $[10, 13]$  against  $L_a[4 + 1 .. r_5] = L_a[5 .. 5]$ , not the sequence containing all the intervals to the right of  $I_4$ .
3. Assume that  $I = [10, 13]$  is compared with  $I_5 = [11, 11]$  in  $L_a$ . We have  $[10, 13] \supset [11, 11]$ . However,  $[10, 13] \subset \gamma_5 = [8, 14]$ . It shows that  $[11, 11]$  is the only interval in  $L_a$ , which can be covered by  $[10, 13]$ . No further search is necessary.
4. Assume that  $I = [8, 14]$  is compared with  $I_4 = [8, 8]$  in  $L_a$ . We have  $[8, 14] \supset [8, 8]$ . But we also have  $[8, 14] = \gamma_4$ . Then, we know immediately that only the intervals in  $L_a[l_4 .. r_4] = L_a[4 .. 5]$  can be covered by  $[8, 14]$ .  $\square$

By Example 3, we can clearly see that LCAs are quite useful to speed up the operation. However, all of them should be efficiently recognized. We will discuss this in the next Section.

## 5. CONCLUSION

In this paper, a new index structure is discussed. It associates each word  $w$  with a sequence of intervals, which partition the inverted list  $\mathcal{X}(w)$  into a set of disjoint subsets, and transform the evaluation of conjunctive queries to a series of checkings of interval containment. Especially, the intervals can be organized into a compact interval graph, which enables us to skip over any useless checking of interval containment. On average, to evaluate a two-word query, only  $O(\log n)$  time is required, where  $n$  is the number of documents. This is much more efficient than any existing method for set intersection. Also, how to maintain such an index is described in great detail. Although the index is of a more complicated structure,

the cost of maintaining it in the cases of addition and deletion of documents is (theoretically) comparable to the inverted file. Extensive experiments have been conducted, which show that our method outperforms the inverted file and the signature tree by an order of magnitude or more.

## REFERENCES

- [1] V.N. Anh and A. Moffat: Inverted index compression using word-aligned binary codes, *Kluwer Int. Journal of Information Retrieval* 8, 1, pp. 151-166, 2005.
- [2] J. Barbay, A. López-Ortiz, T. Lu, A. Salinger: An experimental investigation of set intersection algorithms for text searching, *ACM Journal of Experimental Algorithmics* 14: (2009).
- [3] P. Bille, A. Pagh, and R. Pagh. Fast-Evaluation of Union-Intersection Expression. In ISAAC, pp. 739-750, 2007.
- [4] G.E. Blelloch and M. Reid-Miller. Fast Set Operations using Treaps. In ACM SPAA, pp. 16-26, 1998.
- [5] Y. Chen, Y.B. Chen: On the Signature Tree Construction and Analysis, *IEEE TKDE*, Sept. 2006, Vol.18, No. 9, pp 1207 – 1224.
- [6] Y. Chen: Building Signature Trees into OODBs, *Journal of Information Science and Engineering*, 20, 275-304 (2004).
- [7] Y. Chen and Y.B. Chen: An Efficient Algorithm for Answering Graph Reachability Queries, in *Proc. 24th Int. Conf. on Data Engineering (ICDE 2008)*, IEEE, April 2008, pp. 892-901.
- [8] Y. Chen and Y.B. Chen: Decomposing DAGs into spanning trees: A new way to compress transitive closures, in *Proc. 27th Int. Conf. on Data Engineering (ICDE 2011)*, IEEE, April 2011, pp. 1007-1018.
- [9] K.D. Demaine, A. López-Ortiz, and J.I. Munro: Adaptive set intersections, unions, and differences, in *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, Philadelphia, 743-752, 2000.
- [10] B. Ding, A.C. König, Fast set intersection in memory, *Proc. of the VLDB Endowment*, v.4 n.4, p.255-266, January 2011.
- [11] C. Faloutsos: Access Methods for Text, *ACM Computing Surveys*, vol. 17, no. 1, pp. 49-74, 1985.
- [12] C. Faloutsos and R. Chan: Fast Text Access Methods for Optical and Large Magnetic Disks: Designs and Performance Comparison, *Proc. 14th Int'l Conf. Very Large Data Bases*, pp. 280-293, Aug. 1988.
- [13] D.E. Knuth, *The Art of Computer Programming*, Vol. 3, Massachusetts, Addison-Wesley Publish Com., 1975.
- [14] J. Zobel and A. Moffat: Inverted Files for Text Search Engines, *ACM Computing Surveys*, 38(2):1-56, July 2006.
- [15] J. Zobel, A. Moffat, and K. Ramamohanarao: Inverted Files Versus Signature Files for Text Indexing, in *ACM Trans. Database Syst.*, 1998, pp.453-490.

# Absolute Bandwidth Scheduling via Group-based Partitioned Proportional Share Scheduling and Dynamic Weight Management on Varying-Speed Multiprocessors

Sangwon Shin, Shakaiba Majeed, and Minsoo Ryu\*

Department of Computer Science and Engineering, Hanyang University, Seoul, Korea  
 {swshin, shakaiba}@rtcc.hanyang.ac.kr, msryu@hanyang.ac.kr

**Abstract** - For the past few years there has been an increase in the use of compute intensive applications running on high-performance embedded systems based on multi-core platforms. These applications demand an absolute share of CPU bandwidth to guarantee a certain level of QoS (Quality of Service) and fulfil their timing constraints. Unfortunately, traditional proportional share or priority scheduling algorithms employed in general purpose operating systems are not able to provide an absolute share of processor resources for such time sensitive tasks. In this paper, we present an absolute bandwidth scheduling scheme which aims at providing an absolute share of CPU bandwidth to groups of soft real-time tasks regardless of the work load conditions and varying speeds of CPU. The proposed scheme provides a mechanism of CPU bandwidth allocation for groups of soft-real time tasks by dynamically changing the overall weight of the group while maintaining the proportion of share of each task in the group. Our proposed approach works on top of a traditional proportional share scheduler and does not require any modifications to the kernel layer. To demonstrate the effectiveness and the correctness of our scheme, we have implemented a prototype using Linux cgroups and the existing completely fair scheduler (CFS). A series of experiments are conducted to prove that each soft real-time task in the group maintains its required absolute bandwidth.

**Keywords:** Absolute bandwidth scheduling, group-based proportional share scheduling, dynamic weight management, QoS, soft real-time.

## 1 Introduction

For the past few years there has been an increase in the use of compute intensive applications such as multimedia processing, online gaming and data encryption/decryption running on high-performance embedded systems based on multi-core platforms. These applications demand an absolute share of CPU bandwidth to guarantee a certain level of QoS (Quality of Service) and fulfil their timing constraints. For instance, a video application may require 30% of processor bandwidth at 1GHz to decode 30 frames per second for smooth playback.

Unfortunately, traditional proportional share or priority scheduling algorithms employed in general purpose operating systems are not able to provide an absolute share of CPU resources for time sensitive tasks. The proportional share algorithm aims at providing relative fairness to the tasks proportional to their weights, for distributing CPU bandwidth. As a result, the relative share of each task decreases as the system load increases.

The second hindrance in allocating a guaranteed share of CPU bandwidth is the assumption of fixed speed or performance of each CPU core. In a real world system, however, the speed of CPU may vary with new tasks being dynamically added to the CPU and the underlying DVFS (dynamic voltage and frequency scaling) policy. As a result, each task may generate a different utilization value depending upon the current speed of the CPU. Consider for example that a soft real-time task such as an HVEC (High Efficiency Video Coding) decoder demands 60% of CPU utilization on ARM Cortex A9 processor running at maximum frequency of 1.5 GHz. If the task is scheduled on a CPU which is running at 1.0 GHz, HVEC may not meet its performance metrics because CPU utilization in that case should be 90%. This may adversely affect the resource allocation problem especially in multi-core environments where CPUs can run at varying speeds and task migration occurs frequently. Hence, to provide efficient CPU bandwidth allocation to tasks, the individual as well as overall computation performance of all the CPU resources in a system under a certain condition must be taken into account.

In this paper we propose an absolute bandwidth scheduling (ABS) scheme to accomplish absolute bandwidth guarantees on top of an existing partitioned proportional share scheduler. The proposed scheme performs two important functions. First, it partitions tasks running on each processor into two groups, absolute bandwidth tasks that require absolute bandwidth guarantees and proportional share tasks that require traditional proportional share services. Second, it dynamically adapts the weights of absolute bandwidth task groups to satisfy their absolute bandwidth requirements taking into account the dynamic change in workload characteristics and varying processor speeds. We show through experimental evaluation that the proposed scheme

can efficiently achieve absolute bandwidth guarantees in conjunction with an existing proportional share scheduler.

The rest of this paper is organized as follows: section 2 gives a brief review of the related work, section 3 elaborates our proposed scheme. In section 4 we give an implementation of the scheme, section 5 explains our experimental set up and finally we conclude with section 6.

## 2 Related Work

Many researchers proposed resource reservation schemes to provide guaranteed allocation of resources to a group of tasks present in an application. For example, the processor capacity reserve scheme proposed by Mercer et al. [2] suggests using priority based scheduling to grant resource reservation to each application. Later a kernel module is used to keep track of the CPU usage of each application to implement the granted reservation. A similar approach, called ACTORS [3], reserves resources for each application and readjust the assignment by using feedback from the application. This solution gives user a comprehensive control over resource allocation but the technique requires non-trivial modification to the kernel.

To provide a constant share of CPU bandwidth regardless of the workload conditions a fixed share scheduling (FSS) policy has been proposed in [4]. FSS enables a traditional proportional share scheduler to provide constant share of CPU bandwidth by dynamically modifying the weights of soft real-time tasks under changing workload conditions. However, FSS assumes that the speed or performance of the CPUs remains fixed over time. Note that the dynamic voltage frequency scaling (DVFS) mechanism implemented in modern computing systems scales up or down the speed of CPUs as new tasks arrive and/or depart. Hence, to provide absolute CPU bandwidth allocation to tasks, dynamic CPU speeds must be taken into account.

## 3 Absolute Bandwidth Scheduling

There are two general approaches to multiprocessor scheduling, global scheduling and partitioned scheduling. The global scheduling approach maintains single task queue and schedules tasks selecting eligible tasks guided by a global scheduling policy and assigning them to appropriate processors. On the other hand, the partitioned scheduling approach maintains a separate task queue for each processor and schedules tasks in a way similar to distributed scheduling. In a partitioned scheduling system, tasks are first assigned to processors and each processor runs a separate scheduler instance to schedule them independently of other processors.

In this work, we consider partitioned scheduling systems with proportional share scheduling support. In order to satisfy absolute processor bandwidth requirements in such partitioned proportional share scheduling systems, we present

a group-based proportional scheduling scheme with dynamic weight management as described below

### 3.1 Group-based Proportional Share Scheduling

The goal of group-based proportional scheduling schemes is to meet the performance requirements of a group of tasks within applications. The key idea of group-based proportional share scheduling is to allocate resources to task groups relative to their weights such that the share of each group is defined by a proportional share with respect to the other groups present in the system. The share of each task within a group is defined by a proportional share of its parent group [4]. For example, let  $W_{g_i}$  be the weight of group  $g_i$  and let  $G$  be the set of all active groups at time  $t$ , then the share  $s_{g_i}(t)$  of a group  $g_i$  at time  $t$  is defined as below.

$$s_{g_i}(t) = \frac{W_{g_i}}{\sum_{j \in G} W_{g_j}}$$

Let  $w_{\tau_i}$  be the weight of task  $\tau_i$  and  $T$  be the set of all active tasks included in group  $g_i$  at time  $t$ . The share  $s_{\tau_i}(t)$  of a task  $\tau_i$  with weight  $w_{\tau_i}$  at time  $t$  is defined as below.

$$s_{\tau_i}(t) = s_{g_i}(t) \times \frac{w_{\tau_i}}{\sum_{j \in T} w_{\tau_j}} \quad (1)$$

### 3.2 Dynamic Weight Management for Groups

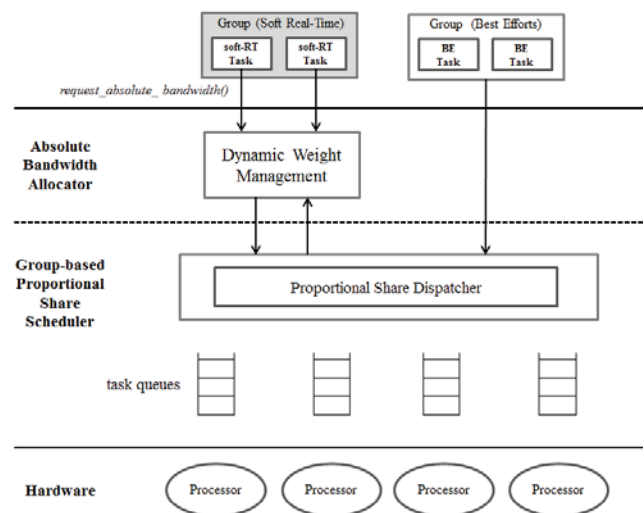


Figure 1. The absolute bandwidth scheduling model.

In order to obtain absolute bandwidth allocation guarantees from the existing proportional share scheduler, we propose to add an absolute bandwidth allocator on top of it with a minimal impact on the existing system architecture. The primary goal of the absolute bandwidth allocator is to

receive absolute bandwidth requirements from the soft real-time tasks and changing the weights of groups of tasks by examining the run queue of each processor and its current speed.

Figure 1 shows the proposed absolute bandwidth scheduling model with an absolute bandwidth allocator running on top of an existing group-based proportional share scheduler on a multiprocessor platform. The platform consists of  $m$  processors  $P = \{P_1, P_2, \dots, P_m\}$ . Each processor has identical maximum processing speed  $F_{\max}$  but can be operating on varying speeds depending on the workload conditions and DVFS (dynamic voltage and frequency scaling) mechanism. Let us denote the current frequency of a processor by  $F_{p_i}$ .

For the proposed model, we define a set of active tasks  $T = \{\tau_1, \tau_2, \dots, \tau_m\}$  running on the multiprocessor platform and divide these tasks into two groups such that

$$T = T_{ab} \cup T_{ps}$$

where  $T_{ab}$  is a group of soft real-time tasks that require absolute bandwidth guarantees and  $T_{ps}$  is a group of best effort tasks that require proportional share bandwidth guarantees.

Each processor  $P_i$  has a separate task run queue  $T_{p_i}$  such that

$$T_{p_i} = T_{abp_i} \cup T_{psp_i}$$

where  $T_{abp_i}$  is a group of soft real-time tasks allocated to processor  $P_i$  and  $T_{psp_i}$  is a group of best effort tasks allocated to processor  $P_i$ . The best effort tasks are allocated bandwidth shares in proportion to their weights in accordance with the existing group-based proportional share scheduling scheme. We denote the weight of a best effort task  $\tau_i \in T_{psp_i}$  by  $w_{\tau_i}$  so that the overall weight of the best effort group  $T_{psp_i}$  scheduled at  $P_i$  is given by

$$W_{psp_i} = \sum_{\tau_i \in T_{psp_i}} w_{\tau_i}$$

The goal of absolute bandwidth scheduling is to provide an absolute bandwidth to soft real-time tasks. This can be achieved by receiving a absolute bandwidth requirement from each task and then dynamically changing the weights of each soft real-time group to maintain the requested bandwidth requirements. We define the absolute bandwidth request by any software task  $\tau_i \in T_{abp_i}$  as CPU utilization  $U_i$  required from a CPU when running at  $F_{\max}$ . In the proposed model each task with in  $T_{ab}$  may put such request through an application programming interface (API) request\_absolute\_bandwidth() as shown in Figure 2.

For a group of soft real-time tasks assigned to a processor  $P_i$ , the absolute CPU utilization  $U_{abp_i}$  is defined by

$$U_{abp_i} = \frac{F_{\max}}{F_{p_i}} \times \sum_{\tau_i \in T_{abp_i}} U_i \quad (2)$$

It is worth noting that when dispatching the tasks to a specific processor it is necessary that the sum of bandwidth requests  $\sum_{\tau_i \in T_{abp_i}} U_i$  cannot be greater than 1. However, the resulting  $U_{abp_i}$  can be greater than 1 in case when the current operating frequency  $F_{p_i}$  is less than the maximum speed  $F_{\max}$  of CPU. This will enable DVFS to increase the operating frequency of CPU as required.

Having obtained the absolute CPU utilization  $U_{abp_i}$  of a soft real-time group, the weight of the group can be obtained as

$$W_{abp_i} = \frac{U_{abp_i} \times \sum_{\tau_i \in T_{psp_i}} w_{\tau_i}}{1 - U_{abp_i}} \quad (3)$$

The group weight obtained from Equation (3) guarantees that each soft real-time task gets an absolute bandwidth allocation and the other best effort tasks present on the run queue are assigned a proportional share of the CPU bandwidth.

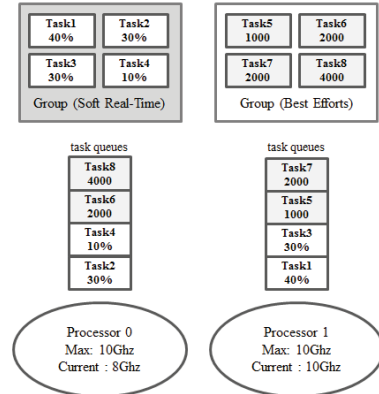


Figure 2. Examples of absolute bandwidth scheduling

Figure 2 shows an example of the proposed absolute bandwidth scheduling model. Consider two sets of tasks  $T_{ab} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$  and  $T_{ps} = \{\tau_5, \tau_6, \tau_7, \tau_8\}$  running on a dual-processor platform. Maximum frequency of each processor,  $F_{\max}$  is 10 GHz while current frequency of  $P_0$  is  $F_0 = 8$  GHz and current frequency of  $P_1$  is  $F_1 = 10$  GHz .

Soft real task  $\tau_2$  and  $\tau_4$  are allocated to processor  $P_0$  and have absolute bandwidth requirements of  $U_2 = 0.3$  and  $U_4 = 0.1$  at 10 GHz, and best effort task  $\tau_6$  and  $\tau_8$  are allocated to processor  $P_0$  and have weights  $w_{\tau_6} = 2000$  and  $w_{\tau_8} = 4000$ . By applying Equation (2) to group of soft real-

time tasks, we have  $U_{abp_0} = \left(\frac{10 \times (0.3+0.1)}{8}\right) = 0.5$  and by applying Eq. (3), we get  $W_{abp_0} = \left(\frac{8 \times 0.5 \times (2000+4000)}{8-8 \times 0.5}\right) = 6000$ . By using this weight value for the group we obtain the share of each soft real-time task present in the group as desired. We can verify the result by applying Equation (1):  $s_{\tau_2} = \frac{6000}{6000+6000} \times \frac{0.3}{0.3+0.1} = 0.375$  and  $s_{\tau_4} = \frac{6000}{6000+6000} \times \frac{0.1}{0.3+0.1} = 0.125$  and  $s_{\tau_6} = \frac{6000}{6000+6000} \times \frac{2000}{2000+4000} = 0.167$  and  $s_{\tau_8} = \frac{6000}{6000+6000} \times \frac{4000}{2000+4000} = 0.333$ .

Soft real task  $\tau_1$  and  $\tau_3$  are allocated to processor  $P_1$  and have absolute bandwidth requirements of  $U_1 = 0.4$  and  $U_4 = 0.3$  at 10 GHz, and best efforts task  $\tau_5$  and  $\tau_7$  are allocated to processor  $P_1$  and have weights  $w_{\tau_5} = 1000$  and  $w_{\tau_7} = 2000$ . By applying Equation (2) to group of soft real-time tasks, we have  $U_{abp_1} = \left(\frac{10 \times (0.4+0.3)}{10}\right) = 0.7$  and by applying Eq. (3), we get  $W_{abp_1} = \left(\frac{10 \times 0.7 \times (1000+2000)}{10-10 \times 0.7}\right) = 7000$ . Using this weight value we can verify the share of each soft real-time task by applying Equation (1):  $s_{\tau_1} = \frac{7000}{7000+3000} \times \frac{0.4}{0.4+0.3} = 0.4$  and  $s_{\tau_3} = \frac{7000}{7000+3000} \times \frac{0.3}{0.4+0.3} = 0.3$  and  $s_{\tau_5} = \frac{7000+3000}{7000+3000} \times \frac{1000}{1000+2000} = 0.1$  and  $s_{\tau_7} = \frac{7000+3000}{7000+3000} \times \frac{2000}{1000+2000} = 0.2$ .

## 4 Implementation

We have implemented a prototype of the proposed scheme on Linux kernel 3.18.3 using control groups (cgroups). Cgroups provide a mechanism to aggregate or partition tasks into hierarchical groups categorized by their peculiar behavior primarily for the purpose of efficient resource management among tasks.

To exploit the benefits of cgroups and to efficiently manage CPU resources we implemented the absolute bandwidth allocation scheduling using CPU subsystem of cgroups. However, we add two new parameters to the existing CPU subsystem. The first parameter `cpu.softRT` is used to classify each task on the system as a soft real-time or best effort task. The other parameter `cpu.absoluteBW` is implemented as a structure and indicates absolute bandwidth requirement of each task in a group.

In order to provide absolute bandwidth allocation to soft real-time tasks we implemented dynamic weight management for groups on top of the existing Linux's completely fair scheduler (CFS) [8]. Our addition of parameters in CPU subsystem of cgroups and the process of dynamic weight management does not modify the existing kernel implementation and has no impact on the existing system if newly defined parameters are not used.

## 5 Experimental Evaluation

We evaluated the effectiveness and correctness of the implementation of absolute bandwidth scheduling scheme by conducting a series of experiments. These experiments were performed on an Intel Core i5-4690 CPU which has four cores with 3.50 GHz maximum speed, running on Linux 3.18.3. Each set of experiments was conducted 10 times to ensure that the experiments and their results are repeatable.

In the first set of experiments, we observed the CPU utilization of soft real-time tasks under varying workload conditions. We used four target tasks  $T_{ab} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$  with absolute bandwidth requirement  $U_{ab} = \{60\%, 60\%, 50\%, 40\%\}$  at 3.5 GHz. Each task used in this experiment is a busy-waiting task which consumes 100% of CPU utilization when executed alone on a processor. Each processor was running on its maximum speed. We started adding new best efforts tasks at 1000 millisecond and noticed that the actual CPU utilization of target tasks is maintained to their demanded absolute bandwidth even though the number of running best efforts tasks on each CPU was dynamically changing. Notice that since the processors are running on their maximum frequency hence the actual CPU utilization and the absolute bandwidth utilization demanded by tasks is same. Figure 3 shows that actual utilization of target soft real-time task is maintained around absolute bandwidth even though the number of tasks varies dynamically.

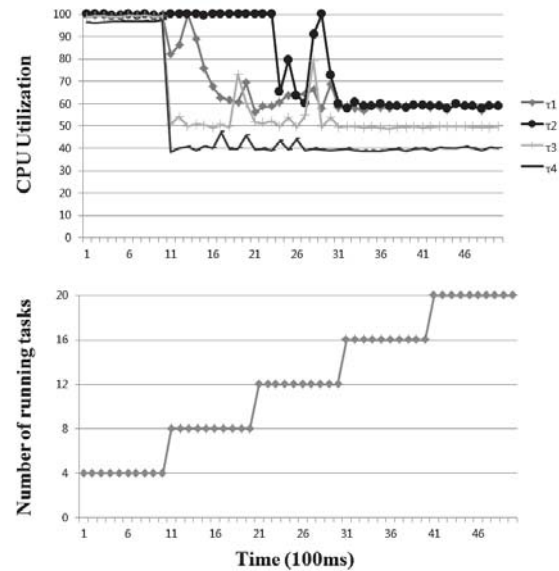


Figure 3. CPU utilization of soft real-time tasks, number of running tasks in the proposed scheme

In the second set of experiments, we observed the CPU utilization and job completion time of target soft real-time tasks under the assumption of varying CPU speeds. We created four soft real-time tasks  $T_{ab} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$  with absolute bandwidth requirement  $U_{ab} = \{100\%, 50\%, 30\%, 30\%\}$  at 3.5 GHz. Each task was

added to a separate group destined to schedule on a particular CPU using processor affinity. To simulate that two of the CPUs, CPU1 and CPU3 are running at slower speeds, we used a value of loop counter inversely proportional to the desired frequency in the busy-waiting loop of soft real-time tasks, such that the current frequency of CPU1 and CPU4 is 3.5 GHz and 1.75 GHz for CPU1 and CPU3.

Table 1 shows that for CPU1 and CPU3, the actual utilization of soft real-time tasks at simulated current frequency is increased as a result of applying Equation (2). We then used this utilization value to recalculate the weights of the soft real-time task groups scheduled at CPU1 and CPU3 to maintain their initial absolute bandwidth request. The last row in Table 1 shows the relationship between the absolute bandwidth requirement and job-completion time of each task; for a given operating frequency, the smaller the absolute bandwidth request the longer it takes to complete the task.

Table 1. CPU utilization and job completion time of each soft real-time groups

|                                      | Group 1  | Group 2   | Group 3  | Group 4   |
|--------------------------------------|--|---|--|---|
| <b>Processor Information</b>         | Processor 0<br>$F_{max} = 3.5$ GHz<br>$Fp_0 = 3.5$ GHz | Processor 1<br>$F_{max} = 3.5$ GHz<br>$Fp_1 = 1.75$ GHz | Processor 2<br>$F_{max} = 3.5$ GHz<br>$Fp_2 = 3.5$ GHz | Processor 3<br>$F_{max} = 3.5$ GHz<br>$Fp_3 = 1.75$ GHz |
| <b>Absolute Bandwidth Request(%)</b> | 100  | 50  | 30   | 30  |
| <b>Actual Utilization(%)</b>         | 100  | 100   | 30.3   | 60.8  |
| <b>Job Complete Time(ms)</b>         | 2459.2   | 5014  | 8184.7   | 8030.9  |

## 6 Conclusions

In this paper, we have presented an absolute bandwidth scheduling scheme which guarantees absolute bandwidth for a soft real-time tasks and proportional bandwidth allocation to best effort tasks. We implemented absolute bandwidth scheduling using the notion of group-based scheduling and by dynamically changing the weights of groups of soft real-time tasks on top of the existing Linux CFS scheduler. We demonstrated with our experiments that the soft real-time tasks maintain their required absolute bandwidth when more tasks were added on the system and even when the current speed of the underlying CPUs was changed.

## Acknowledgment

This work was supported partly by Seoul Creative Human Development Program (HM120006), partly by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (NRF-2011-0015997), and partly the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the C-ITRC(Convergence Information Technology Research Center) (IITP-2015-H8601-15-1005) supervised by the IITP(Institute for Information & communications Technology Promotion).

## References

- [1] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, pp. 600-625, 1996.
- [2] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: Operating system support for multimedia applications," in *Multimedia Computing and Systems, 1994., Proceedings of the International Conference on*, 1994, pp. 90-99.
- [3] E. Bini, G. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, *et al.*, "Resource management on multicore systems: The ACTORS approach," *IEEE Micro*, vol. 31, pp. 72-81, 2011.
- [4] H. Kim, H. Yoon, P. Wu, and M. Ryu, "Fixed Share Scheduling via Dynamic Weight Adjustment in Proportional Share Scheduling Systems."
- [5] P. Holman and J. H. Anderson, "Group-based pfair scheduling," *Real-Time Systems*, vol. 32, pp. 125-168, 2006.
- [6] J. Kay and P. Lauder, "A fair share scheduler," *Commun. ACM*, vol. 31, pp. 44-55, 1988.
- [7] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *International Journal of Computer Aided Engineering and Technology*, vol. 6, pp. 440-459, 2014.
- [8] C. S. Pabla, "Completely fair scheduler," *Linux Journal*, vol. 2009, p. 4, 2009.

# Warehouse Pick Path Optimization Algorithm Analysis

Ryan Key

Edinboro University of Pennsylvania  
rk096065@scots.edinboro.edu

Anurag Dasgupta

Edinboro University of Pennsylvania  
adasgupta@edinboro.edu

## ABSTRACT

Warehouse operational costs are heavily influenced by the efficiency in which workers are able to traverse the warehouse and gather items on orders around the warehouse that must be shipped to customers; this action accounts for over 50% of warehouse operations expenses. The act of traversing the warehouse is greatly optimized by following a designated pick path; however, algorithms for pick path generation are complex and heavily unexplored by the industry. Generating pick paths involves solving two common place graph theory problems: the shortest path problem and the traveling salesperson problem. We will analyze algorithms used for solving both of these problems and discuss the feasibility of generating pick paths through the use of the algorithms. We also introduce a simplified implementation to illustrate the viability of the described approaches.

## KEY WORDS

Warehouse Pick Path Optimization, Traveling Salesperson Problem, Shortest Path, Algorithm Comparison

## 1. Introduction

A common goal of nearly all businesses is to reduce man-hours and increase the overall profit margin of the business. In warehouse related businesses, optimizing the efficiency of order picking can lend itself to great reductions in the time it takes orders to ship out, as well as, improving the overall effectiveness of its workers. Of all costs associated with warehouse operations, 55-65% of the operational funds are allocated towards order picking [1] [2], showing the importance of optimizing this phase of the warehouse process.

To better understand the problem at hand, we will now describe a common scenario in a warehouse and show how order picking fits into the mix. First, a warehouse is comprised of 3 primary components: receiving, storage, and shipping [2]. Receiving is responsible for checking things into the warehouse; this is the entryway for all items into the warehouse stock. Once items have been received, they must be put away and stored. Storage can be of any form, although large shelving units, gaylords, and/or pallets are traditional options. The storage area of a warehouse is quite important and should be well organized to create an advantageous environment for order pickers. The final area, shipping, is similar to receiving. This is the exit for all items leaving the warehouse; any items ordered by customers of the warehouse must pass through this area before arriving at

the customer's location [2]; a customer of the warehouse could be larger entities such as the operation of a large franchise storage warehouse, or a single customer that is ordering products from an online store.

After understanding the layout of a warehouse, we must look at the tasks performed by a warehouse. Nearly all activities at a warehouse are centered on receiving orders from customers. As an order comes into the warehouse, an individual in the warehouse becomes responsible for the order; this is the *order picker*. The order picker is responsible for gathering all items on the order from around the warehouse, also known as *picking*, and then placing them in the shipping area. Once in shipping, the order will be packaged in a box or pallet and shipped to the customer. The picking process can be time consuming and by far is the biggest operational expense of any warehouse [2].

Picking items for orders is the most costly part of the process, because order pickers must traverse the warehouse layout to find all items on the order, starting and ending in the shipping area; this is a *pick path*. For example, an order picker will print an order from the shipping area with 10 items on it. If there is a 20,000 square foot warehouse, it can be expected to have at least 500 unique storage locations throughout the warehouse defining where items are stored. This means, the order picker must determine an efficient route through the 500 locations to get to the 10 locations identified on the order, where any location can be travelled to from any other location; we know this because any practical warehouse layout will not create any isolated, or pendant, storage area. After the order picker has found all 10 items in the warehouse, they must then bring the items back to the shipping area so the order can be shipped and the order picking process can be started fresh.

Now to analysis this problem in terms of graph theory, finding a pick path involves solving two of the most common problems in graph theory; the Travelling Salesperson Problem (TSP) and the shortest path problem. The TSP is the problem of finding the shortest tour through  $n$  cities that visits every city exactly once, starting and ending in the same city [3]. Where in the case of pick path optimization, we want to visit every location on the order exactly once, generally starting and ending in shipping.

The shortest path/route problem comes in five varieties, two of which pertain to the work accomplished by this paper. The first being, finding the shortest path between some vertex,  $x$ , and some other vertex in the graph,  $y$  [4]. Depending on the implementation and algorithms used to optimize warehouse pick paths, this scenario will come

into play such as, wanting to make it from some last location on the order back to the single shipping location. More often, it will be the case where the following shortest path is sought: the shortest path between some vertex and all others [4]. This is the shortest path most often sought during optimized pick path generation, as distances between the current vertex and all other locations on an order commonly need to be found.

To summarize the TSP/Shortest Path problem encountered when solving the pick path problem: we are solving a TSP between all locations on an order; starting and ending in the shipping area (which is also defined by a vertex). Then as the TSP is being solved, at each location we encounter the further difficulty of finding the shortest paths between the current node and all remaining locations on the order. This occurs because we are solving a TSP which assumes a complete graph where all nodes are connected with one edge between them; however, this is not guaranteed to be the case in the warehouse layout. We are guaranteed all locations will be reachable from any location in the graph, so we must define the shortest path to each of these vertices. After finding these shortest paths, we can then treat the problem like a normal TSP, where all edges defining the path between nodes are treated as one edge with one minimum distance associated with it.

Section 4 of this paper will focus on the algorithms and enhancements that can be used to find optimal pick paths by solving the TSP and shortest path problem. Section 5 will offer insight into an implementation for finding an optimal pick path.

## 2. Related Work

There have been hundreds of papers published and dozens of algorithms developed around solving the shortest path problem alone, as a large number of mathematical optimization problems are mathematically equivalent to the shortest path problem [4]. In the same respect, the TSP has been analyzed by dozens of professions, researched to no end, and proved to be a member of the NP-Complete problem set, with numerous heuristics developed that present polynomial time solutions within a fair degree of accuracy [3].

In contrast, there has been limited research completed around warehouse efficiency, and more specifically pick path optimization. There is some degree of research related to the business operations of warehouses [2]; however, there is almost no research related to the algorithms required to solve the problems of optimizing warehouse operations. This is the gap in research we have aimed to fill throughout the course of this paper.

## 3. Contribution

With an astonishing amount of research in solving shortest path problems and the travelling salesperson problem, we aim to explore popular algorithms for

solving these problems, taking a closer look at how each algorithm works and the practicality of generating optimal pick paths with these algorithms. We will look at the role of each algorithm in solving the warehouse pick path optimization problem and evaluate the characteristics of the algorithm. We will look at the timing complexity of these algorithms, as well as, the flaws and potential concerns for implementing each algorithm.

We also introduce a basic implementation used for solving the warehouse pick path problem. In this regard, we focus on the components of the implementation and the improvements that should be made before using the algorithm to generate optimized pick paths.

## 4. Algorithms

There are a plethora of shortest path and TSP algorithms available, this section will focus on a handful of popular algorithms used for solving these problems. We describe each algorithm and the way in which it works. We then make mention of its time complexity, closeness to the optimum solution, and give a brief analysis on whether the algorithm is practical for usage in generating optimized pick paths.

### 4.1 Shortest Path Algorithms

This section will focus on algorithms used to find the shortest path between some vertex and all others in the graph; algorithm enhancements will also be considered.

#### 4.1.1 Dijkstra's

Dijkstra's algorithm is used to find the shortest distance between some starting vertex and all other vertices in the graph [5]. Dijkstra's algorithm is quite popular for its performance, with a worst case performance of  $O(|E| + |V| \log|V|)$ , where  $E$  = number of edges and  $V$  = number of vertices [3]. The algorithm is also easy to alter so that Dijkstra's will not only return the distance of the shortest path to each vertex, but also the path to traverse.

In pick path optimization, Dijkstra's is quite useful as it can be used at each location to find the shortest distance between this location and all remaining locations on the order. This is quite practical and is often exactly what we need to solve in the process of generating an optimal pick path in the midst of solving the TSP portion of the problem.

Section 4.1.3 further elaborates on enhancements that can be made to improve Dijkstra's algorithm.

#### 4.1.2 Floyd's

Floyd's algorithm is used to find the all-pairs shortest paths, meaning that in one run Floyd's can find the shortest path between all vertices on a graph [3]. This can be seen as extremely advantageous to Dijkstra's in finding pick paths, because the algorithm can be run once before solving the TSP end of the problem. By running the algorithm once, and storing the result to be referenced throughout the solving of the TSP, we are able to greatly



reduce the time spent determining the distance between the current location and all other locations on the order; however, we do not believe Floyd's will always be more efficient than running Dijkstra's at each location while solving the TSP, considering its worst case performance is  $O(|V|^3)$  [5].

#### 4.1.3 Dijkstra Enhancements

The reliability, popularity, and speed of Dijkstra's makes it a heavily implemented and researched algorithm, especially in the realm of map routing and GPS programming. Under this umbrella, there have been numerous algorithm enhancements suggested [6]. These enhancements can be used to potentially greatly reduce the time it takes Dijkstra's to run.

##### 4.1.3.1 Subgraph Partitioning

One of the most reasonable enhancements for pick path optimization is the idea of partitioning a graph into a subgraph, where the subgraph contains a limited number of unused/untraversed vertices [7]. For example, if a GPS were determining an optimal route from Washington, DC to New York City, it would not need to consider any vertices in California or Florida, as it is unreasonable to traverse that part of the graph when travelling from DC to NYC. This has a practical application to the pick path problem, as there is no need to look at the south side of the warehouse if no locations on an order pertain to that portion of the warehouse.

The holdup with the subgraph concept is the fact that there must be an algorithm run to determine what subgraph should be looked at and then form that subgraph [8]. This can be a costly operation and it can be difficult to predict what vertices should be dropped when developing a subgraph of the warehouse per order. It seems that in most circumstances, it would be more costly to determine what subgraph to send through Dijkstra's rather than simply running Dijkstra's algorithm on the entire graph.

##### 4.1.3.2 Bidirectional Search

Bidirectional search is an extension of Dijkstra's algorithm specifically targeting a two-node shortest path problem, when a starting vertex and target vertex are explicitly given [9]. When given these two points it is possible to create a mapping for the set of nodes and the set of edges such that Dijkstra's algorithm can be adapted to start running from the start vertex and the target vertex simultaneously, where each thread will meet in the middle of the path, reducing the time taken to find the shortest path between two points using Dijkstra's [9].

Bidirectional search initially seems like a practical enhancement for solving the pick path problem, although after considering the problem this is not the case. Dijkstra's algorithm is run in order to determine the shortest distance between the current location node and *all* other locations on the order. The pick path problem does not typically involve finding the shortest path between the current vertex and one other location vertex

in the graph, except if there is only one location on the order. It may be the case that some warehouses would find it advantageous to implement this Dijkstra's enhancement for this special case but, in general, order pickers are able to efficiently find their own path to orders with less than three locations on them [2]. This enhancement is not recommended for use in pick path generation algorithms.

#### 4.2 Travelling Salesperson Algorithms

This section will focus on algorithms used to find an exact optimized solution to the TSP, as well as, approximation algorithms that offer solutions within some guaranteed degree of closeness to the optimal solution.

##### 4.2.1 Exhaustive Search

The exhaustive search algorithm offers the only implementation that can produce the guaranteed shortest tour to the TSP every time. This algorithm searches through all permutations of tours, computing the distance travelled by each; if a new shortest tour is found it is stored as such until all possible tours have been checked [3]. Exhaustive search will always produce the shortest path because it is looking at every possible tour that could be taken. This is ideal in terms of a guaranteed shortest path; however, the performance of the algorithm is quite awful; having a time complexity of  $O(n!)$  [3] [4]. This is not a recommended approach for generating optimized pick paths.

##### 4.2.2 Nearest-Neighbor

The nearest-neighbor algorithm is a very simple algorithm to understand and implement. The algorithm starts at some random city, travelling to the city closest to the current city, until all cities have been visited. Once at the final city, come home. This algorithm cannot guarantee any degree of accuracy as to how close it will be to the optimum solution [3]. For this reason alone, we do not recommend using this when generating optimal pick paths.

##### 4.2.3 Multifragment-heuristic

The multifragment-heuristic algorithm works by looking at the edges of the graph, rather than vertices. The algorithm approaches the problem by creating a minimally weighted set of edges that makes each vertex in the graph of degree 2 [3].

The algorithm is as follows: first sort the set of edges by their weights and set the shortest distance set of edges to empty. Then, for the number of cities in the graph, add the shortest edge left in the set to the shortest distance set of edges, provided the addition of this edge does not make any vertex greater than of degree 2. After the loop has been completed, the shortest distance set of edges will contain the approximate shortest distance [3] [5].

This algorithm generally creates a more optimal result than the nearest-neighbor algorithm, but it also does not guarantee any degree of accuracy [3]. For this reason, this

algorithm is also not an ideal implementation for the pick path problem.

#### 4.2.4 Ant Colony Optimization

For solving the TSP, there are a number of different ant colony solution algorithms available, many of which are based on genetic algorithms. These algorithms are modeled after the natural ability of an ant colony to find the shortest path to their food source [10]. When ants arrive at decision points in their travels, they have no knowledge of what lies ahead of them or what distance must be travelled based on their decision. Since the choice is random, it can be expected that when presented with two directional choices (both ending at the same point), half the ants will go right and the other half will go left. Eventually, ants will be travelling to-and-from this location so ants will be choosing direction when headed both directions to-and-from the food source. As the ants travel, they release pheromones. After the ants have been travelling for a short time, the pheromone will accumulate on both paths; eventually the shorter path will have a much higher accumulation and this will begin to attract all the new ants to this path. Ants are able to discover the shortest paths between their food sources by measuring the amount of pheromone deposited on each decision path [10] [12].

One proposed Ant Colony algorithm for solving the TSP is the Ant Colony System (ACS) [10]. The algorithm's primary feature is the use of agents as *ants*. These ants work in a threaded, parallel fashion, simultaneously searching for a *good* solution to the TSP. The ants communicate on a global level, as well as, indirect communication through pheromone release on the edges. Each ant acts independently searching for a solution, using pheromones as a form of *memory* and making iterative improvements on its path selection. In the end it is proposed that the shortest path can be found by examining the pheromones left on each edge and selecting the maximal pheromone-weighted edges in order to form an optimal solution [10].

Dorigo presents numerous results of ACS tests in relation to other top-notch TSP algorithms [10]. Moreover, ACS presents accurate results for both small and large problems. The algorithm was able to produce the optimum tour in all tours with less than 100 cities in a minimal number of runs. For larger travelling salesperson problems (198 to 1577 cities), ACS was able to generate optimal paths within 3.5% error from the optimum [10]. This solution is recommended in terms of accuracy; however, it is not a practical implementation for many due to its degree of difficulty.

#### 4.2.5 Twice Around the Tree

The twice-around-the-tree algorithm is a minimum spanning tree-based algorithm [3]. These types of algorithms leverage the connection between Hamiltonian circuits and spanning trees, where a Hamiltonian circuit minus one edge produces a spanning tree [3].

The algorithm works by first constructing a minimum spanning tree of the graph. Then, starting at some random node, perform a walk around the spanning tree that was constructed (using a Depth First Search) and keeping track of vertices passed through. Then search the list of vertices that was generated; delete all repeats of nodes so that each vertex only appears once, except the start/end vertex. The start/end vertex should appear at both the beginning and end of the list. This produces a Hamiltonian circuit that is an approximation for the shortest path between all nodes [3].

This algorithm can be performed in polynomial time, although its exact timing depends on the implementation of the first step, where a minimum spanning tree is constructed. An MST can be constructed using any popular algorithm such as Prim's or Kruskal's [3] [4].

Another benefit of this approach is that, it is guaranteed that accuracy of the shortest tour generated by this algorithm is at most twice as long as the optimum tour. This algorithm is recommended based on its guaranteed upper bound and the fact that the algorithm is performed in polynomial time.

#### 4.2.6 Christofides' Algorithm

Christofides' algorithm works similarly to the twice-around-the-tree algorithm as it also works with minimum spanning trees. Christofides' utilizes more advanced implementations of graph theory to form a guaranteed lower cost tour than the previously discussed algorithms [3] [4] [5] [11].

Christofides' first creates a minimal spanning tree,  $T$ , using some known algorithm. Then create a set of all odd degree vertices,  $V$ . Then find a perfect matching,  $P$ , with the minimum weight of the graph over the vertices in  $V$ . This will create a set of minimally weighted edges without any common vertices from  $V$ . Then, add the edges from  $P$  and  $T$  to form a multigraph,  $M$ . A multigraph is simply a graph that allows parallel edges. Now form an Euler circuit from  $M$ , call it  $E$ . This will produce a circuit that visits every edge once. Now, remove edges that visit nodes more than once. This will create a Hamiltonian circuit, which as we previously defined is a solution to the TSP [3] [5] [11].

Christofides' algorithm can be performed in polynomial time and produces a minimal tour that is guaranteed to be within 1.5 times the optimum tour [11].

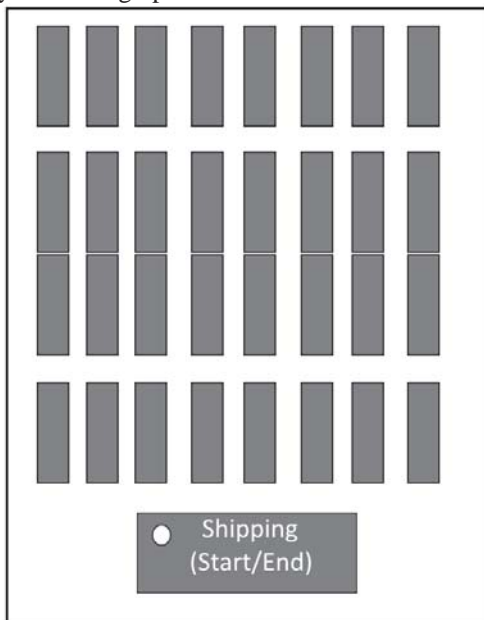
This algorithm is highly recommended for implementation in finding an optimal pick path.

## 5. Code

Throughout the research of this paper, a small case study project was developed to show that the algorithms described could be implemented to create a usable pick path generator. The application is a C# Windows Form application that provides the basic implementation to create a warehouse layout in the form of a graph, save it,

and then use it to generate an optimal pick path based on a handful of algorithms described above.

The initial step to use this application is the process of converting a warehouse layout into a graph. We will use *Figure 5.1* as an example throughout this section. First we must define the components of this layout. All whitespace in the layout represents aisles in the warehouse that can be traversed to travel around the warehouse from location to location. The large rectangular gray square in the bottom of the layout is the shipping area of the warehouse. As described in Section 1, this is typically the start and end point of the pick path. All remaining gray squares are storage locations in the warehouse such as shelving units or pallets. It is also important to note, distances are associated with each aisle and the shelving units; this comes into play as we move through the transformation of the layout into a graph.

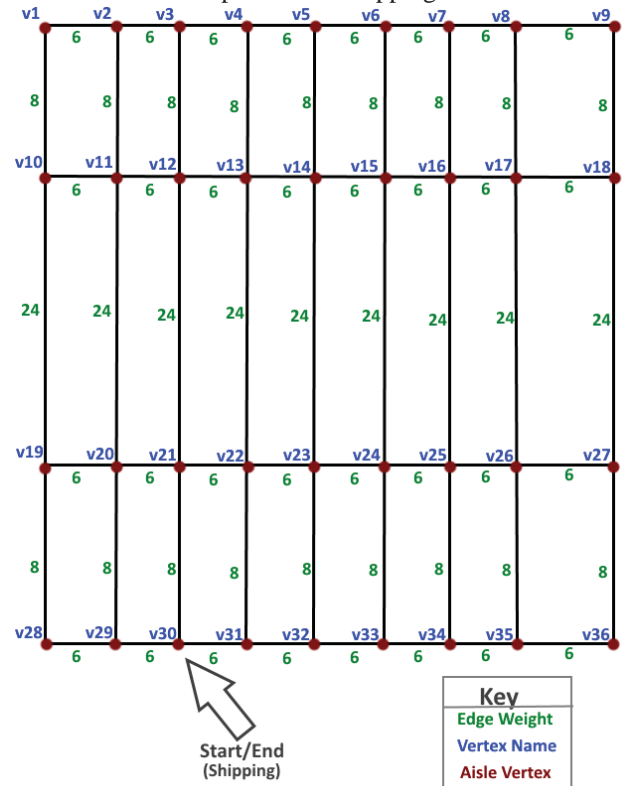


**Figure 5.1: Warehouse Layout**

After we have our layout defined, we can begin the process of turning this into a graph that can have traditional TSP and shortest path algorithms applied. The first step in this transformation is to create vertices at the intersection of all aisles, we do this because each aisle is an edge and the edges meet at intersections. Then draw each aisle as an edge.

Now, it is time to assign weights, of a uniform unit, to each edge. These weights represent the distance from the middle of one aisle intersection to the center of the neighboring aisle intersection. After assigning weights, it is time to randomly and uniquely assign each vertex an ID. This is done so that the user is able to interact with the Windows Form application, because the vertex ID correlates to an application vertex name. With that said, the application presents nodes to the user in the naming convention of “v1” to “vMax#Vertices”, so for the sake of simplicity, we will name our layout graph vertices as such. After completing the process defined above, a graph

similar to *Figure 5.2* is developed. Note we must also select a vertex to represent the shipping area at this time.



**Figure 5.2 Warehouse Layout Graph**

It can be seen that the red dots represent our vertices; the blue represents the identities of our vertices. We see we have a total of 36 vertices in our layout. Next, create all of the edges and place the weight along each edge. It is recommended that a thorough comparison of *Figure 5.1* and *Figure 5.2* be done to clearly see the figures represent the same warehouse layout. The entire reason for drawing this graph layout is to prepare all metadata that is collected by the application; planning and creating this image representation helps to reduce data entry errors and greatly reduce the time taken to turn the layout into a graph theory applicable problem. After our graph and its metadata are complete, we can begin using the application.

At this point it is important to note that this graph is a representation of the warehouse layout that will allow us to traverse from any aisle intersection to any other aisle intersection in the warehouse. This is different from the pick path problem, as the pick path problem travels from storage location to storage location. We eliminated the location vertices from our graph, as the same algorithms and processes apply to a graph going from aisle intersection to aisle intersection as a graph traversing location to location. This is true, because locations are found in our storage units. Our storage units are represented by the edges in our graph (see *Figure 5.2*). This means if we want to work with locations, rather than aisle intersections, we turn one aisle edge into multiple location edges. For example, if (from *Figure 5.2*) the

storage unit between v18 and v27 contained 5 locations, we would add 5 vertices to the one edge of length 24. These 5 locations would then be joined by smaller-portioned edges whose sum would add up to the original length of 24. This shows that by performing our analysis on the graph of aisle intersections, we are able to create simplified tests with fewer vertices without loss of generality for locations. As an example of a complete location graph see *Figure 5.3*; it shows pink dots that represent each location. It is easy to see that this graph has the same properties as *Figure 5.2*.

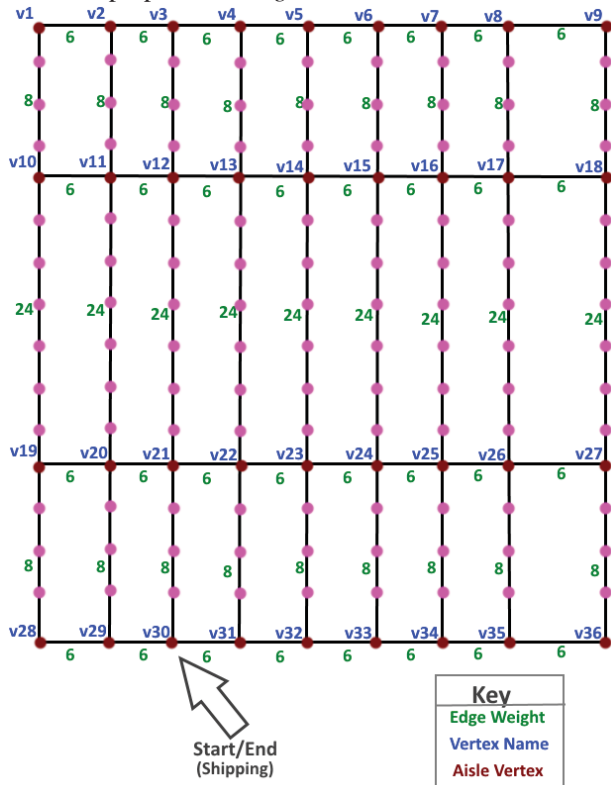


Figure 5.3 Layout Graph with Location Vertices

Now we must enter the graph from *Figure 5.2* into the C# application. After launching the application, we first enter the number of vertices and select “Create Graph”, see *Figure 5.4*.

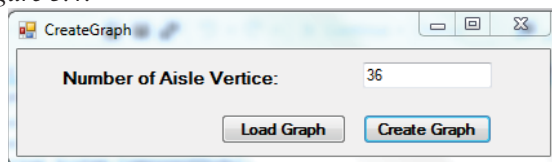


Figure 5.4 Create Graph with Number of Vertices

After selecting “Create Graph”, the application dynamically generates a form that is essentially an adjacency matrix. Each cell of the matrix has two input cells, we enter the distance between the vertices in the first input cell, ignoring the second. To fill this adjacency matrix form, we translate the metadata from our drawn layout graph in *Figure 5.2* to the adjacency matrix, entering the distance weight for each edge as in *Figure 5.5*.

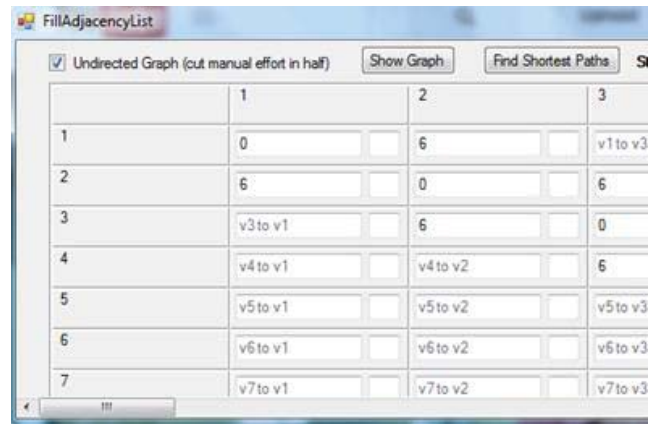


Figure 5.5 Fill Adjacency Matrix

After entering all of the metadata from the graph, we need to “Save Graph” so we can use this graph to solve the pick path problem from another form. After saving the graph, go back to the form from *Figure 5.4* and select “Load Graph”, select the graph that was just created and saved from the prior step. Now, enter the vertex that is the shipping area and the list of all vertices to visit as seen in *Figure 5.6*.

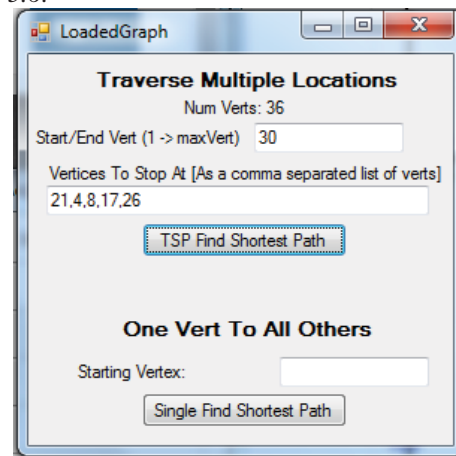
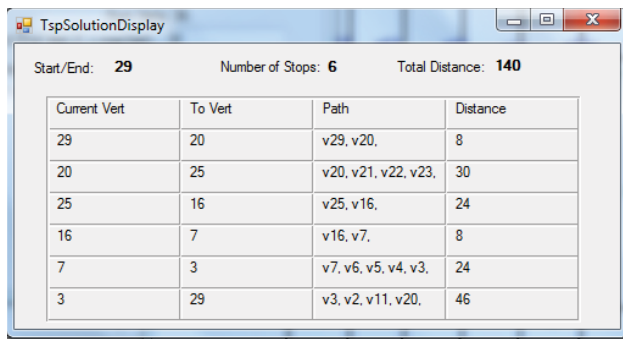


Figure 5.6 Find a Tour

After entering all information, select “TSP Find Shortest Path”. This generates an optimized pick path for the specified vertices using Dijkstra’s algorithm to find the closest neighboring vertex to each current vertex as we traverse the graph. The application uses the nearest-neighbor approach for solving the TSP end of the pick path problem, and Dijkstra’s algorithm to find the shortest paths.

After the optimal tour has been found using the above algorithms, it is displayed to the user. It is important to note that this form starts counting vertices at 0, rather than 1 so all vertex names are decremented by a constant of 1. As *Figure 5.7* shows, the form displays the vertices in the order in which they should be traversed, showing the exact path to get to each location as well as the path distance. All of these components make it easy to piece the tour together and present a usable path to the end users.



| Current Vert | To Vert | Path                | Distance |
|--------------|---------|---------------------|----------|
| 29           | 20      | v29, v20.           | 8        |
| 20           | 25      | v20, v21, v22, v23. | 30       |
| 25           | 16      | v25, v16.           | 24       |
| 16           | 7       | v16, v7.            | 8        |
| 7            | 3       | v7, v6, v5, v4, v3. | 24       |
| 3            | 29      | v3, v2, v11, v20.   | 46       |

Figure 5.7 Pick Path Tour

This application shows the feasibility of using the algorithms described in Section 4 to generate optimized pick paths using simple vertex and edge data structures. This application does not; however, implement a recommended TSP algorithm, as the nearest-neighbor approach does not guarantee any degree of accuracy in regard to the optimum tour [3]. With a TSP algorithm guaranteeing an upper bound, this application would be far more reliable and recommended for use.

In short, we are able to demonstrate the practicality of using advanced TSP and shortest path algorithms for solving the optimized pick path, although further implementation is required to guarantee any degree of accuracy.

## 6. Future Work

It is our goal to implement an optimized pick path finder that implements a TSP solution with a guaranteed upper bound to the optimum tour. This will involve reworking the vertex and edge data structures, as well as storing them in a more optimal data structure than a list; preferably using a data structure that closely lends itself to finding a minimal spanning tree based on the fact that many upper bound TSP solutions first find a MST before solving the TSP. This will make it more efficient to implement algorithms such as Christofides' or twice-around-the-tree.

## 7. Conclusion

In conclusion, pick path optimization is a component of warehouse operations with much room for improvement in efficiency. To optimize the creation of pick paths, further research must be done in heuristics for solving the travelling salesperson problem with tight upper bound guarantees. The algorithms analyzed in this paper are capable of generating pick paths, although to guarantee any degree of accuracy to the optimum pick path tour, more advanced and complicated algorithms must be implemented such as Christofides' or twice-around-the-tree. With the implementation of these algorithms, optimal pick paths can be reliably generated and used for directing order pickers.

## References

- [1] Theys, C., Braysy, O., Dullaert, W., Raa, B., 2010. Using a TSP heuristic for routing order pickers in warehouses. *European Journal of Operational Research* 200 (3), 755–763.
- [2] Bartholdi, J.J., Hackman, S.T., 2006. *Warehouse and Distribution Science*. Release 0.96.
- [3] A. Levitin, *The design and analysis of algorithms 3<sup>rd</sup> Edition* (Upper Saddle River, NY: Addison-Wesley, 2012).
- [4] N. Deo, *Graph theory with applications to engineering and computer science* (Englewood Cliffs, NJ: Prentice-Hall, 1974).
- [5] T. Cormen, C. Leiserson, R. Rivest, & C. Stein, *Introduction to algorithms* (Cambridge, MA: MIT Press, 1990).
- [6] F. Zhan, "Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures," *Journal of Geographic Information and Decision Analysis, Vol.1, No.1*, pp. 69-82, 1998.
- [7] Q. Song, X. Wang. Partitioning Graphs to Speed Up Point-to-Point Shortest Path Computations. *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. IEEE, 2011.
- [8] R. Möhring, H. Schilling, B. Schütz, D. Wagner, and T. Willhalm. Partitioning graphs to speed up Dijkstra's algorithm. *4th International Workshop on Efficient and Experimental Algorithms (WEA)*, pages 189–202, 2005.
- [9] Pohl, I. Bi-directional search. In B. Meltzer and D. Michie (Eds.). *Machine Intelligence 6* (American Elsevier, New York, 1971) 127-140.
- [10] Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1 (1), 53-66.
- [11] CHRISTOFIDES, N. 1976. Worst-case analysis of a new heuristic for the traveling salesman problem. *Symposium on New Directions and Recent Results in Algorithms and Complexity*, J. F. Traub, ed. Academic Press, Orlando, Fla., p. 441.
- [12] F. Chen, H. Wang, C. Qi, and Y. Xie, "An ant colony optimization routing algorithm for two order pickers with congestion consideration," *Computers & Industrial Engineering*, vol. 66, no. 1, pp. 77–85, 2013.

# Proportional Share Scheduling employing Performance-aware Virtual Time in Multiprocessor Systems

Munseok Kim<sup>1</sup>, Hyunmin Yoon<sup>2</sup>, and Minsoo Ryu<sup>1</sup>

<sup>1</sup>Department of Computer Science and Engineering, Hanyang University, Seoul, Korea

<sup>2</sup>Department of Electronics Computer Engineering, Hanyang University, Seoul, Korea  
{mskim, hmyoon}@rtcc.hanyang.ac.kr, msryu@hanyang.ac.kr

**Abstract** – In proportional share scheduling, the different performances of CPUs can make running tasks unfair during a period. To make them fair in a system, we present a proportional share scheduling employing performance-aware virtual time (PVT) maintained globally. This PVT is the share of CPU time received by a task and increases at a rate proportional to the performance of CPU where the task is running on and inversely proportional to the weight of the task. The schedulers of CPUs, when they assign CPU time to a task, utilize PVT to make a decision which task and how long it should preempt CPU to minimize the difference of PVTs among tasks. We evaluated our approach experimentally on general purpose operating system in the homogeneous and heterogeneous multiprocessor (HMP) systems. On both systems, the results show the significant improvement that is near-perfect (around 99% better) fairness in the homogeneous multiprocessor system and much better (more than 60% better) in the HMP system.

**Keywords:** Proportional share scheduler, virtual time, performance-aware, heterogeneous multiprocessor, fairness.

## 1 Introduction

Proportional share scheduling which provides abstractions for multiplexing resources among tasks allocates resources to a task proportional to its weight to guarantee the weighted fairness in a system. Unfortunately, generally this fairness cannot be completely achieved in practice because infinitesimal CPU quanta are required in theory. To minimize the difference of CPU time between a task ideally needed in theory and actually received, previous works have introduced various approaches such as [3] and [12]. These approaches, however, do not consider that the unfairness among tasks can arise also by the different performance of CPUs. In practice, the unfairness arises in the homogeneous multiprocessor system which has CPUs of different frequencies like x86-based, and it is more obvious in case of heterogeneous multiprocessor (HMP) systems.

The HMP system such as ARM big.LITTLE processor was introduced to make an energy-aware scheduling possible by processing tasks on the core of less energy consumed. In such a system, since each CPU has different capacity and

frequency, unless schedulers consider the performance of CPUs, the unfairness could be amplified and more frequent than the homogeneous. Nevertheless, in HMP related works, most focus is on the performance optimization, energy-saving and showing the benefit of them [6]-[8], while not much effort is being given to guarantee the fairness among tasks.

The fairness among tasks is significantly important factor to guarantee quality-of-service (QoS) of multi-program workloads [9], [10]. For example, applications such as immersive virtual environments and interactive multi-media can lead to unpredictable and undesirable result because they require real-time computation and communication services from the operating system on the assumption that all tasks make equal progress on CPUs. Yet, this expectation cannot be guaranteed in the case of that a task running on a big core (or the higher frequency of CPU) works more than the other on a small core (or the lower frequency of CPU). In this case, finally, the difference of work done among tasks should be minimized with the consideration about the performance of CPUs to guarantee QoS based on the fairness.

To achieve this goal with considering about the frequency and capacity of CPUs as major factors of the performance, we present a proportional share scheduling employing performance-aware virtual time (PVT). PVT, a virtual time of a task maintained in a system widely, increases at a rate proportional to the performance of CPU where the task is running on and inversely proportional to the weight of the task based on CPU time received by the task. This PVT makes the unfairness among tasks traceable relatively, and schedulers utilize it when they assign CPU time to a task. By leveraging existing proportional share scheduler employing PVT, this work provides near-perfect fairness (more than 99%) in the homogeneous multiprocessor system and much better fairness (more than 60%) in the HMP system than the previous one.

The remainder of this paper is composed of several sections as follows. Section 2 describes an existing proportional share scheduling and its limitation, and Section 3 introduces PVT and how to utilize it in a system. In section 4, we show substantially improved results based on the completely fair scheduler (CFS) employing PVT in Linux as a representative fair scheduler in practice. Section 5 concludes this paper with the consideration of future works.

## 2 Proportional Share Scheduler and Limitation

Proportional share scheduling which is able to provide abstractions for multiplexing resources among tasks is allocating resources to a task proportional to its weight to guarantee weighted fairness in a system [1]. Proportional share resource allocation is ideally generalized processor sharing (GPS) scheme [2]. A fluid-style resource and perfect fairness based on an infinitesimal fluid resource model are assumed, but actual system cannot provide resource infinitesimally in practice. Therefore, approximate scheduling scheme is being proposed like packet by packet GPS (PGPS) [4], and weighted fair queuing (WFQ) [2]. Figure 1 shows the ideal scheme and quantum-based scheduling which is able to be implemented to achieve proportional share scheduling in practice [11].

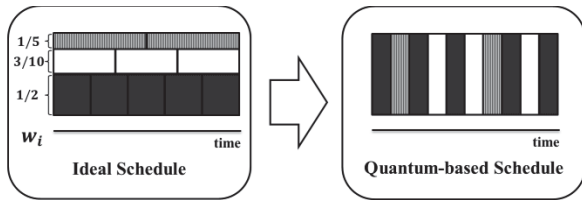


Figure 1. Proportional share scheduling.

Let  $W_i$  be the weight of task  $\tau_i$  and  $\Phi$  be the set of all active tasks at time  $t$ . The share  $S_i(t)$  of a task  $\tau_i$  at time  $t$  is defined as follows.

$$S_i(t) = \frac{W_i}{\sum_{j \in \Phi} W_j}.$$

The share  $S_i(t)$  is changeable in runtime because the number of tasks in a system can be changed dynamically. For example, if a new task is initialized, the total weight  $\sum_{j \in \Phi} W_j$  is increased and the share  $S_i(t)$  of task  $\tau_i$  is decreased on the contrary. Therefore, a proportional share scheduler only guarantees a relative share of CPU time according to the total weight changes.

To measure the difference of CPU time received by a task between the ideally needed and the actually received, virtual-time domain [1] can be utilized. In this domain, the virtual time is the share of CPU time received by a task. The share of CPU time is allocated to a task proportional to the weight of the task. Therefore, the virtual time can be computed as follows.

$$VT(\tau_i, t) = A(\tau_i, t) \times \int_0^t \frac{\sum_{j \in \Phi(t')} W_j}{W_i} dt'.$$

Let  $A(\tau_i, t)$  be the CPU time assigned to task  $\tau_i$  by time  $t$ , and let  $\Phi(t')$  be the set of all tasks active at time  $t'$ . Because the virtual time increases at a rate proportional to the sum of

weights of all tasks, if the total sum of weights increases, the virtual time of  $\tau_i$  increases faster and vice versa.

Additionally, the *lag* is defined as the ideal CPU time which should be assigned to a task by subtracting the actual CPU time received by a task. Suppose that task  $\tau$  is active and have a fixed weight in the interval  $[t, t']$ . Let  $S_{\tau, A}(t, t')$  denotes the CPU time received by the task  $\tau$  in  $[t, t']$  under a certain scheduling scheme A, and  $S_{\tau, GPS}(t, t')$  denotes the CPU time under the Generalized Processor Sharing (GPS) scheme; an idealized scheduling model which achieves perfect fairness. The *lag* of task  $\tau$  at time  $t$  ( $t \in [t, t']$ ), for any interval  $[t, t']$ , is formally defined as

$$lag_{\tau}(t) = S_{\tau, GPS}(t, t') - S_{\tau, A}(t, t').$$

However, in the case of proportional share schedulers based on partitioned scheduling, they have each run queue individually and try to guarantee the fairness with the consideration about the weights of tasks only within the run queue where the scheduler involved in. Although this approach has no problem in a system which has the same performance of CPUs, it can incur a problem in a system which has different performance of CPUs. In such a system, as well as the sum of weights of tasks on each CPU, the performance of each CPU can be different by the dynamic frequency changes or the static capacity of processors.

In case of the different frequency of CPUs, for example, consider four tasks  $\tau_1, \tau_2, \tau_3$  and  $\tau_4$  which have the same weight value 100 individually on the dual-core processor which has CPUs 1 and 2. The frequency of CPU 1 is 1 GHz while it of CPU 2 is 2 GHz. They can operate as following scenarios.

1. Four tasks  $\tau_1, \tau_2, \tau_3$  and  $\tau_4$  start simultaneously at time 0
2. Execute  $\tau_1$  and  $\tau_2$  in CPU 1,  $\tau_3$  and  $\tau_4$  in CPU 2 during 2 sec
  - a.  $\tau_2$  moved to CPU 2 without delay
  - b.  $\tau_3$  moved to CPU 1 without delay
3. Execute  $\tau_1$  and  $\tau_3$  in CPU 1,  $\tau_2$  and  $\tau_4$  in CPU 2 during 2 sec
  - a.  $\tau_3$  moved to CPU 2 without delay
  - b.  $\tau_2$  moved to CPU 1 without delay
4. Execute  $\tau_1$  and  $\tau_2$  in CPU 1,  $\tau_3$  and  $\tau_4$  in CPU 2 during 2 sec

In this case, all of tasks can receive the same amount of CPU time; however, at the point of the amount of work done, it can be totally different result. If we consider the performance of CPUs, according to the proportion of performance of each CPU, we can describe the relative ratio of the amount of work done by each task based on CPU time like follows. The  $\tau_1$  processed  $(2 + 2 + 2)$  of work in CPU 1,  $\tau_2$  processed  $(2 + 2 \times 2 + 2)$  and  $\tau_3$  processed  $(2 \times 2 + 2 + 2 \times 2)$  in both CPUs while the  $\tau_4$  processed  $(2 \times 2 + 2 \times 2 + 2 \times 2)$  in CPU 2. Finally, the amount of work done by each task is in the order of  $\tau_1 < \tau_2 < \tau_3 < \tau_4$ . If the length of execution

time at the scenario 2, 3, and 4 could be manipulated properly,  $\tau_2$  and  $\tau_3$  can be fair while  $\tau_1$  is of the least work done and  $\tau_4$  is of the most. Even though the total sum of weights in each CPU during execution is absolutely balanced, the result of work done among tasks can be totally different by the performance of CPUs and it eventually leads tasks to the unfairness. Finally to guarantee the fairness among tasks in a system, as well as the sum of weight of tasks, the performance of CPUs should be considered together.

### 3 Performance-aware Virtual Time

In this section, we propose performance-aware virtual time (PVT) where the performance of CPU is defined using two major factors: frequency and capacity. PVT, as a virtual time of a task maintained globally, increases at a rate proportional to the performance of CPU where the task is running on and inversely proportional to the weight of the task based on CPU time received by the task. Additionally, we utilize PVT to monitor the fairness measure in a system. Basically our approach considers both systems homogeneous and heterogeneous multiprocessor (HMP), and each CPU individually has a dynamic voltage and frequency scaling (DVFS) which is efficient technology for dynamic power management (DPM). Therefore, PVT of task  $\tau_i$  of weight  $W_i$  in CPU  $P$  is calculated as follows.

$$PVT_P(\tau_i, t) = \frac{A(\tau_i, t) \times W_{max}}{W_i} \times \frac{C_P \int_0^t F_P(x) dx}{F_{max} \times t}.$$

Let  $W_{max}$  and  $F_{max}$  be the maximum weight of a task and CPU frequency acceptable in a system, and  $\int_0^t F_K(x) dx$  be the total amount of frequencies by time  $t$ , and  $C_P$  ( $0 \leq C_P \leq 1$ ) is a constant value of CPU  $P$  which indicates the relative ratio of the performance among CPUs from the fastest ( $C_P = 1$ ). This constant value can be determined depending upon the types of tasks running on a system based on the results of various benchmarks or some specific metrics reported by chip vendors also.

Additionally, if we scale  $W_{max}$  and  $F_{max}$  as a same value to reduce a fraction, we finally can simplify equation as follows.

$$PVT_P(\tau_j, t) = \frac{A(\tau_j, t)}{W_j} \times C_P \int_0^t F_P(x) dx.$$

All tasks in a system have their own PVT values. By utilizing these values, we are able to monitor the change of fairness in a system by the changes of results of periodic repeating follows.

$$MaxDiff(t) = \max_{\tau \in \Phi} PVT_P(\tau, t) - \min_{\tau \in \Phi} PVT_P(\tau, t).$$

The maximum result of  $MaxDiff(t)$  should be maintained as a similar level. This means eventually that the results are

always bounded into the specific value, and the level of fairness measure is being maintained in a system.

The difference (*lag*) of CPU time received by the task  $\tau_i$  between the ideally needed and the actually received by time  $t$  based on the performance of CPU  $P$  can be derived as follows.

$$lag_{\tau_i}(t) = \left( \frac{\sum_{j \in n} \left( \int_0^t F_j(x) dx \times C_P \right)}{\sum_{i \in \Phi} W_i \times t} \times W_i - \frac{\int_0^t F_P(x) dx}{W_i} \right) \times A(\tau_i, t).$$

Let  $n$  be the number of CPUs in a system. When  $lag_{\tau_i}(t)$  is greater than zero,  $\tau_i$  received less time than the time ideally needed, and in case of zero the time was ideally received and the more time received in the other case; however, operating system cannot reclaim CPU time from a task that has already received. Therefore, we utilize PVT to find a task with the lower PVT and give more CPU time prior to the tasks with higher PVT in a system to minimize the difference of PVTs.

In case of a I/O intensive task or a task of starting, its PVT is revised exceptionally because PVT of a task out of run queue can be maintained as the lowest without system progression applied when they are coming back to run queue again. To make a decision of revision needed or not, we classify two groups of tasks according to the state transition of task. In case of task transition between ready and running, tasks should be revised, and the other cases except for the first are not revised. Therefore, PVTs of I/O tasks of the latter case need to be revised by subtracting as much time as system progressed during being out of run queue. Finally PVTs of the tasks is revised to keep the previous position in a system widely as

$$PVT_M(\tau_i, t') = \left( \min_{\tau \in \Phi} PVT_P(\tau, t') - \min_{\tau \in \Phi} PVT_P(\tau, t) \right) + PVT_P(\tau_i, t).$$

The I/O intensive task  $\tau_i$  is out from the run queue of CPU  $P$  at time  $t$  and inserted into the run queue of CPU  $M$  at time  $t'$ . In this case, the minimum PVT of each time  $t$  and  $t'$  is subtracted and added to keep the previous level of fairness position in a system.

### 4 Experimental Evaluation

Completely fair scheduler (CFS) is the most popular and the first fair scheduler applied to the general purpose operating system while the other operating systems like Windows and Linux of earlier version (before the version of 2.6.23) are providing round-robin scheduling. By these reasons, to achieve our goal, we utilized existing CFS in Linux (after the version of 2.6.23) to employ performance-aware virtual time (PVT). PVT of each task was also utilized to monitor the fairness measure in a system. To evaluate the effectiveness of our proposed approach, we considered two different environments both homogeneous multiprocessor system and heterogeneous multiprocessor (HMP) system. The characteristics of these are shown in Tables 1 and 2.



Table 1. homogeneous multiprocessor environment

|          |                  |   |
|----------|------------------|---|
| Hardware | CPU              | Intel i7-4770 3.4 GHz Dual processor                    |
|          | RAM              | 4 GB DDR3 SDRAM   |
| Software | Operating System | Ubuntu 14.04 (on VMWare), Linux Kernel Version : 3.18.2 |

Table 2. heterogeneous multiprocessor environment

|          |                  |  |
|----------|------------------|--|
| Hardware | CPU              | Samsung Exynos5 Octa big.LITTLE processor            |
|          | RAM              | 2 GB LPDDR3 RAM                                      |
| Software | Operating System | Android 4.4.4 Kit Kat, Linux Kernel Version : 3.10.9 |

To evaluate the fairness among tasks between operating systems the original version and the version of PVT employed in the system Table 1, we created simple application whose performance is mostly proportional to the frequency of CPU to make  $C_p$  simply be 1 in all CPUs. The execution time of each instance was compared to measure the fairness among tasks. This application just repeats infinite loop simply and print out the cumulative average time consumed at every 2.5 billionth loop. This cumulative average execution time was approximately 1.3 sec in practice. Three instances of the application were executed concurrently in the system shown in Table 1, and we got the results 10 times per 1 min from the original version and our version of PVT applied in Linux kernel 3.18.2.

Even though there is not a significant difference in the execution of three instances evaluated on the homogeneous dual-core processor, as depicted in Figure 2, the different execution time of tasks arises and being kept consistently. Interestingly, this experiment was evaluated in the one of most popular general purpose operating system and processors even though the type of application was not very common.

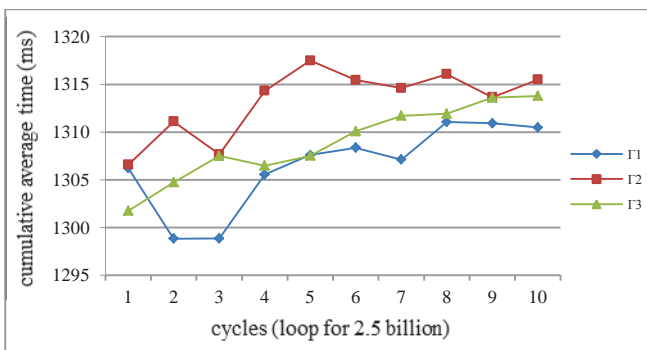


Figure 2. cumulative average time of cycles on Ubuntu 14.04 LTS of kernel 3.18.2

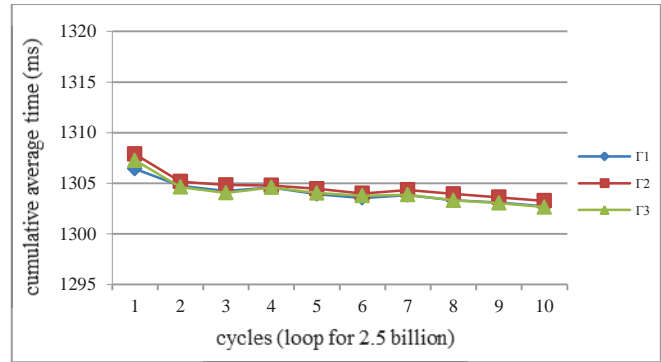


Figure 3. cumulative average time of cycles after PVT applied

CFS employing PVT in the version of inux kernel 3.18.2 achieves near-perfect fairness among three tasks on the homogeneous dual-core processor as depicted in Figure 3 while the unfairness arises obviously in Figure 2. When the time  $t$  is 100, Figure 4 shows that  $MaxDiff(t)$  of three tasks is bounded in 3 sec (of PVT) approximately.

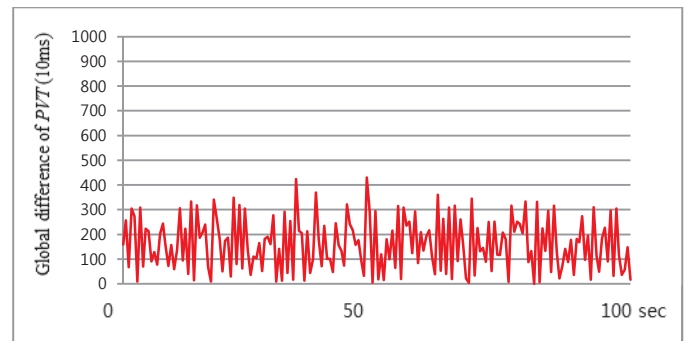


Figure 4. Periodic repeats of  $MaxDiff(t)$  of 3 tasks in the homogeneous during 100 sec

In case of the system in Table 2, we created a similar application, but the number of loop operation was changed to 1 million, and a sleep code for 5ms was inserted after printing out (per about 850ms in practice) to simulate the effects of I/O operation together. Twelve instances of this application were created and executed concurrently, and we gathered the results 10 times per 1 min.

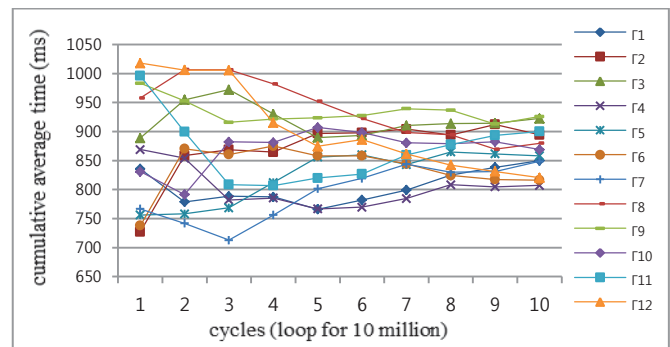


Figure 5. Cumulative average time of cycles on Android KitKat of kernel 3.10.9

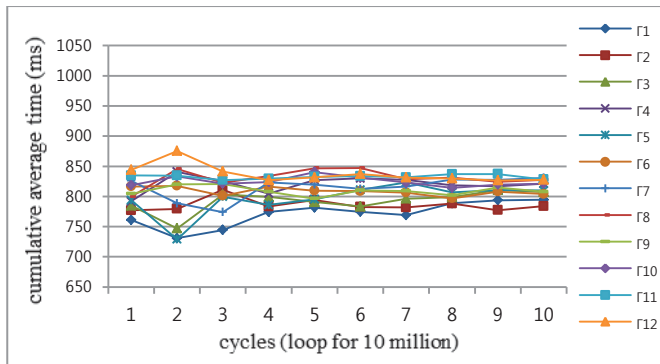


Figure 6. Cumulative average time of cycles after PVT applied

Figure 5 shows the results when we measure the cumulative average time of each task using the existing kernel. There is a lot of variation among the execution times of tasks even though all tasks are identical. In Figure 6, when compared to Figure 5, the cumulative average time values of the twelve tasks are more narrowed down. Based on the results, the maximum deviation of the cumulative average time can be shown as Figure 7.

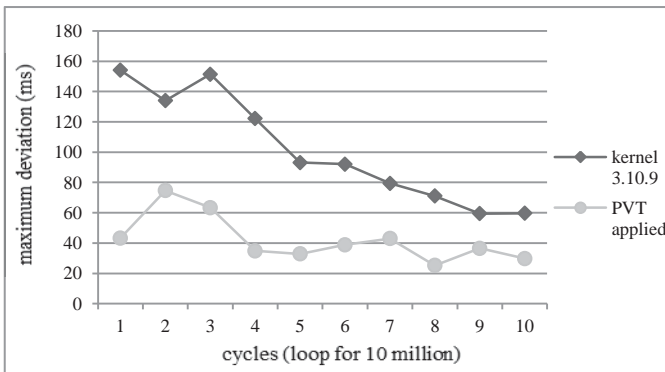


Figure 7. maximum deviation of cumulative average time

Figure 7 shows that PVT applied kernel guarantees the fairness among tasks by giving at least 60% better than the original kernel in HMP system. During 600 sec,  $MaxDiff(t)$  of the twelve tasks are bounded into 5 sec as shown in Figure 8.

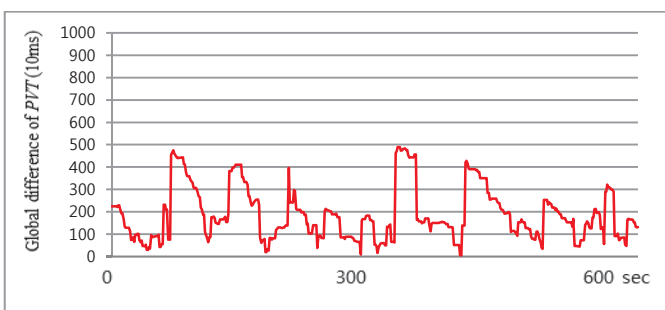


Figure 8. Periodic repeats of  $MaxDiff(t)$  of 12 tasks in the HMP system during 600 sec

Additionally, according to the each result of  $MaxDiff(t)$ , the fairness among three tasks in the homogeneous is being guaranteed better than it of twelve tasks in the heterogeneous as described above results.

## 5 Conclusion

In this paper, we proposed a proportional share scheduling employing performance-aware virtual time (PVT) to guarantee and measure the fairness among tasks in a system. The proposed approach leads the fairness to the significantly improved than previous systems. We also introduced how to monitor the fairness by utilizing PVT to compare the level of fairness measure in a system.

Additionally, if we consider the remaining 40% in the results of HMP system, there could be more factors such as the number of heterogeneous CPUs and migration, the type of tasks, miss-rate of cache, the policy of load balancer and so forth which may affect the fairness of tasks in a system. As a future goal, we plan to focus on the relationship between these factors and the fairness guarantees. As a future work, we also intend to determine the capacity of CPU which can affect the fairness measure.

## 6 Acknowledgment

This work was supported partly by Seoul Creative Human Development Program (HM120006), partly by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (NRF-2011-0015997), and partly the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the C-ITRC (Convergence Information Technology Research Center) (IITP-2015-H8601-15-1005) supervised by the IITP (Institute for Information & communications Technology Promotion).

## 7 References

- [1] NIEH, Jason; VAILL, Christopher; ZHONG, Hua. Virtual-Time Round-Robin: An  $O(1)$  Proportional Share Scheduler. In: *USENIX Annual Technical Conference, General Track*. 2001. p. 245-259.
- [2] MARKATOS, Evangelos P.; LEBLANC, Thomas J. Using processor affinity in loop scheduling on shared-memory multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, 1994, 5.4: 379-400.
- [3] CHANDRA, Abhishek, et al. Surplus fair scheduling: A proportional-share CPU scheduling algorithm for symmetric multiprocessors. In: *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*. USENIX Association, 2000. p. 4-4.
- [4] PAREKH, Abhay Kumar; GALLAGER, Robert G. A generalized processor sharing approach to flow control in integrated services networks-the single node case. In:

*INFOCOM'92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE*. IEEE, 1992. p. 915-924.

[5] KUMAR, Rakesh, et al. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In: *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*. IEEE, 2003. p. 81-92.

[6] KOUFATY, David; REDDY, Dheeraj; HAHN, Scott. Bias scheduling in heterogeneous multi-core architectures. In: *Proceedings of the 5th European conference on Computer systems*. ACM, 2010. p. 125-138.

[7] SHELEPOV, Daniel, et al. HASS: a scheduler for heterogeneous multicore systems. *ACM SIGOPS Operating Systems Review*, 2009, 43.2: 66-75.

[8] GREENHALGH, Peter. Big. little processing with arm cortex-a15 & cortex-a7. *ARM White paper*, 2011.

[9] OGRAS, Umit Y.; MARCULESCU, Radu. " It's a small world after all": NoC performance optimization via long-range link insertion. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2006, 14.7: 693-706.

[10] BONALD, Thomas; MASSOULIÉ, Laurent. Impact of fairness on Internet performance. In: *ACM SIGMETRICS Performance Evaluation Review*. ACM, 2001. p. 82-91.

[11] KIM, Hyungwoo, et al. Fixed Share Scheduling via Dynamic Weight Adjustment in Proportional Share Scheduling Systems.

[12] D. Ok, B. Song, H. Yoon, P. Wu, J. Lee, J. Park, and M. Ryu, "Lag-Based Load Balancing for Linux-based Multicore Systems," *The 2013 International Conference on Foundations of Computer Science*, July 2013.



## **SESSION**

# **FOUNDATION OF COMPUTER SCIENCE, COMPUTATIONAL SCIENCE AND NOVEL CONCEPTS + EDUCATION**

**Chair(s)**

**TBA**



# Peircean Semiotics: Perspectives in Computer Science

Ricardo Maciel Gazoni<sup>1</sup>

<sup>1</sup>TIDD – Technologies of Intelligence and Digital Design,  
PUC – Pontificia Universidade Católica, São Paulo, Brazil

**Abstract**—*The aim of this study is the analysis of some concepts of Computer Science in the light of Peircean semiotics taken as a cognitive theory. Based on the main principles of Peirce's Phenomenology, it is possible to point out new ways and approaches to a wide range of subjects, that may go from Artificial Intelligence to Programming.*

**Keywords:** Charles S. Peirce, Computer Science, Semiotic

## 1. Introduction

In 2011 Mihai Nadin [1] stated what seems to be common sense among semioticians: semiotics is relevant to Computer Science. He then further asks: why do computer scientists, with few exceptions, continue to ignore semiotics? His rich analysis presents some causes, among them a superficial knowledge of semiotics. In the following pages I will outline an account of what could result from the use of a Peircean approach to Computer Science. The paper starts with a brief introduction of Peirce's semiotics and concepts then proceed to an explanation of some elements that are part of Computer Science in a Peircean perspective<sup>1</sup>.

## 2. Peircean Semiotics – some essentials

According to Lucia Santaella [3], Peircean semiotics is a sign-based cognitive theory. Peirce says that “all thinking is conducted in signs” (CP 6.338, 1909), therefore it is natural that the science that studies signs also studies thoughts. It is impossible to give a complete exposition of Peircean semiotics —or cognitive theory, as it might be called— in this article, so I will try to expose the concepts that are most interesting for Computer Science in a way that, albeit incomplete, I believe is more appropriate for computer scientists.

### 2.1 Cartesian $\subset$ Peircean

One important thing to keep in mind when we are talking about the Peircean cognitive theory is that it differs from the “usual” Cartesian dualist theory of cognition. Peirce strongly disagrees with some Cartesian principles and conclusions. But although he dismisses the Cartesian philosophy, he does not dismiss logic thinking, but gives it an increased range: Peircean semiotics gives us the tools that are supposed to

explain thinking (among other things). Peirce's theory of cognition should explain not only Cartesian dualist thinking, but also its tools. There is no need to abandon the clarity and precision that characterizes Cartesian thought: it is enough to disregard a conclusion as true only because it is clear, precise and derives from indubitable premises —it should also, for instance, be verifiable, otherwise one may accept beliefs that one is incapable of doubting but which are, nevertheless, false.

One of the most interesting consequences of the rejection of Cartesian dualism for computer scientists is that this position eliminates the separation between mind and matter, as Descartes postulates it in his *Discourse on the Method* [4]. This is important because it allows us to place the phenomena of semiosis —and thinking is a form of semiosis— anywhere inside or outside our minds. Thus it is possible to talk about such things as machine semiosis, disembodied (or embodied) thoughts, and compare the various forms of semiosis on a common ground.

The rejection of the dualism between mind and matter reflects, according to Peirce, the fact that we, and our minds, are consequences of natural evolution, and therefore cannot present phenomena that do not already exist in nature. What happens in our minds must also happen elsewhere. But Peirce is also strongly opposed to any form of dualism; he postulates the theory of Synechism, according to which all that exists is continuous (cf., e.g. CP 1.172, 1897) — which includes the continuity between mind and matter. His thoughts are founded on a coherent philosophy that begins with his Phenomenology; we will start with the Peircean categories, but let us first remember that, opposed to the Cartesian method —where complex thoughts may be decomposed in simpler ones in order to be apprehended—, these complex concepts lose much of their meaning if we try to simplify them by decomposing them.

### 2.2 The Peircean categories

Here are Peirce's views about the role of the categories in his own words: “I essay an analysis of what appears in the world. It is not metaphysics that we are dealing with: only logic. Therefore, we do not ask what really is, but only what appears to everyone of us in every minute of our lives. I analyze experience, which is the cognitive resultant of our past lives, and find in it three elements. I call them Categories” (CP 2.84, 1902). And they are only three, constituting all that there is in consciousness (CP 1.382,

<sup>1</sup>Citations to the Collected Papers [2] are made within parenthesis in the form “CP V.P, YYYY”, where V stands for the volume, P for the paragraph number and YYYY for the year the text was probably written.

1890). Peirce called them *firstness*, *secondness* and *thirdness*. Following one of Peirce's presentation of his categories (CP 2.79-118, 1902: *Partial Synopsis of a Proposed Work in Logic*), let us begin with the one that seems easiest from the Cartesian background.

### 2.2.1 Secondness

In order to characterize secondness, Peirce used words such as *binarity*, *brute force* (CP 2.84, 1902), *struggle* (CP 1.322, 1903, CP 5.45, 1903), *obistence*, "suggesting *obviate*, *object*, *obstinate*, *obstacle*, *insistence*, *resistance*" (CP 2.89, 1902). We can easily identify secondness on whatever resists to us in a way that does not depend on argumentation. Just like most of the immaterial world around us: it persists, no matter what. This is typical secondness. But it is not restricted to the so-called "existent" objects. If you imagine a fictitious creature, say a dragon; imagine its color. No matter what happens, the color you just imagined will not change —although you could have imagined any other color, and in spite of the fact that you can "change" the color of the dragon just imagined, once the first dragon's color has been thought its memory will remain and cannot be changed by your effort: this is also secondness. Another example is the duality husband-wife: it is the husband that makes the wife a wife, and the wife that makes the husband a husband. So there is a form of reaction beyond the binary form husband-wife. This is also secondness.

If we think about the computer as a real world device that follows the laws of Physics and behaves according to the set of electrical signals inside it that we call "bits", then we have a behavior that is prominently determined by secondness. It is exactly the same when we think about an imaginary device such as a Turing machine: its behavior, albeit imaginary, is primarily determined by phenomena of secondness. And the imaginary machine has a much more predictable behavior than the real one since it is not composed of parts that may malfunction.

Secondness always involves two elements, two *relata*, which unavoidably impose themselves on each other. When we "feel" secondness, it is because we are one of these *relata*. Almost everything that is "evident", "precise", "clear" in a Cartesian sense —even deductive ratiocination—, constitutes or at least embeds a phenomenon that presents secondness. A good remark that could be made now is: since there are two more categories, it means that whatever is not "evident" in a Cartesian sense can be so in two different ways. It is possible to start thinking about it knowing that the names of the categories are related to the number of related elements in them.

### 2.2.2 Firstness

The question is: what is it that does not relate to any other thing? Can it in any form be cognized? It cannot be properly thought, although we could infer how it is like.

Peirce take as an example "what could appear as being in the present instant were it utterly cut off from past and future" and further speculates: "there might be a sort of consciousness, or feeling, with no self; and this feeling might have its tone" (CP 2.85, 1902). There could be no action, and no binarity, neither continuity nor synthesis. He also refuses to call it unity, since unity presupposes plurality. It is what he calls "Firstness, Orience or Originality" (CP 2.85, 1902): whatever is without a reason or compulsion. It should not be confounded with perception —see [3]. According to Santaella, perception is a complex phenomenon that involves the three Peircean categories. Roughly speaking, when we perceive there is a sort of novelty present in it —and this is the firstness component— but as soon as whatever is perceived does not evanesce a strong secondness component arises; thirdness will appear only in perceptual judgment. In a general way, secondness contains a component of firstness, but the opposite is not true. Firstness should not also be confounded with the concept of quality. The mere characterization of a quality presupposes that it presents itself as secondness, although this also has a strong firstness component.

It is natural to think that firstness cannot appear in the realm of computer. But modern equipments have at their disposal external sources of entropy, such as cameras and microphones, or more sophisticated devices that generate streams of truly unpredictable bits. These devices are sources of manageable firstness that can be used with very interesting consequences.

An interesting realm to study firstness is the one of mental phenomena. The most interesting cases in which it appears are (1) when we are considering possibilities —which are a form of firstness—, (2) when we are observing mental diagrams —here firstness appears in the observation itself, for it allows us to find things in the diagram that were not used in its construction— and (3) when we create new hypotheses in what Peirce called *abductive reasoning*.

### 2.2.3 Thirdness

Peirce himself pointed out the apparent adequacy of these two categories to describe the facts of experience (CP 1.359, 1890), but they are not, in fact, sufficient. The concept of thirdness is essential, according to Peirce, to complement the other two and explain all that appears to the mind. It relates three elements, but this relation is not —although it can be in some cases— determined by brute force, or by a blind causal relation (secondness), or by chance (firstness). In a brilliant explanation (CP 2.86, 1902), Peirce uses the fact that the future affects us in mental forms, intentions and expectations, but it does not do so directly, for it does not exist yet: this is an example of thirdness.

A phenomenon of prominent thirdness in computers is related not to what happens after the bits that will be "run" are put in place, but before it: to define what people



want the computer to do, or the intention behind the use of the computer, is such a case. Some important ideas — according to Peirce (CP 1.340, 1895)— are of prominent thirdness: generality, infinity, continuity, diffusion, growth, intelligence. But the easiest is the idea of *sign*.

### 2.3 Sign

As just stated above, a sign is an idea of predominant thirdness. It relates three things: a *representamen*, which is sometimes also called sign—in a narrower sense—, an *object* and an *interpretant*. In Peirce's words: "A *Sign*, or *Representamen*, is a First which stands in such a genuine triadic relation to a Second, called its *Object*, as to be capable of determining a Third, called its *Interpretant*, to assume the same triadic relation to its Object in which it stands itself to the same Object. The triadic relation is *genuine*, that is its three members are bound together by it in a way that does not consist in any complexus of dyadic relations" (CP 2.274, 1903). And now we have to take the utmost care not to oversimplify the concept and cause confusion. For we are used to think about signs as things that work like words, and tend to think of them as the word being the sign, what it denotes being its object and the idea it imprints being the interpretant. Although somehow true, this view of the phenomenon obliterates its marvelous and useful complexity.

The sign is a triadic relation of which the first element that appears, when it is functioning as a sign, is the representamen (and that is why we also call it 'sign'). The sign is *determined* by the object, and the interpretant is the *effect* it (the sign) produces. So it is correct to say that the object causes the interpretant, but this causation is indirect: it happens *mediated* by the representamen in a process that is called *semiosis*. Let us not lose generality here: first, when we talk about the effects of a sign—the interpretant—we are talking about something that is not very well defined at this moment. We can be talking about the specific effect of it in a mind at a defined moment, or we can be talking about all the possible effects that it may produce, or even about its long-term effect, for instance. Second, when we talk about a mediated relation between interpretant and object, let us not forget that it makes the object itself unreachable to the interpreter—in the cases where there is an interpreter, usually the mind affected by the interpretant. Of course, there are different degrees of 'unreachability' of the object of the sign, depending on various factors.

It sounds strange to think of signs being determined by their objects. We are used to think about words being caused by their utterers—and it is true, signs are determined by their objects, although caused by their utterers. The uttering of a word is an act of semiosis, where the interpretant is the word uttered, the object is whatever the utterer wants to mean by this word, and the representamen may be the specific situation that made the utterer choose this meaning—and no other—, which in its turn may also be an interpretant

of whatever causes such desire to communicate (signs can "concatenate" in series of semiosis, possibly generating a "train of thought"). And this is the first moment we talk about "communication", and let it be the last. For when a paramecium finds a stimulus that makes it move towards a source of food, it is perfectly clear that the stimulus worked as a sign—a representamen of the source of food, which is its object, having the movement as the interpretant. But there is no "communication".

One can always argue that in the example above the paramecium's movement is caused by a series of phenomena that are of the nature of secondness, and the paramecium is not conscious of the sign action. However, consciousness is not a prerequisite for a triadic mediated relation to take place. What we need to keep in mind, at least for the moment, is that the sign is a triadic relation of mediation. This brings us to the classification of the signs.

### 2.4 Types of sign

The only complete classification of signs—in ten classes—that Peirce left us was regarded as incomplete by himself<sup>2</sup>. What follows is the minimum necessary to understand some concepts related to the process of reasoning and, albeit poorly, the process of semiosis.

The principle of this classification is to classify the three relata of the sign according to the categories. In Peirce's words, "signs are divisible by three trichotomies; first, according as the sign in itself is a mere quality [firstness], is an actual existent [secondness], or is a general law [thirdness]; secondly, according as the relation of the sign to its object consists in the sign's having some character in itself [firstness], or in some existential relation [secondness] to that object, or in its relation to an interpretant [thirdness]; thirdly, according as its Interpretant represents it as a sign of possibility [firstness] or as a sign of fact [secondness] or a sign of reason [thirdness]" (CP 2.243, 1903). Therefore there are three trichotomies, depicted in Table 1 that would give rise to  $3^3 = 27$  classes of signs, but since some of them are impossible, there are only ten of them. It

Table 1: Three trichotomies.

| Category   | Trichotomy    |        |              |
|------------|---------------|--------|--------------|
|            | Representamen | Object | Interpretant |
| Firstness  | Qualisign     | Icon   | Rheme        |
| Secondness | Sinsign       | Index  | Dicent       |
| Thirdness  | Legisign      | Symbol | Argument     |

happens due to some limitations: a qualisign cannot be an index, and an icon cannot be a proposition, for instance.

<sup>2</sup>Peirce wrote a footnote (CP 4.536, 1905) about sixty-six classes of sign and in a letter to Lady Welby (CP 8.343, 1908) he stated that the analysis of the classes of the signs implied the analysis of  $3^{10}$ , or 59,049 possible configurations, reason why he left it for "future explorers".

For illustration purposes only, the ten possible classes of sign are depicted in Table 2. There we can see that almost imperceptible things such as a sensation, and very complex ones such as arguments are signs. All can be so, according to Peirce, including ourselves, the people. The fact that

Table 2: Ten classes of sign, adapted from CP 2.254-264 and [5].

| Representamen | Object | Interpretant | Example  |
|---------------|--------|--------------|--|
| Qualisign     | Icon   | Rheme        | A sensation of redness                         |
| Sinsign       | Icon   | Rheme        | An individual diagram                          |
| Sinsign       | Index  | Rheme        | A spontaneous cry                              |
| Sinsign       | Index  | Dicent       | A thermometer                                  |
| Legisign      | Icon   | Rheme        | A traffic sign (“deers”) in the traffic manual |
| Legisign      | Index  | Rheme        | A demonstrative pronoun                        |
| Legisign      | Index  | Dicent       | A traffic sign on the street                   |
| Legisign      | Symbol | Rheme        | Any substantive                                |
| Legisign      | Symbol | Dicent       | A proposition                                  |
| Legisign      | Symbol | Argument     | A syllogism                                    |

this classification is universal brings a common basis for the analysis of complex phenomena, including the different kinds of inference, derived from the last class of signs showed above.

## 2.5 Types of argument

The three kinds of arguments related to Peirce’s categories (CP 2.96, 1902) are:

- a *Deduction* is an argument where the facts stated in the premises are an index of the fact stated in the conclusion. One example is that if we have a bag full of balls known to be of the same color and pull one of them to verify that it is red, it is unavoidable to deduce that all other balls in the bag are also red.
- an *Abduction*: here, the facts in the premises present a similarity with the facts in the conclusion, what does not regard the later to be true —and what makes the premises an icon of the conclusion. It is the abductive ratiocination that allows new hypothesis in science. For example, if we have a bag full of red balls and we see a red ball on the floor, we can (abductively) infer that the ball on the floor came from the bag, which is a conclusion to be investigated, namely a hypothesis.
- an *Induction*, in Peirce’s own words, “is an Argument which sets out from a hypothesis, resulting from a previous Abduction, and from virtual predictions, drawn by Deduction, of the results of possible experiments, and having performed the experiments, concludes that the hypothesis is true in the measure in which those predictions are verified. [...] Since the significance of the facts stated in the premises depends upon their predictive character, which they could not have had if the conclusion had not been hypothetically entertained, they satisfy the definition of a Symbol of the fact stated

in the conclusion” (CP 2.96, 1902). Example: pull a ball from a bag full of balls. It is red. The second, and the third, up to the, say, thirtieth are also red. This leads to the conclusion that all remaining balls in the bag are red, too.

Nevertheless, Peirce’s thought about reasoning and inference go far beyond this mere classification.

## 2.6 Diagrammatic reasoning

Since diagrams are iconic signs whose semiosis is based on firstness and deductions, including mathematical ones, are signs based on secondness, we can appreciate the observational nature of abstraction and mathematical reasoning taking Peirce’s own word: “As to that process of abstraction, it is itself a sort of observation. The faculty which I call abstractive observation is one which ordinary people perfectly recognize, but for which the theories of philosophers sometimes hardly leave room. It is a familiar experience to every human being to wish for something quite beyond his present means, and to follow that wish by the question, ‘Should I wish for that thing just the same, if I had ample means to gratify it?’ To answer that question, he searches his heart, and in doing so makes what I term an abstractive observation. He makes in his imagination a sort of skeleton diagram, or outline sketch, of himself, considers what modifications the hypothetical state of things would require to be made in that picture, and then examines it, that is, *observes* what he has imagined, to see whether the same ardent desire is there to be discerned. By such a process, which is at bottom very much like mathematical reasoning, we can reach conclusions as to what *would be* true of signs in all cases, so long as the intelligence using them was scientific” (CP 2.227, 1897).

The complex processes that give rise to conclusions and scientific evolution can then be observed in the light of Peircean semiotics. What matters here is that it is possible to distinguish, in published works in the field of Mathematics, these two components of diagrammatic reasoning: the diagram, or diagrams, on which the ratiocination was based, and the deductions themselves; and this may point some very interesting consequences.

## 3. Some Computer Science concepts

We start with the famous decision problem and the solution given by Turing in [6].

### 3.1 Computable functions

Turing’s article was one of the first papers that defined what is now known by *effective procedure*; it also gave a solution to the decision problem, proving that it is impossible to have an effective procedure capable of determining the solution of any effective procedure. The definition of effective procedure as a procedure that can be performed by what we call now a Turing machine is well known. The

Turing machine is in fact the diagram used by Turing in his deduction; this work is a clear example of diagrammatic reasoning: it cannot be denied that the paper fully adheres to Peirce's considerations about abstractive observation.

But there is an interesting possibility if we take a Peircean semiotic approach to the decision problem. We have already seen that the so-called effective procedures presented by Turing are prominently “secondness-based”: once settled, a Turing machine follows its way no matter what, blind move after blind move. This is what makes them predictable, mathematically describable, in a word: effective. Let us notice only that it is the constructed effective procedures that are “secondness-based”, not the process of constructing such procedures.

The question formulated in the decision problem is if it is possible to find an effective procedure (let's call it  $\mathfrak{P}$ ) to decide the results of any effective procedure, including  $\mathfrak{P}$  itself. Having determined that what characterizes an effective procedure is the fact that it is secondness-based we can infer that it is not possible for an effective procedure to derive conclusions from a diagram if these conclusions are not present at the construction of the diagram, because these conclusions would demand some sort of firstness-based iconic reasoning in order to take place<sup>3</sup>. In other words,  $\mathfrak{P}$  cannot decide the results of a procedure whose results can be derived only by diagrammatic reasoning.

So, in order to use Peircean semiotics to prove that the decision problem has no solution it is sufficient to prove that there is at least one effective procedure (let's call it  $\mathfrak{E}$ ) whose results can be inferred only by diagrammatic reasoning. I suspect that  $\mathfrak{P}$  has this characteristic, but I leave it open; it is possible that any procedure that cannot be verified by trial and error (e.g. a procedure that may show a result for every input number) and that implements the consequences of a theorem that can be proved only by contradiction fall into this category. And this rises the suspicion that Peircean semiotics may also give a ground for the discussions among constructive and non-constructive Mathematics.

Of course the proof outlined above, if found, would lack mathematical validity since a formal mathematical definition of the concepts outlined is not given. But it seems to be an interesting field of research. Peirce himself was a mathematician, and one of his concerns was to find what he called a theory of the plan of demonstration (CP 5.162, 1903), a task that he seemed to tackle with his existential graphs, a work that he also left unfinished.

### 3.2 Computers

We have seen that if everything works well, it seems that we could consider computers as secondness-based machines, and we could think about them in terms of computable

functions —or in terms of “things that can be carried out by Turing machines”—, which are a prominently secondness-based form of ratiocination. But real computers are not only secondness-based machines. What if we take into consideration other categories? We tend to think that computers have a very well defined behavior —some of us would say that in fact they perform only computable actions—, and if we think about computers as the machines depicted by John von Neumann in his famous 1945 report [7], that are practical versions of Turing Machines —as Turing himself stated during a Lecture to the London Mathematical Society on February, 20<sup>th</sup>, 1947— we can say that, apart from some malfunctioning, there should not be a more predictable machine in the world.

But let us first consider a little the concept of “predictability”. What is “predictable comportment”? The formal definition of effective procedure is that of the computable function, so the outcome of a computable function should be the most predictable in a predictability scale. But there are other degrees of predictability. In fact, predictability has to do with what is habitual or general: it is a phenomenon of thirdness. And as such, it can be extended to phenomena that are not composed of secondness only. Take, for instance, the instruction “throw a die and show the result”. Let's call it  $\alpha$ . It has a predictable outcome, although not completely predictable: we know that we will get an integer number from 1 to 6, with a uniform probability of 1/6 each. Which number will be the outcome of any particular cast, no one knows. Nevertheless we can create a machine — a secondness-based, mechanical one— that can perform  $\alpha$ . We can say that its behavior would be somewhat predictable, albeit not computable. It is very different from the outcome of instruction  $\beta$ : “perform one of any possible actions”. It has completely unpredictable results, and it is hard to imagine a mechanical device that performs  $\beta$ , whose outcome is neither computable nor predictable.

Since modern computers, as any real object, presents the three categories, let us see what happens if we inject some controllable firstness into them, in the form of a proper source of entropy. Thanks to this source, it is possible to write programs that, in spite of the predictability of all its steps, can have an unpredictable outcome, just as instruction  $\alpha$ . It suffices to take some of the bits of the unpredictable stream that enters the machine, convert them in, say, integers from 1 to 6, and show the results. These values will be fully unpredictable and hence, not computable. And if we combine this firstness with proper thirdness, in the form of evil intentions, we can write a program that executes instruction  $\beta$ : we take same unpredictable bits from the incoming unpredictable stream, write them in a specific memory address and order the CPU to execute the instruction written on that address: we end up with a computer performing an equivalent to the instruction “perform one of any possible actions”, a truly non-computable step with truly

<sup>3</sup>By “present at the construction of the diagram” I mean present in a “secondness” way, not as an intention, for instance, nor as something that may be observed in the diagram after, but not before, its construction.

unpredictable outcome.

The only use of this program for now is to show that a modern computer can be purposely programmed to perform a non-predictable task. But it also brings attention to the fact that computers can perform actions that are not of a prominent secondness. The problem is that we are used to think about computers with tools of prominent secondness: computable functions, and it is very difficult to think about phenomena of firstness and thirdness within the framework of computable functions. Attaining ourselves to the predictability of prominent secondness-based tools may give rise to gaps in the final outcomes.

### 3.3 Programming

The task of programming is one of those which would benefit a lot from adopting a framework based on Peircean semiotic. Computer programs, as they are seen today — signs evolve with time—, are signs mediating between two sign systems that are very different of each other. On the one hand, programs have to fulfill their users expectations. On the other, they must compile and generate software that works, free of errors. The problem is that this second task carries primarily secondness-based semiosis, which may be very different of the categories needed for the first task.

Programs have to be written in what we call computer languages. The term “languages” here expresses badly what is really happening. We expect languages to be able to describe the world, or, as some may prefer, languages should enable us to describe the world<sup>4</sup>. According to Peircean semiotic scholars (see [8]) natural language is vague by its own nature, and it is this vagueness that, at the bottom line, allows them to describe everything, even those things that are yet to come. There is no such a thing as a “perfect” or “universal” language, capable of perfectly describe everything, leaving no room for doubt. Computer languages today are far from the vagueness of natural languages: they are mathematical specifications of the valid sequences of a predefined alphabet, created in order to allow the programmers to generate texts that may be uniquely transformed in a set of “actions” to be “performed” by a CPU, or as some may prefer, a set of electrical signals that will act in a defined circuitry —in a few words, a secondness-based tool.

This resolves some of the difficulties involved in writing and reading programs in current programming languages: unless the intentions behind the program can also be described as a phenomenon of secondness, the program itself does not contain what is necessary to express completely what it is for. Programs within this context do not, and never will, work like texts in natural language. So, depending on

<sup>4</sup>There is a very interesting discussion about languages and their limits in the field of semiotics, but we are not going to enter it here —it suffices to say that some believe that languages are composed of “signs” that have “meanings”, forming pairs of “significants” and “signifieds”, which is not the Peircean view.

the nature of the user’s specifications, there is nothing in the program itself that allows us to verify whether it fulfills them. The task of systems development can be seen as the struggle to conform a general description, that embeds all three Peircean categories, to programs that cannot be vague from the machine’s point of view, for the machine itself cannot deal with firstness or thirdness in a predictable way. Notice that even when we have at hand both the user’s specifications, in whichever language, and the program, nothing written in the program language directly links it to whatever may be vague in the specification. This fact may unavoidably detach the final product of the activity of programming from the reasons that led to it, specially if these reasons cannot be described in a “secondness” way.

Careful observation of the landscape of computer activities might show movements that seem to fulfill this gap. We can see the desire to surpass the limitations of “secondness-based” description in concepts like object orientation, development frameworks —instead of development languages— and gamification, not to mention that these limitations may explain the CASE tools<sup>5</sup> fiasco. There is also a huge demand to empower common people and enable them to broaden their control over their now ubiquitous computers, and this empowerment demands other ways to control the machine other than a thus limited framework. Peircean semiotic analysis shows that the solution for the blindness to the program’s purpose does not lie in the creation of a new computer language in the same way others have been constructed. So, the question is now: how to control computers without using a programming language that works in a so prominent “secondness” way?

### 3.4 Artificial Intelligence and beyond

The pursue for what we call Artificial Intelligence stumbles in the definition of what intelligence is. It does not help that Turing had proposed, in [9], a way to check if a behavior can be regarded as intelligent: it characterizes intelligence without saying a word about how to achieve it. The dream of a machine that “thinks” is earlier than the advent of electronic computers. Peirce himself reminds us, in [10] —a work about logical machines of his time— that the theme already appears, in an ironic way, in Jonathan Swift’s “Voyage to Laputa”. In the same paper Peirce reminds us that we do not want a machine that behaves with the initiative of a human being: we want them to stay submissive to our will. Since it is possible to change the behavior of today’s electronic computers in an unprecedented way, question is: are they able to learn new behaviors without having to be programmed in a modern computer language? And the answer, in my opinion, is “yes”.

This will probably evolve in two directions. One is through the improvement of machine learning. It has to

<sup>5</sup>Let us remember that one goal of these tools, when they were announced, was to eliminate the need of programming.

do with making computers do something that has not been programmed in advance. Maybe the most remarkable examples are the so-called neural networks. This can be improved: Peirce wrote about the process of learning as a habit change governed by some characteristic properties of the nervous system. Nevertheless these characteristics do not work in a fully deterministic way —for if they were completely deterministic, the act of learning would lack thirdness: “it is essential that there should be an element of chance” (CP 1.390). So, the proper implementation of machine intelligence implies in the presence of thirdness in a non-deterministic way in order to apply it to the process of semiosis —an entirely new field of study, in spite of the many ways that allows us to introduce indeterminacy in the computer’s outcome.

The other direction is natural evolution: this is the slow path that will naturally happen. Language is capable to deal with the world as it is; but the understanding of symbols relies on our collateral experience of their objects (CP 8.314, 1909, CP 8.183, undated). This collateral experience is hard to achieve in an ever evolving environment as is the computer landscape today. It lacks the stability necessary to construct a common language. But it is slowly moving to a more stable position: computers will become ever easier to use and to program, and people will become more and more savvy about them, to the point that it will not be necessary any special knowledge beyond the ordinary expected at the moment in order to make computers act as we would like them to act. A symptom of this movement is the number of study years needed to program a computer in the 60’s compared to the same number today. This slow pace can be, of course, accelerated by a comprehension of the Peircean semiotic framework and its application to Computer Science.

## 4. Conclusion

Unlike many other texts that try to apply Peircean semiotics to Computer Science, this one has been written without any new concept apart from Peirce’s theory. In spite of it, it seems that this simple move gave rise to new interesting ways to think about the foundations of Computer Science, with possible consequences to Artificial Intelligence, Systems Development, Programming and maybe some concepts in pure Mathematics.

I would like to thank Professor Winfried Nöth for his kind review of this text, which allowed to avoid many mistakes. The remaining mistakes are, of course, due only to the author.

## References

- [1] M. Nadin, “Processos semióticos e de informação: a semiótica da computação,” *TECCOGS - Revista Digital de Tecnologias Cognitivas*, vol. 5, pp. 89–121, 2011.
- [2] C. S. Peirce, *The Collected Papers of Charles Sanders Peirce*, C. Hartshorne, P. Weiss, A. W. Burks, Eds. Cambridge, MA, USA: Harvard University Press, 1931-1958.
- [3] L. Santaella, *Percepção: fenomenologia, ecologia, semiótica*, São Paulo, Brazil: Cengage Learning, 2012.
- [4] R. Descartes, *Discurso do Método*, Porto Alegre, Brazil: L&PM, 2004.
- [5] W. Nöth and L. Santaella, *Introdução à Semiótica*, Unpublished São Paulo, Brazil, 2015.
- [6] A. Turing, “On Computable Numbers, with an Application to the Entscheidungsproblem,” *Proceedings of the London Mathematical Society*, vol. 2, pp. 230–265, 1936.
- [7] J. von Neumann, “First draft of a report on the EDVAC,” *Annals of the History of Computing, IEEE*, vol. 15, pp. 27–75, 1993.
- [8] W. Nöth and L. Santaella, *Meanings and the vagueness of their embodiments*, in: T. Thellefsen, B. Sørensen and P. Copley (Eds.) *From First to Third via Cybersemiotics – A Festschrift Honoring Professor Søren Brier on the Occasion of his 60th Birthday*. Copenhagen: SL Forlagene, 2011. p. 247–282.
- [9] A. Turing, “Computing Machinery and Intelligence,” *Mind*, vol. 59(236), pp. 433–460, 1950.
- [10] C. S. Peirce, “Logical Machines,” *American Journal of Psychology*, vol. 1(1), pp. 165–170, 1887.

# Construct a Perfect Hash Function in Time Independent of the Size of Integers

Yijie Han<sup>1</sup>

<sup>1</sup>School of Computing and Engineering, University of Missouri at Kansas City, Kansas City, Missouri 64110, USA

**Abstract**— We present an algorithm for constructing a perfect hash function that takes  $O(n^4 \log n)$  time. This time is independent of size of the integers or the number of bits in the integers. Previous algorithms for constructing a perfect hash function have time dependent on the number of the bits in integers. Our result is achieved via an algorithm that packs the extracted bits for each integer to  $O(n)$  bits in  $O(n^2 \log^2 n)$  time. Perfect hash function constructed using our method allows a batch of  $n$  integers to be hashed in  $O(n)$  time.

**Keywords:** Hashing, perfect hash functions, integers.

## 1. Introduction

A perfect hash function is a hash function that has no collision for the integers to be hashed. Previous known perfect hash functions require construction time dependent on the number of bits of integers to be hashed. Thus when dealing with very large integers these perfect hash functions are at disadvantage as when we are constructing a perfect hash function for  $n$  integers the time for construction cannot be bounded by a polynomial of  $n$ . Earlier Fredman et al. provided a perfect hash function [1] which require  $O(n^3 \log m)$  time to construct, where  $\log m$  is the number of bits in an integer (i.e. integers to be hashed are taken from  $\{0, 1, \dots, m-1\}$ ). Dietzfelbinger et al. gave a randomized hash function in [3] and Raman showed [5] how to derandomize it to obtain a deterministic perfect hash function in time  $O(n^2 \log m)$ . Thus the construction time of these hash functions depends on the number of bits of integers and therefore cannot be classified exactly as polynomial time algorithms.

In this paper we give an algorithm for converting  $n$  integers of  $\Omega(n^2 \log n)$  bits to  $O(n)$  bits integers in  $O(n)$  time for hashing purpose. This conversion can be done after we computed shift distances  $d_1, d_2, d_3, \dots$  (to be explained in the following sections). These shift distances can be computed in  $O(n^2 \log^2 n)$  time. Because we require integers having  $\Omega(n^2 \log n)$  bits the construction time for a perfect hash function from Raman's algorithm can be bounded by  $O(n^2 \log m + n^2 \log^2 n) = O(n^4 \log n)$ . In our method hashing has to be done in batches of  $n$  integers and the hash time for  $n$  integers is  $O(n)$ .

We can reduce the time for computing the shift distances to  $O(n^2 \log n)$ . Then the conversion time for  $n$  integers has to be increased to  $O(n \log n)$ . The hashing of  $n$  integers would take  $O(n \log n)$  time in this case.

After integers are converted to  $O(n)$  bits integers then the construction of a perfect hash function in the algorithm of Fredman et al. would require only  $O(n^4)$  time and the construction of a perfect hash function in Raman's algorithm would require  $O(n^3)$  time. However because in our algorithm we require integers having  $\Omega(n^2 \log n)$  bits and therefore the construction of a perfect hash function via the algorithm of Fredman et al. requires  $O(n^5 \log n)$  time and via Raman's algorithm requires  $O(n^4 \log n)$  time.

Current integer sorting can be done in  $O(n \log \log n)$  time [4]. The algorithms presented in this paper may help in the search of an optimal algorithm for integer sorting.

## 2. Extracting Bits

To construct a perfect hash function for  $n$  integers we will first sort these  $n$  integers in  $O(n \log \log n)$  time using the current best integer sorting algorithm of Han [4]. Let  $a_0 < a_1 < a_2 < \dots < a_{n-1}$  be the sorted integers (all integers with the same value can be excluded except one). Let  $msb(a)$  be the index of the most significant bit of  $a$  that is 1, where index counts starting from least significant bit at 0. We compute  $m(i) = msb(a_i \oplus a_{i+1})$ ,  $0 \leq i < n-1$ , where  $\oplus$  is the bit-wise exclusive-or operation. We take all the bits indexed in  $M = \{m(i) \mid i = 0, 1, \dots, n-2\}$  for each integer  $a_j$  to form  $a'_j$ ,  $0 \leq j < n$ . Thus now each  $a'_j$  has at most  $n-1$  bits. If  $a_i \neq a_j$  then  $a'_i \neq a'_j$ .

**Example 1:** Let  $a_0 = 0100001$ ,  $a_1 = 0100101$ ,  $a_2 = 0110000$ , then the most significant bit  $a_0$  and  $a_1$  differ is the 2nd bit (counting from the least significant bit) and the most significant bit  $a_1$  and  $a_2$  differ is the 4th bit, thus  $M = \{2, 4\}$  and  $a'_0 = 00$ ,  $a'_1 = 01$ ,  $a'_2 = 10$  (the 2nd and the 4th bits).  $\square$

There are two problems here. The first problem is how to obtain set  $M$ . The second problem is after bits are extracted how do we pack them to  $n-1$  consecutive bits.

First we will not compute  $M$  but instead compute  $M' = \{2^{msb(a_i \oplus a_{i+1})} \mid i = 0, 1, \dots, n-2\}$ . In fact even  $M'$  is difficult to compute and we will adapt our method.

The second problem will be solved in the following sections.

As we said that  $M'$  is difficult to compute. We will instead use the least significant bit. Let  $lsb(a)$  be the index of the least significant bit of  $a$  that is 1. Note that it will be easy to extract the least significant bit that is 1. To extract the least significant bit of  $a$  that is 1 simply do  $2^{lsb(a)} = (a \oplus (a-1)) + 1)/2$ . Next we will view each integer reverse-wards, that is, we view the least significant bit as the most significant bit and the most significant bit as the least significant bit. As will be shown that we will use this order to sort the  $n$  integers. We will call this order as the least significant bit order. Now the approach we described earlier will work if we sorted integers by the least significant bit order, i.e. say  $a''_0, a''_1, \dots, a''_{n-1}$  are the integers sorted by the least significant bit order, then the least significant bit that  $a''_i$  and  $a''_{i+1}$  differ,  $i = 0, 1, \dots, n-2$ , will give us  $n-1$  bits that make integers differ between each other. We let  $m'(i) = lsb(a''_i \oplus a''_{i+1})$ . There are at most  $n-1$  different values for  $m'(i)$ ,  $0 \leq i < n-1$ . Now to sort integers by the least significant bit order we use comparison sorting and compare integers  $a$  and  $b$  by examining  $(2^{lsb(a \oplus b)} \vee a) == a$  and  $(2^{lsb(a \oplus b)} \vee b) == b$ , where  $\vee$  is the bit-wise OR operation. If  $(2^{lsb(a \oplus b)} \vee a)$  is equal to  $a$  then  $a$  is "larger" than  $b$  in the least significant bit order.

After we get  $a''_0, a''_1, \dots, a''_{n-1}$  we then compute  $L_i = 2^{m'(i)} = 2^{lsb(a''_i \oplus a''_{i+1})}$ ,  $i = 0, 1, \dots, n-2$ , to get the least significant bits. Let  $L = \bigvee_{i=0}^{n-2} L_i$ .  $L$  provides the mask for us to extract out needed bits as we now do  $b_i = a_i \wedge L$ ,  $i = 0, 1, \dots, n-1$ , where  $\wedge$  is the bit-wise AND operation. Note that no more than  $n-1$  bits will be extracted from each integer.

### 3. Pack Bits

In the last section we showed how to extract at most  $n-1$  bits from each integer. These extracted bits need to be packed. In this section we will show how to pack into  $O(n)$  bits with integers of  $\Omega(n^4)$  bits. We will compute shift distances  $d_1, d_2, \dots$ . These shift distances can be computed in  $O(n^4)$  time. In the later sections we show how to improve the algorithm to work with integers of  $\Omega(n^2 \log n)$  bits.

Without loss of generality we assume that all  $L_i$ 's are different.

Note that in Fredman and Willard [2] Lemma 2 it was shown that these extracted  $n-1$  bits (scattered bits of 1's among  $\log m$  bits) can be packed to  $n^4$  bits by multiplying a multiplier and this multiplier can be computed in  $O(n^4)$  time. However, the method in [2] requires that set  $M_2 = \{m'(i) \mid i = 0, 1, \dots, n-1\}$  be obtained. In the last section we only obtained  $M_1 = \{2^{m'(i)} \mid i = 0, 1, \dots, n-1\}$ . To obtain  $M_2$  from  $M_1$  we need to apply a logarithm which may not be readily available. Fredman and Willard's method do have the advantage of hashing one integer at a time. Our method requires the hashing of batches of  $n$  integers at a time.

Because there are only  $n-1$  extracted bits there are less than  $n^2$  different distances (the set of distances is  $D = \{|m'(i) - m'(j)|, 0 \leq i, j \leq n-2\}$ ) among them. By trying out  $n^2$  different distances  $n+1, n+2, \dots, n+n^2$  (the reason we have this additive  $n$  is because we want to shift at least  $n$  bits) we will find a distance not in  $D$ . Because we did not obtain  $M_2$  we check a distance  $d_1$ ,  $d_1 = n+1, n+2, \dots, n+n^2$ , by trying out  $(L \vee (L \rightarrow d_1)) == (L + (L \rightarrow d_1))$ , where  $\rightarrow d_1$  is shift  $d_1$  bits to the right. If it is equal then distance  $d_1$  is available (not in  $D$ ). Let  $b_i = a_i \wedge L$  contain the extracted bits. After we find an available distance  $d_1$  we do  $b_{i/2} = b_i \vee (b_{i+1} \rightarrow d_1)$ ,  $i = 0, 2, 4, \dots$ . We also do  $L'_0 = L_0 \leftarrow d_1$ ,  $L'_1 = L_0$ ,  $L_i = L_i \vee (L_i \rightarrow d_1)$ ,  $i = 0, 1, \dots, n-2$ , and  $L = L \vee (L \rightarrow d_1)$ .  $L'_i$ 's indicate the location of  $m'(0)$ 's and they will be used later to extract the packed bits. Now we have  $n/2$  integers and each integer has  $2(n-1)$  extracted bits. Therefore there are no more than  $4n^2$  distances among them. We pick an available distance  $d_2$  among  $n+1, n+2, \dots, n+4n^2$  using the same method and then do  $b_{i/2} = b_i \vee (b_{i+1} \rightarrow d_2)$ ,  $i = 0, 2, 4, \dots$ . Also  $L'_2 = L'_0$ ,  $L'_0 = L'_0 \leftarrow d_2$ ,  $L'_3 = L'_1$ ,  $L'_1 = L'_1 \leftarrow d_2$ ,  $L_i = L_i \vee (L_i \rightarrow d_2)$ ,  $i = 0, 1, \dots, n-2$ , and  $L = L \vee (L \rightarrow d_2)$ . After we do this  $\log n$  times we have all the  $n(n-1)$  extracted bits in all  $n$  integers  $\vee$ -ed into one integer  $b_0$ . The time spent is  $O(n^4)$ . We then extract the  $m'(i)$ -th bits by doing  $m_i = b_0 \wedge L_i$ . Note that  $m_i$  contains the  $m'(i)$ -th bits of all  $a_j$ ,  $j = 0, 1, \dots, n-1$ . Note also that the bit patterns in  $L_i$  and  $L_j$  are exactly the same (i.e.  $L_j$  can be obtained from  $L_i$  by shifting  $L_i \log(L_j/L_i)$  bits (because logarithmic function is not readily available we can substitute shifting  $\log(L_j/L_i)$  bits by multiplying  $L_j/L_i$ )). Now we pack integers together by doing

```
for(i = 1; i <= n - 2; i++)
{
    m_0 = m_0 \vee (m_i * L_0 / (2^i L_i));
}
```

Because we added an addend  $n$  when we do shifting previously and therefore there will be enough space when we pack integers here. After integers are packed we can then obtain packed integer  $b'_i$  (packed from  $b_i$ ) as  $L''_i = (L'_i \leftarrow 1) - (L'_i \rightarrow (n-1))$  (obtain mask and  $\leftarrow 1$  is shift left by 1 bit) and  $b'_i = m_0 \wedge L''_i$ . Now to move extracted bits to the least significant  $n-1$  bits do  $b'_i = b'_i / (L'_i \rightarrow (n-1))$ .

**Example 2:** Let  $b_0 = 000a0a00000aa0$ ,  $b_1 = 000b0b00000bb0$ ,  $b_2 = 000c0c00000cc0$ ,  $b_3 = 000d0d00000dd0$ , where  $a, b, c, d$  are extracted bits.  $L_0 = 00010000000000$ ,  $L_1 = 00000100000000$ ,  $L_2 = 00000000000100$ ,  $L_3 = 00000000000010$ ,  $L = 00010100000110$ .

Take  $d_1 = 5$ . Then do  $b_0 = b_0 \vee (b_1 \rightarrow 5) = 000a0a00b0baa000bb0$ ,  $b_1 = b_2 \vee (b_3 \rightarrow 5) = 000c0c00d0dcc000dd0$ ,  $L'_0 = L_0 \leftarrow 5 = 000100000000000000$ ,  $L'_1 = L_0 = 000000010000000000$  (thus  $L'_0$  indicates the position of first  $a$  or  $c$  and  $L'_1$

indicates the position of first  $b$  or  $d$ ).  $L_0 = L_0 \vee (L_0 \rightarrow 5) = 000100001000000000$ ,  $L_1 = L_1 \vee (L_1 \rightarrow 5) = 000001000010000000$ ,  $L_2 = L_2 \vee (L_2 \rightarrow 5) = 0000000000010000100$ ,  $L_3 = L_3 \vee (L_3 \rightarrow 5) = 0000000000001000010$ ,  $L = L \vee (L \rightarrow 5) = 0001010010111000110$ .

Take  $d_2 = 15$ . Then do  $b_0 = b_0 \vee (b_1 \rightarrow 15) = 000a0a00b0baa000bbc0c00d0dcc000dd0$ ,  $L'_2 = L'_0 = 00000000000000000000000000000000$ ,  $L'_0 = L'_0 \leftarrow 15 = 00010000000000000000000000000000$ ,  $L'_3 = L'_1 = 00000000000000000000000000000000$ ,  $L'_1 = L'_1 \leftarrow 15 = 00000000100000000000000000000000$  (thus  $L'_0$  indicates the position of first  $a$ ,  $L'_1$  indicates the position of first  $b$ ,  $L'_2$  indicates the position of first  $c$ ,  $L'_3$  indicates the position of first  $d$ .) And

$$\begin{aligned} L_0 &= L_0 \vee (L_0 \rightarrow 15) = 000100001000000000010000100000000000, \\ L_1 &= L_1 \vee (L_1 \rightarrow 15) = 000001000010000000000100001000000000, \\ L_2 &= L_2 \vee (L_2 \rightarrow 15) = 0000000000010000100000000010000100, \\ L_3 &= L_3 \vee (L_3 \rightarrow 15) = 0000000000001000010000000001000010, \\ L &= L \vee (L \rightarrow 15) = 0001010010111000111010010111000110. \end{aligned}$$

Here we see that  $L_0, L_1, L_2, L_3$  have the same pattern and they differ by only a shift of bits.

Now compute

$$\begin{aligned} m_0 &= b_0 \wedge L_0 = \\ &000a0a00b0baa000bbc0c00d0dcc000dd0 \wedge \\ &000100001000000000010000100000000000 = \\ &000a0000b000000000c0000d0000000000, \end{aligned}$$

$$\begin{aligned} m_1 &= b_0 \wedge L_1 = \\ &000a0a00b0baa000bbc0c00d0dcc000dd0 \wedge \\ &000001000010000000000100001000000000 = \\ &00000a0000b000000000c0000d000000000, \end{aligned}$$

$$\begin{aligned} m_2 &= b_0 \wedge L_2 = \\ &000a0a00b0baa000bbc0c00d0dcc000dd0 \wedge \\ &0000000000010000100000000010000100 = \\ &00000000000a0000b000000000c0000d00, \end{aligned}$$

$$\begin{aligned} m_3 &= b_0 \wedge L_3 = \\ &000a0a00b0baa000bbc0c00d0dcc000dd0 \wedge \\ &0000000000001000010000000001000010 = \\ &000000000000a0000b000000000c0000d0. \end{aligned}$$

Now do

$$\begin{aligned} m_0 &= m_0 \vee (m_1 * L_0 / (2L_1)) = m_0 \vee (m_1 * 2); \\ m_0 &= m_0 \vee (m_2 * L_0 / (4L_2)) = m_0 \vee (m_2 * 2^6); \\ m_0 &= m_0 \vee (m_3 * L_0 / (8L_3)) = m_0 \vee (m_3 * 2^6); \end{aligned}$$

We get that  $m_0 = 000aaaa0bbbb000000cccc0dddd0000000$ .

That is, bits for each integer are packed together.

Now do

$$\begin{aligned} L''_0 &= (L'_0 \leftarrow 1) - (L'_0 \rightarrow 3) = \\ &00100000000000000000000000000000 - \\ &00000010000000000000000000000000 = \\ &00011110000000000000000000000000 \text{ (obtain mask)} \end{aligned}$$

and

$$\begin{aligned} b'_0 &= m_0 \wedge L''_0 = \\ &000aaaa0bbbb000000cccc0dddd0000000 \wedge \\ &00011110000000000000000000000000 = \\ &000aaaa00000000000000000000000000 \end{aligned}$$

Now do

$$b'_0 = b'_0 / (L'_0 \rightarrow 3) = aaaa.$$

Similarly  $bbbb$  for  $b'_1$ ,  $cccc$  for  $b'_2$ ,  $dddd$  for  $b'_3$  can also be extracted.  $\square$

**Theorem 1:** The  $n - 1$  bits with indices in  $\{m'(0), m'(1), \dots, m'(n - 1)\}$  can be packed to  $n - 1$  bits in  $O(n^4)$  time for constructing a perfect hash function in  $O(n^2 \log m + n^4) = O(n^6)$  time, thereafter the packing and hashing of a batch of  $n$  integers take  $O(n)$  time.  $\square$

Here  $\log m = O(n^4)$  because we require integers have  $\Omega(n^4)$  bits.

Hashing takes  $O(n)$  time comes from the nature of the constructed hash functions [1], [5].

Note that the time  $O(n^4)$  in Theorem 1 is actually the time for computing distances  $d_1, d_2, \dots$ . Thus this time affects the time for constructing a perfect hash function as a perfect hash function can be constructed by first packing the extracted bits to  $n - 1$  bits and then use Raman's algorithm [5] to construct the perfect hash function in  $O(n^2 \log m) = O(n^3)$  time. After distanced  $d_1, d_2, \dots$  have been computed we can then pack the extracted bits for a batch of  $n$  integers in  $O(n)$  time by the algorithm presented in this section. The only requirement for packing is that integers have  $\Omega(n^4)$  bits. Because of this the construction time for a perfect hash function in Raman's algorithm becomes  $O(n^6)$ . Of course if integers have less than  $n^4$  bits then we can directly construct a perfect hash function in  $O(n^6)$  time using Raman's algorithm.

## 4. Construction in $O(n^2 \log n)$ Time

In this section we show the improvement to pack the extracted bits to  $O(n)$  bits in  $O(n^2 \log n)$  time.

First note that the time  $O(n^4)$  for packing in the last section can really be reduced to  $O(n^3)$  time by a better analysis. Note that after we  $\vee$ -ed  $b_0, b_1, \dots, b_{2^i-1}$  integers into one integer  $b$  the number of possible distances in  $b$  is much smaller than  $(2^i(n - 1))^2$ . This is because the



distance in  $b$  between two bits can be represented as  $\pm k + \delta_1 d_1 + \delta_2 d_2 + \dots + \delta_i d_i$ , where  $k$  is a distance taking from the  $(n-1)^2$  distances in a  $b_j$  and each  $\delta_j$  can assume the value of 0 or 1. Thus the number of possible distances is bounded by  $2(n-1)^2 \cdot 2^i$ . This analysis will bound in the the number of possible distances to  $O(n^3)$  and therefore the time complexity of the algorithm in Theorem 1 can be reduced to  $O(n^3)$ .

In last section we first  $\vee$ -ed all bits into  $b_0$ . This requires  $O(n^4)$  bits as there are a total of  $n(n-1)$  extracted bits and therefore there are  $O(n^4)$  possible distances among these bits. Here we do this: we find an available distance  $d_1$  and do  $b_{i/2} = b_i \vee (b_{i+1} \rightarrow d_1)$ . We have 2 integers  $\vee$ -ed into 1 integer. Thus we have now  $n/2$  integers remain. Instead of continuing  $\vee$ -ing integers together we now extract half of the bits corresponding to indices  $m'(i)$ ,  $i = 0, 1, \dots, (n-1)/2 - 1$  to one integer and another half of the bits corresponding indices  $m'(i)$ ,  $i = (n-1)/2, (n-1)/2 + 1, \dots, n-1$  to another integer. Before extracting the number of possible distances in the integer is  $2(n-1)^2$ . After extracting them into 2 integers each integer has the number of possible distances  $2((n-1)/2)^2 = (n-1)^2/2$  (each  $\vee$ -ed in integer has only  $(n-1)/2$  bits now). The situation is basically the same as the analysis we had for the  $O(n^3)$  possible distances. This is equivalent to say that we have put 2 integers into 2 integers and the number of possible distances among bits decreased from  $(n-1)^2$  to  $(n-1)^2/2$  for each integer. However, for the further  $\vee$ -ing together integers we have now to pick a  $d_2$  for  $(n-1)/2$  integers and pick another  $d'_2$  for the other  $(n-1)/2$  integers and thus the number of possible distances is  $(n-1)^2/2 + (n-1)^2/2 = (n-1)^2$ . We can repeat this  $\log n$  times, each time  $\vee$ -ing 2 integers into 1 integer and then extract out 2 integers from this integer. Every time we extract 2 integers the number of  $m'(i)$ 's the bits corresponds to in an integer is divided by 2. After  $\log n$  times the number of  $m'(i)$  the bits in an integer correspond to is reduced to 1, i.e. the  $m'(i)$ -th bits of all integers are now in one integer and the  $m'(j)$ -th bits of all integers for  $j \neq i$  are now in another integer. We repeated  $\log n$  times and each time we have to spend  $O(n^2)$  time searching for  $d_i$ 's. Thus the overall time of our algorithm is  $O(n^2 \log n)$ .

**Theorem 2:** The  $n-1$  bits with indices in  $\{m'(0), m'(1), \dots, m'(n-1)\}$  can be packed to  $n-1$  bits in  $O(n^2 \log n)$  time for constructing a perfect hash function in  $O(n^2 \log m + n^2 \log n) = O(n^4)$  time, thereafter the packing and hashing of a batch of  $n$  integers take  $O(n \log n)$  time.

□

Note that although we packed the extracted bits to  $n-1$  bits and therefore it seems that  $\log m = n-1$  in Theorem 2. However our algorithm assumed that integers having  $\Omega(n^2)$  bits and this requirement sets  $\log m = O(n^2)$  in Theorem 2.

Note also that although Theorem 2 reduced the computing time for finding  $d_1, d_2, \dots$  to  $O(n^2 \log n)$  it has the disad-

vantage of packing and hashing a batch of  $n$  integers in  $O(n \log n)$  time instead of the  $O(n)$  time we have achieved in the last section. In the next section we will show how to reduce the time for computing  $d_1, d_2, \dots$ , to  $O(n^2 \log^2 n)$  time while keep the packing and hashing time to  $O(n)$ .

## 5. Achieving $O(n)$ Packing Time with $O(n^2 \log^2 n)$ Time for Construction

Here we first compute  $d_j$  and do  $b_{i/2} = b_i \vee (b_{i+1} \rightarrow d_j)$ ,  $i = 0, 2, 4, \dots$ , for  $j = 1, 2, \dots, \log \log n$ . We have thus  $\vee$ -ed  $\log n$  integers into one integer and we have only  $n/\log n$  integers remain. Now the number of possible distances in each integer becomes  $(\log n)(n-1)^2$ . We now take over the algorithm in last section. That is we will  $\vee$  two integers into one integer and then extract two integers from one integer. We need to do this  $O(\log n)$  times. In doing so the total number of distances is kept at  $(\log n)(n-1)^2$  and each time we do this we spent  $O(n/\log n)$  time as we have only  $n/\log n$  integer. This makes the overall time for packing become  $O(n)$ . The time for computing distances  $d_1, d_2, \dots$  becomes  $O(n^2 \log^2 n)$  as we repeated  $O(\log n)$  times and each time expending  $O(n^2 \log n)$  time.

**Theorem 3:** The  $n-1$  bits with indices in  $\{m'(0), m'(1), \dots, m'(n-1)\}$  can be packed to  $n-1$  bits in  $O(n^2 \log^2 n)$  time for constructing a perfect hash function in  $O(n^2 \log m + n^2 \log^2 n) = O(n^4 \log n)$  time, thereafter the packing and hashing of a batch of  $n$  integers take  $O(n)$  time. □

Here again we require that integers have  $\Omega(n^2 \log n)$  bits.

## References

- [1] M. L. Fredman, J. Komlós, E. Szemerédi. Storing a sparse table with  $O(1)$  worst case access time. *J. ACM*, Vol. 31, No. 3, 538-544(1984).
- [2] M. L. Fredman, D. E. Willard. BLASTING through the information theoretic barrier with FUSION TREES. *Proc. 1990 ACM Symp. on Theory of Computing*, 1-7(1990)
- [3] M. Dietzfelbinger, T. Hagerup, J. Katajainen, M. Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms* **25**, 19-51(1997).
- [4] Y. Han. Deterministic sorting in  $O(n \log \log n)$  time and linear space. *Journal of Algorithms*, 50, 96-105(2004).
- [5] R. Raman. Priority queues: small, monotone and trans-dichotomous. *Proc. 1996 European Symp. on Algorithms, Lecture Notes in Computer Science 1136*, 121-137(1996).

# Sieving Interior Collinear Points for Convex Hull Algorithms

Dr. Michael Scherger  
 Department of Computer Science  
 Texas Christian University  
 Fort Worth, TX, 76129, USA  
 Email: [m.scherger@tcu.edu](mailto:m.scherger@tcu.edu)

Mr. Estaban Kleckner  
 Department of Computer Science  
 Texas Christian University  
 Fort Worth, TX, 76129, USA  
 Email: [e.kleckner@tcu.edu](mailto:e.kleckner@tcu.edu)

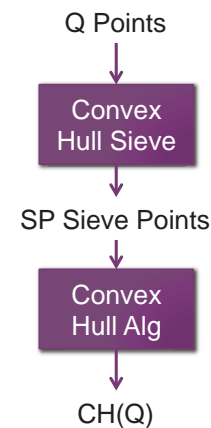
**Abstract** – (FCS'15) *The convex hull of a set of points,  $Q$ , denoted by  $CH(Q)$ , is the smallest convex polygon that encloses all the points of  $P$ . Each point in the convex hull is either on the boundary of  $P$  or in its interior. Traditional algorithms such as Graham's Scan, Jarvis March, and QuickHull use all the points in  $Q$  as input to obtain the  $CH(Q)$ . This research developed a pre-processing “sieve” algorithm that reduces the number of points necessary to compute the  $CH(Q)$ . Given a set  $Q$  with integer coordinates, an  $O(n)$  sieve algorithm “filters” interior collinear points from  $Q$  into a reduced set of sieve points  $SP$ . This reduces the number of points necessary to compute the  $CH(Q)$  and improves the overall performance of the traditional convex hull algorithms. The sieve points remain a superset of the hull points;  $SP \supset CH(Q)$ . A program has been created to test the correctness of the sieve algorithm[4][8][9]. Using a uniform distribution of points, the program has shown that  $|SP| \ll |Q|$ .*

**Keywords:** Computational geometry, convex hull, sieve algorithms.

## 1 Introduction

Computing a convex hull of a set of points is a well-studied research topic in computational geometry. A typical course in analysis of algorithms often covers traditional convex hull algorithms such as Graham Scan, Jarvis March, and QuickHull [4][8][9]. It is widely known that the complexity of Graham Scan is  $O(n \lg n)$  since the points must be sorted radially about the lowest-leftmost point. Over the years, other algorithms and improvements to existing algorithms have been developed.

Other improvements to existing convex hull involve heuristics such as the Akl-Toussaint heuristic that finds upper, lower, left-most, and right-most extreme points to form an irregular convex quadrilateral. Points within this convex quadrilateral can safely be eliminated, thus reducing the number of points for subsequent convex hull algorithms. Many of these improvements involve comparing two or more points together to satisfy some criterion. It is the number of point-to-point comparisons that is used to measure the complexity of a convex hull algorithm.



### 1.1 Research Overview

Figure 1: Data flow of points through the sieving algorithm followed by computing the convex hull.

This research presents a pre-processing algorithm that can be applied to a set of points prior to calling a traditional convex hull algorithm. This research considers points with integer coordinates in the first quadrant. This pre-processing step “sieves” or “filters” the interior collinear points to eliminate interior points prior to computing the convex hull

using a traditional algorithm. As shown in Figure 1, the output from this sieve is a set of points that are a superset of the convex hull points.

The sieving algorithm uses a form of “bucket sort” that sort/organizes points with corresponding coordinates in the first dimension, then uses a second “bucket sort” in the second dimension to arrange the points in ascending order. The sieving algorithm does not compare any point with any other point; i.e. no comparisons. The complexity of the algorithm is  $O(n)$  and will be further explained later in this paper.

## 1.2 Organization

The organization of this paper is the following. Section 2 will present some background information on sieving algorithms, traditional convex hull algorithms, and sorting in linear time using bucket sort. Section 3 will present and explain our algorithm and discuss the program used to test its correctness and gather timing results. Section 4 will present results from our algorithm testing for varying number of points and point densities. Section 5 will present some conclusions and future work for this research.

## 2 Background

Numerous algorithms and algorithmic techniques were researched and explored. The major components of this research are the concepts of sieving algorithms, traditional convex hull algorithms, and sorting in linear time using bucket sort.

### 2.1 Sieving Algorithms

A sieve, in the mathematical sense, is an algorithmic technique to filter or eliminate non-essential data to achieve a desired result. The classic sieving algorithm is the Sieve of Eratosthenes used to find all prime numbers less than or equal to some number  $n$  [4][6]. Named after the Greek mathematician, this prime number sieve algorithm repeatedly eliminates multiples of prime numbers ( $2p, 3p, 4p, \dots$ ) until the value of  $np$  is less than the  $\sqrt{n}$ . The overall complexity of the Sieve of Eratosthenes is  $O(\lg \lg n)$  [4].

A characteristic of sieve algorithms is that they often do not have to compare data values with

other data values: no comparisons and few (if any) data dependencies.

### 2.2 Traditional Convex Hull Algorithms

There are many algorithms that compute the convex hull of a set of points. Three classic algorithms are the Jarvis March (gift-wrapping) algorithm, Quickhull, and Graham Scan. The Jarvis March algorithm was developed in 1970 by Chand and Kapur and also independently in 1973 by R. Jarvis. [7] This algorithm begins with a known point on the hull,  $p_0$ , and compares polar angles of other points. It selects the next point on the hull that has the smallest polar coordinate and then the process repeats “wrapping” around the points to form the convex hull. The complexity of Jarvis March is  $O(nh)$  where  $n$  is the number of points, and  $h$  is the number of hull points.

The Quickhull algorithm was discovered by Eddy 1977 and also independently by Bykat in 1978. [1][9] It is analogous to the quicksort algorithm. The Quickhull algorithm finds two points with the minimum and maximum  $x$  coordinates to create a dividing line through the set of points creating an upper set and lower set of points. It next finds a point in  $P$  that is a maximum distance from the dividing line. All points lying in this triangle of points are excluded from the convex hull. Then the process repeats until no other hull points are discovered. The same procedure repeats for finding the lower convex hull. Finally the two sets of points are combined. The complexity of Quickhull is  $O(n \ln n)$  in the average case and  $O(n^2)$  in the worst case.

The Graham scan algorithm first finds the lowest right-most point  $p_0$ . This point is on the convex hull. Next, all points are sorted radially about  $p_0$ . Next, the first two points are pushed on a stack. If the top two points on the stack and the next point considered make a counter-clockwise turn, then the new point is pushed on the stack. If the points make a clockwise turn, then the top point on the stack is popped off. This process repeats until the three points considered make a counter-clockwise turn. The complexity of Graham scan is bounded by sorting and has an overall complexity of  $O(n \lg n)$ . [4][5][9][11]

## 2.3 Sorting in Linear Time Using Bucket Sort

Bucket sort is also known as bin sorting.[4] It can sort a set of data by partitioning the input array into buckets. Each bucket can then be sorted using a traditional sorting algorithm or recursively calling bucket sort again. The complexity of bucket sort depends on the number of buckets. If  $n$  is the input data size and  $M$  is the number of buckets, then the complexity of bucket sort is  $O(n+M)$ .

## 3 Methods

This section will present an overview of the convex hull sieve describing major stages of the algorithm. This section will also present a discussion of the full algorithm and implementation.

### 3.1 Sieve Algorithm Overview

For purposes of this discussion, the following example uses a set of 50 points. All input points to this algorithm have integer coordinates and are in the first quadrant. Figure 2 shows the initial set of points. Notice the  $x$ -collinearities (which are points that share a common  $x$ -coordinate) and  $y$ -collinearities (which are points that share a common  $y$ -coordinate). The set of input points are in no particular order (random permutation of the list of points).

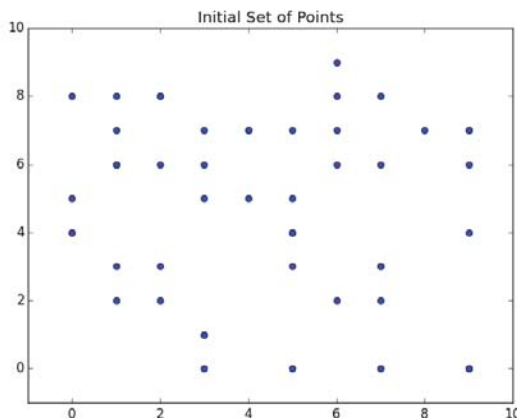


Figure 2: Initial set of 50 points.

The algorithm begins by first identifying  $y$ -collinearities. This is accomplished by traversing the list of points and placing each point in a common bucket if they share a common  $y$ -coordinate. Again,

because the initial list of points is random, the points in each “ $y$ -bucket” are also randomized (unsorted). The points in each  $y$ -bucket are passed to a bucket sort again and lexicographically ordered by  $x$ -coordinate. Any point between the first point and last point in each  $y$ -bucket is removed because it is an interior point and not on the convex hull. Figure 3 illustrates the  $y$ -collinear points in each bucket.

The next stage of the algorithm then eliminates the  $x$ -collinearities. The remaining points from the previous step are stored in respective “ $x$ -buckets” in lexicographic order. Any point between the first point and last point in each  $x$ -bucket is removed because it is also an interior point and not on the convex hull. Figure 4 illustrates the  $x$ -collinear points in each bucket.

The output of the sieving algorithm is a set of candidate points that are on the outer boundaries of the initial set of points. These points are either maxima or minima in each  $x$  and  $y$  directions. Figure 5 shows the resultant set of points with the  $x$ -collinearities removed. The output points are used as input points into a traditional convex hull algorithm and the convex hull points are determined. The final convex hull is also shown in Figure 5.

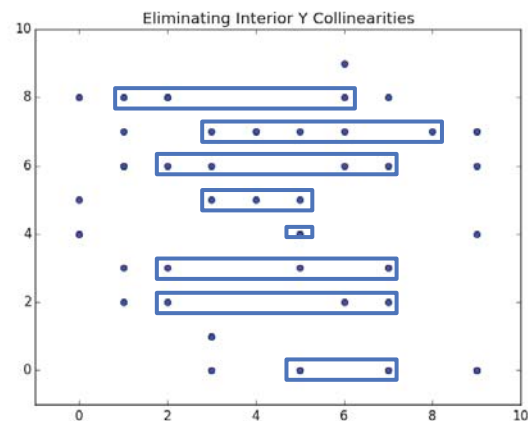


Figure 3: Identifying and eliminating  $y$ -collinearities.

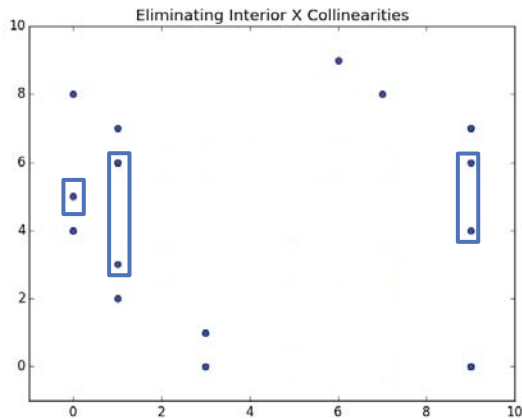


Figure 4: Identifying and eliminating x-collinearities.

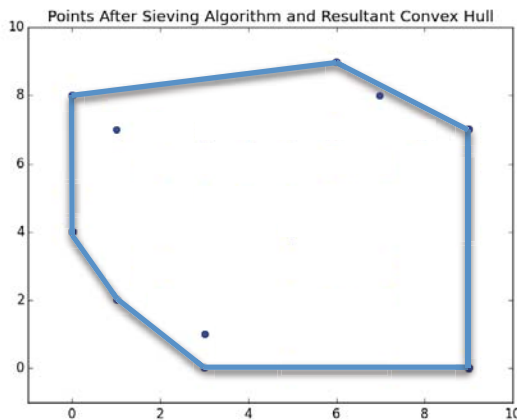


Figure 5: Resultant points after sieving algorithm also showing the final convex hull.

### 3.2 Detailed Sieve Algorithm

Using these major steps, a detail algorithm was developed and shown in Figure 6.

Algorithm SievePoints( Q ) returns SP

1. Let  $y$  be a lists of lists with magnitude defined by the Range( $Q$ )
2. Let  $x$  be a list of lists with magnitude defined by Domain( $Q$ )
3. For each point  $p$  in  $Q$  Do:
4.    $y[p.y].append(p)$
5. For each bucket  $b$  in  $y$  Do:
6.    $b = \text{sort\_bucket}(b)$
7.   If  $b.length > 0$  Then
8.      $p = b[0]$

9.      $x[p.x].append(p)$
10.    If  $b.length \geq 2$  Then
11.      $p = b[-1]$
12.      $x[p.x].append(p)$
13.    For each bucket in  $x$  Do:
14.     Else If  $b.length > 0$  Then
15.       $SP.append(b[0])$
16.     If  $b.length \geq 2$  Then
17.       $SP.append(b[-1])$

Figure 6: Detailed sieving algorithm.

The algorithm has input a set of points  $Q$  and returns a set of points  $SP$ . Steps 1 and 2 set up list of lists that are used temporarily store the points in buckets. These will be used to sort the points in the “y-direction” and “x-direction” respectively. Step 3 and 4 scan through the list of points and assign them to a bucket based on their y-coordinate. Then, in steps 5 and 6, each y-bucket is sorted using a stable sort such as bucket-sort. Step 6 identifies buckets with size greater than zero. In steps 7, 8, and 9, if a bucket size is found, then the first point is then reassigned to an x-bucket. In steps 10, 11, and 12, if the number points in each y-bucket is greater than or equal to 2, then the last point is also reassigned to an x-bucket. At this point in the algorithm all y-collinearities have been removed.

The second phase of the algorithm is similar to the first stage. In steps 13, 14, 15, and 16, each x-bucket is examined. If the number of points in an x-bucket is greater than 0, then the first point is copied to the sieve point list. If the number of points in an x-bucket is greater than 2, then the last point in the x-bucket is copied to the sieve points list.

### 3.3 Complexity of the Sieve Algorithm

In terms of space complexity, the sieving algorithm clearly uses  $O(n)$  space where  $n$  is the number of points in  $Q$ . In terms of time complexity, steps 3 and 4 can be computed in  $O(n)$  time. Each bucket is sorted using the `sort_bucket()` algorithm. This sorting algorithm uses bucket-sort that is a form of radix-sort. This is a stable-sort that can execute in  $O(n)$  time. Also noted is that the input points to the sort are only those that are y-collinear or x-collinear and only analyzed once per bucket. This reduces the time complexity significantly to  $\|y\| * O(\|y.b\|)$  where  $\|y\|$  is the number of buckets and  $\|y.b\|$  is the number of points in a given bucket. The value  $\|y\|$  is

a constant determined at runtime. Therefore, steps 5 through 12 run in  $O(n)$  time. Steps 13 through 17 run in  $O(n)$  time. Overall the runtime complexity of the sieving

## 4 Results and Analysis

This section will present the results and analysis of an implementation of the convex hull sieve algorithm when combined with an execution of the Graham Scan algorithm to compute the actual convex hull.

### 4.1 Test Suite

As a proof of concept, this convex hull sieve algorithm has been implemented in Python. Several sets of input points were used. The data sets were a uniformly distributed set of points that varied number the domain space of input points as well as the density of points in the domain. For example, for a domain space of  $1000 \times 1000$ , the total number possible points is 1 million points. However, the density would then reduce the number maximum number of points. With a density of 0.4, then a total of 400,000 points would be uniformly spaced in a domain of  $1000 \times 1000$ . The Python Timeit module was used to record the execution times.

### 4.2 Results for $10 \times 10$ Space

Figure 6 is a plot of computation time in seconds for different point densities. For this set the total number of unique points was 100. The algorithm was tested for the density range of .1 to .8. The series in orange displays the time it takes to sieve plus the time it takes to compute the convex hull using one of the conventional convex hull algorithms. From here, it is observed that at a density of 0.25 (25 points), the addition of a preprocessing step surpasses the performance of computing the convex hull of the entire set of points  $G$ .

### 4.3 Results for $1000 \times 1000$ Space

Table 1 shows sample data gathered for a  $1000 \times 1000$  space.  $Time(sieve)$  is the time it took to transform  $P \rightarrow SP$ .  $Time(CH(SP))$  is the time it took to compute the convex hull of the sieved points.  $Time(S+CH(SP))$  is the sum of the first two

columns.  $Time(CH(G))$  is the time it took to sort the same points using Graham Scan without the sieving step.  $G'$  is the number of points after the sieve.  $G$  is the original number of input points.

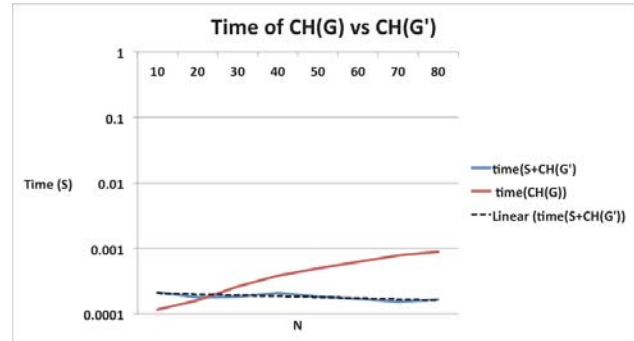


Figure 6: Results from a  $10 \times 10$  space with varying point densities.

| time(sieve) | time(CH(G')) | time(S+CH(G')) | time(CH(G)) | G'  | G      | Density of Points |
|-------------|--------------|----------------|-------------|-----|--------|-------------------|
| 0.227752066 | 0.001738024  | 0.229490089    | 2.581102037 | 181 | 100000 | 0.1               |
| 0.341944408 | 0.000799632  | 0.34274404     | 6.962977719 | 90  | 200000 | 0.2               |
| 0.435487461 | 0.000538683  | 0.436026144    | 13.42363429 | 66  | 300000 | 0.3               |
| 0.545644712 | 0.00041821   | 0.546062922    | 22.36479933 | 48  | 400000 | 0.4               |
| 0.652560496 | 0.000332856  | 0.652893353    | 34.00909503 | 39  | 500000 | 0.5               |
| 0.771003985 | 0.000257301  | 0.771261287    | 48.57069659 | 31  | 600000 | 0.6               |
| 0.870968318 | 0.000190163  | 0.871158481    | 66.29452085 | 23  | 700000 | 0.7               |
| 0.994344592 | 0.000150919  | 0.994495511    | 87.11052165 | 18  | 800000 | 0.8               |

Figure 7: Sample execution times for a  $1000 \times 1000$  space with varying densities.

The graph in Figure 7 is similar to Figure 6 above and shows the time it takes to compute the convex hull using Graham's Scan algorithm ( $O(n \lg n)$ ). This Figure also displays the time it takes to compute the convex hull using Graham's Scan with the addition of a preprocessing step. Each series was tested with the same set of unique points starting at 100,000 points and increasing in increments of 100,000 until the total reaches 800,000 (increasing densities from 0.1 to 0.8). The addition of a preprocessing step drastically reduces the amount of time needed to compute the convex hull. At 800,000 points the conventional algorithm took nearly 90 seconds to complete, while the algorithm in addition to the preprocessing step took under 1 second to compute the convex hull for the same set of points  $G$ .

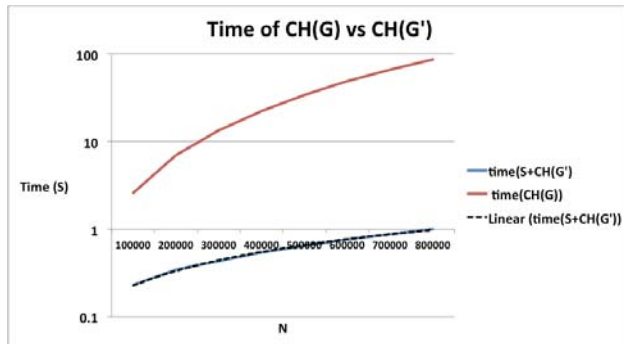


Figure 7: Results from a 1000 x 1000 space with varying point densities.

Figure 8 verifies empirically that the convex hull sieving algorithm is a linear function with respect to the number of points:  $O(n)$ .

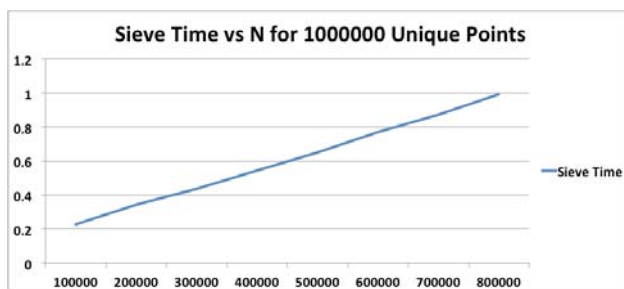


Figure 8: Algorithmic performance of the convex hull sieve algorithm.

Figure 9 is a plot showing the efficiency of adding the convex hull sieve algorithm. The efficiency of executing Graham Scan improves as the density of points increases. As the density increases, so does the likely hood that it will be removed using the sieving algorithm.

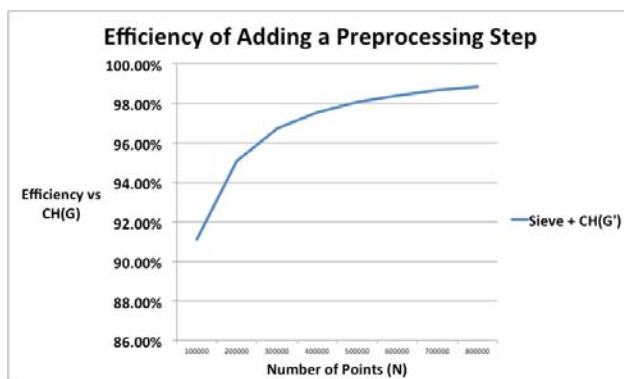


Figure 9: Efficiency of adding the preprocessing step.

## 5 Conclusions and Future Work

The research presented a technique to increase the performance of traditional convex hull algorithms. This is accomplished by a preprocessing sieving algorithm that filters the points prior the execution of a convex hull algorithm. The sieving algorithm repeatedly uses bucket-sort, which has complexity  $\Theta(n)$ , in the "x" and "y" directions, to remove interior collinear points.

For the case of a 10 x 10 space, it is shown that the benefits of using the convex hull sieving algorithm begins at roughly 25 points. For the case of 1000 x 1000, the performance improvement of using this convex hull sieving algorithm is significant. In the case of 800000 points the execution time of the sieve algorithm and Graham's Scan was approximately 1 second as compared to 90 seconds using Graham's Scan alone.

Future directions for this research are numerous. First, the test program is going to be converted from Python to C++ for more native and accurate timing analysis. This may lead to a more accurate timing model for this algorithm. Second, a variant of the algorithm is being developed for 3-dimensional convex hulls. This should be fairly straightforward extension, but will require significant testing and verification. Finally, the sieving algorithm has opportunities for parallelism. Perhaps using threads, the buckets could independently be populated from the original list of points. Using a portable thread library such as pthreads, would increase the portability of this algorithms to include hyper-threading processors and hardware accelerators.

## 6 References

- [1] Barber, C. Bradford, Dobkin, David P., Huhdanpaa, Hannu, "The quickhull algorithm for convex hulls", ACM Transactions on Mathematical Software, Vol. 22, No. 4, pp. 469-483.
- [2] Corwin, E., Logar, A., "Sorting in linear time – variations on the bucket sort", Journal of the Computing Sciences in Colleges, Vol. 20, No. 1, pp. 197-202, October 2004.

- [3] De Berg, Mark, van Kreveld, Marc, Overmars, Mark, and Schwarzkopf, Otfried, *Computational Geometry*, 2<sup>nd</sup> ed., Springer-Verlag, 2001.
- [4] Cormen, T. A., Leiserson, C. E., Rivest, R. L., and Stein, C., *Introduction to Algorithms*, 3<sup>rd</sup> ed. MIT Press, 2009.
- [5] Graham, R. L., "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set", *Information Processing Letters*, Vol. 1, pp. 132-133.
- [6] Horsley, Rev. Samuel, The Sieve of Eratosthenes, *Philosophical Transactions*, Vol. 62, 1772, pp. 327-347.
- [7] Jarvis, R. A., "On the identification of the convex hull of a finite set of points in the plane", *Information Processing Letters*, Vol. 2, pp. 18-21, 1973.
- [8] Laszlo, M. J., *Computational Geometry and Computer Graphics in C++*, Prentice Hall, 1996.
- [9] O'Rourke, J., *Computational Geometry in C*, 2<sup>nd</sup> ed. Cambridge University Press, 2005.
- [10] O'Neill, Melissa E., "The Genuine Sieve of Eratosthenes", *Journal of Functional Programming*, Cambridge University Press, Oct., 2008.
- [11] Preparata, Franco P., Hong, S. J., "Convex Hulls of Finite Sets of Points in Two and Three Dimensions", *CACM*, Vol. 20, No. 2, pp. 87-93, 1977.



# Applying Mobile and Pervasive Computing Security Projects in CS Courses

Yujian Fu<sup>1</sup>, Di Ma<sup>2</sup>

<sup>1</sup>Department of EE & CS, Alabama A&M University, Huntsville, AL, USA

<sup>2</sup>Department Computer Science, University Of Michigan at Dearborn, Dearborn, Mi, USA

**Abstract** - *In the information era, the Internet provides an unlimited platform and supports the mobile and pervasive computing in everyone's daily life, from oven, microwaver to the space craft. Today more and more users favorite the hand-held devices, mobile in a computer aided environment. Pervasive computing is in our life everyday. Security education in academic has been raised to an unanticipated level due to the proliferation of hand-held devices and applications. In this paper, we will presented an implementation of integrating mobile and pervasive computing security projects to computer science curriculum. This is a collaborative effort implemented by faculties at two national recognized institutions – Alabama A&M University and University of Michigan at Dearborn. The research study has involved several classes in undergraduate and graduate level with over 50 students participated, and demonstrated very exciting results in both pedagogical and scientific aspects among participated institutions. The developed projects with regarding to the curriculum design are presented and discussed.*

**Keywords:** Mobile security, pervasive computing, information assurance, cyber security, information security education

## 1 Introduction

In the information era, the Internet provides a unlimited platform and supports the mobile and pervasive computing in everyone's daily life, from oven, microwaver to the space craft. Today more and more users favourites the hand-held devices, mobile in a computer aided environment. Pervasive computing is in our life everyday. Security education in academic has been raised to an unanticipated level due to the proliferation of hand-held devices and applications. According to [1], malware attacks to hand-held devices has increased by 25 percent across all platforms since 2012. The McAfee Labs count of new suspect URLs set a three-month record with more than 18 million, a 19% increase over Q4 and the fourth straight quarterly increase. Among these attacks of various platforms, Android is the platform favored by the most malware with most growth. 67.7% of host locations in North America. From mobile report of F-Secure [2], 91% new families or threats was identified, and among these new

families of malware on mobile device 99% on Android platform. To reduce and mitigate the security threats and increase the defence capabilities of benign apps, traditional Intrusion Detection Systems (IDSs) has been shown inefficient. New behaviour-based IDSs are paid attention to by many mobile security researchers [3, 4]. In addition, a single or individual approach without the support of formal specification has been proved with more false positives in the results. The needs of formal specifications to provide the sound theories are more and more desired.

From the side of the educational point of view, the needs of the security issues of mobile and pervasive computing fall in the requirements of stakeholders, expectations of users at industry and academic. The needs of cyber security job market has increased 3.5 times percent since 2012 among all IT and information jobs according to CIO report from The Journal of Wall Street [5]. In addition, salary of cyber security position is 17% more than average of IT positions. To teach and train the next generation of cyber security workforce is kind of first need of current education in computer and STEM curriculum. This is one of the ultimate goal of this NSF supported project which is aligned with one of objectives of NSF SaTC program.

Two objectives of this NSF project are: First, enhancing the information security education through curriculum development. Second, enhancing students in the IA security through hands-on lab development. Before we illustrate the implementation towards these objectives, we will first introduce the current status of cyber security education in the institutions. After that, the course and lab development will be discussed in the next section.

### 1.1 Current Cyber Security Education at AAMU

The computer science department at Alabama A&M University is one of the oldest department of Alabama state since 1960s'. The information security and forensics curriculum of computer science was included in 2009 to fulfill ABET requirements and satisfy the evaluation criteria. Both courses does not have programming language courses as prerequisite courses, students with any level can register. This attracts many students but increases the concerns of the learning results and teaching quality in the content of information security. In addition, due to short of faculties in

the area, our students cannot be exposed enough security in theories and short of hands-on labs.

In 2011, driven by the needs of marketing, supported by Deans office, computer science announced a cyber security concentration in undergraduate student curriculum. This cyber security concentration includes six courses: CMP 381 Computer Organization, CMP 384 Operating Systems, CMP 386 Cryptography, CMP 321 Principle of Information Security, CMP 414 Forensic Computing, CMP 421 Computer Security. Except for the core courses of CMP 381 and CMP 384, the other four courses offered once per year, and the student number registered to other four courses per semester is around 14 to 25. The peak of registration was reached by 2012. By the 2013 fall, the registration started to drop. These four courses are taught by 3 faculties and one adjunct professor. There is no pervasive computing and mobile courses offered in computer science curriculum.

There are several reasons of the student number dropped in these security courses. One of the reasons is lack of the interesting and research related, hands-on lab. Some of the new research findings and security detections are not included and exposed, nor well designed lab, short of security theory and design analysis in the lectures. Second reason, there is no new topics to feed current millimium era students. This generation of students are grown up with electronics and internet. Information and electronics are their whole life. Traditional teaching and content are far from enough to satisfy our current students. Last reason is due to the limited number of students, there are not enough students to register these courses. There is an obvious need of the security in the pervasive and mobile computing from current computer science curriculum.

The paper is organized as follows. In the next section, we introduce the course design and curriculum development supported by this NSF program. In Section 3, the developed labs regarding to the mobile and pervasive computing are presented. Section 4 discusses the current evaluation by the survey in AAMU campus. In Section 5, a short discuss of pedagogical results will be described. Finally, we conclude our work in section 6.

## 2 Applied Course and Design

In this section, we will present the courses that are integrated with the security contents and projects. Furthermore, a new course developed in UMD was presented here.

Several courses that were updated with security and integrated with security projects in both AAMU and UMD campus. In AAMU campus, we have simply attempted security projects in software engineering (CS 401) and senior design classes (CS 403).

### 2.1 Software Engineering Course (CS 401)

CS 401 is software engineering class which requires senior standing. This course is designed to explore the traditional approach to software development & construction life cycle,

software crisis, and software characteristics. In the course description, it is to cover various software engineering paradigms, and the fundamental concepts of analysis, design, coding, testing and maintenance [6]. Besides, this course introduces various CASE tools that support these methodologies.

Three student learning outcomes are covered from the current syllabus:

- a) Understand the traditional approach of software development process, software characteristics, software quality, ethics issues, crisis issues and software development cost and management.
- b) Understand and be able to use basic software engineering methodologies to solve large scale software/software-intensive system development. Understand and be knowledgeable about existing tools of some software engineering methodologies. *Understand and be able to design and implement cyber physical systems with critical concerns including security aspects.*
- c) Be knowledgeable of software testing strategies and be able to use basic software testing technologies to validate program.

Security issues have brought a lot of attention of system analysts and software engineering. The earlier to detect and identify errors and faults, the more to reduce the cost of failure. It remains challenge to identify vulnerabilities in the software design models of systems and applications. To help students to understand how design level can help reduce the system crash due to malicious attack, we have updated the courses with two aspects – in lecture and in the hands-on projects, as follows:

In lecture, UML sec was introduced to develop the model of the secure system. In addition, to identify the security properties of the applications, OCL (Object Constraints Language) was introduced to the class in three levels – the syntax, the semantics and practice questions. We have introduced complete set of OCL syntax by combining OCL security specification with the UML diagram. To help student to have a better understanding, we introduce segment of systems based on the context and domains. The hands-on project selected is testing of SMS message passing on Android. This hands-on project will be introduced next section.

### 2.2 Updated Senior Design Course (CS 403)

The senior design course is a core course for undergraduate students at AAMU that requires CS 401. The course aims at exposing students to various types of systems and development processes. Development and successful completion of a sponsored software development project. Specific objectives include the development of effective project management, communication, and technical skills, experience with the implementation and testing phases of a realistic product design cycle, and an ordered transition from a

classroom-oriented academic environment to a performance-oriented professional environment. Student learning outcomes:

1. Improve the ability to apply knowledge of computing, mathematics, science and engineering.
2. Improve the ability to design, implement and validation a computer-based system, process, component, or program to meet desired needs.
3. Enhance the ability to analyze a problem, and identify, formulate and use the appropriate computing and engineering requirements for obtaining its solution.
4. To be understanding an understanding of professional, ethical, legal, security and social issues and responsibilities.
5. Prompt the ability to use current techniques, skills, and tools necessary for computing and engineering practice.

In the project designed in this course, we particularly increase the project pool with a couple of Android and security projects – including SMS message passing, testing on the cryptography algorithm in SMS app, Android app of AAMU faculty info. Students have shown great interest in the android apps and security projects. In the data collected, it shows that one group worked on Android app, and one group worked on the security testing of SMS app.

In addition to the above two courses, we have designed and implemented other mobile security projects and used in the Object oriented design (CS521), and Software Engineering Methodology (CS561). The detailed project description will be presented in Section 3.

### 2.3 New Course Development at UMD

A new course of pervasive computing and mobile computing was developed by UMD last year. The course aims at integrating the latest research results in pervasive computing and mobile security to the current CS curriculum. In addition, this course is designed to a thorough analysis of the major trends in pervasive and mobile computing and explain the implications in terms of security and privacy. For every security and privacy issue, we will give a detailed description of the problem and a precise explanation of mainstream solutions wherever they exist, and of potential solutions otherwise. Tentatively, a total of nine topics were developed. A brief description of each topic is given as follows.

Topic 1: Introduction to Pervasive & Mobile Computing. This topic covers the wireless communication and security & privacy risks.

Topic 2: Wi-Fi LAN and cellular network security. This topic introduces network access security requirements, wifi security, cellular network security.

Topic 3: RFID security and privacy. This topic exposes students with RFID technology and applications, security and privacy threats, defenses mechanism, and NFC & mobile payment systems.

Topic 4: Smartphone security. This topic brings the current research study in smart phone systems, security models,

attacks on android permission, smartphone malware detection, and BYOD security.

Topic 5: Vehicle and vehicular Ad-Hoc (VANET) security and privacy. The cutting-edge research studies on the intelligent transportation systems, in-vehicle data & communication systems and vulnerabilities, IEEE 802.11p for wireless access in vehicular environments and IEEE 1609.2 for VANET security are discussed.

Topic 6: Secure device pairing. This topic includes Bluetooth-enabled device pairing and other device pairing mechanisms through SMS, directory service, multiple antenna.

Topic 7: Secure ranging. This topic covers the relay attack, relay attack defense.

Topic 8: Secure neighbor discovery. The wormhole attack, centralized and decentralized approach for wormhole detection will be discussed.

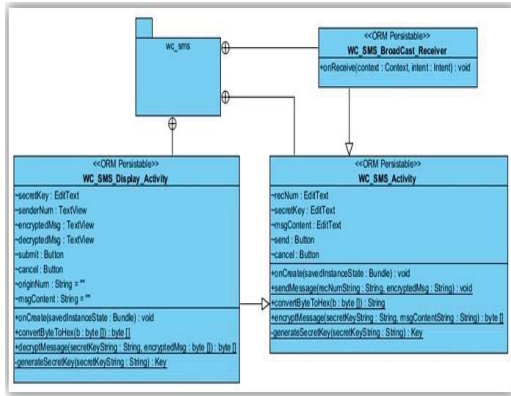
Topic 9: Secure localization and location privacy. This topic focuses on the device localization and vulnerability, secure localization based on own measurements, and location privacy in VANET.

## 3 Project and Lab Design of Mobile Security and Pervasive Computing Design

In this section we will present the projects developed for mobile and pervasive computing project that were used to integrate with the updated courses and lab development. These projects are developed by two institutions in the past project year.

### 3.1 Testing – SMS Encryption Algorithm

SMS messaging [7] is a mobile and stronger version of “any time” and “any where” service. A switched-on mobile device is able to receive or send a message regardless of if a voice or data call is in progress. To secure the private data and ensure the correctness of the system implementation, this project is developed in three phases: I) develop a SMS app that is able to pass simple message with AES. Through this phase, student will be exposed to fundamental skills of Android apps, the simple AES algorithm (Fig.1. (b)). II) Develop a class diagram of the SMS message passing (Fig. 1 (a)). In addition, define the authentication security properties in OCL on the class model. Through this phase, students will be able to understand the software design methodology, security properties in a simple format (UML diagram). III) Install JUnit and run assertions on the OCL properties to demonstrate if the cryptographic algorithm implemented correctly by a set of properly designed security properties. By doing this, students will have a better understand of the system quality, crisis, how to ensure the correct implementation, and crash cost bring by the vulnerability.



(a)

Figure 1. The design and snapshot of Android security testing



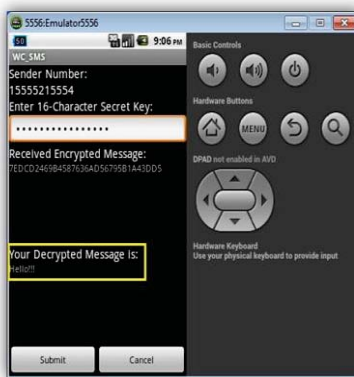
Figure 2. Snapshot of WiFi Sniffing – AAMU WiFi

### 3.2 WiFi Sniffing on Android

Traffic monitoring is one of the key approaches in the network security to detect the potential vulnerabilities and/or attacks. Rooted on various types of methodologies, many tools are developed regarding to the monitoring aspects and flow strategy. One of the most powerful spying tools is Interceptor-NG [8]. It is a free application with unrestricted functionality and is virtually universal: works on Windows, Linux, Mac OSX, iPhone and Android. It is a multifunctional network toolkit for various types of IT specialists [9]. It has functionality of several famous separate tools and more over offers a good unique alternative of Wireshark for Android. After connected to the AAMU WiFi using Interceptor-NG, we can run the scan command to see all the devices with IP addresses that are connecting to AAMU WiFi. Result is shown in Fig. 2, where a list of AAMU WiFi address was recognized and displayed to the screen.

### 3.3 Permission ID based Security Analysis

The Android OS system runs each application under the privileges of different “user”. A unique user ID to each of them is assigned when the application request is coming. Applications are required to declare in a manifest that can take place in the course of execution [10]. This project is designed to expose the different security levels of permission ID in the Android systems, and if sensitive permission ID is required. It is straightforward to predicate that an application overprivileged needs to be suspected (Fig. 3).



(b)

Figure 1. The design and snapshot of Android security testing

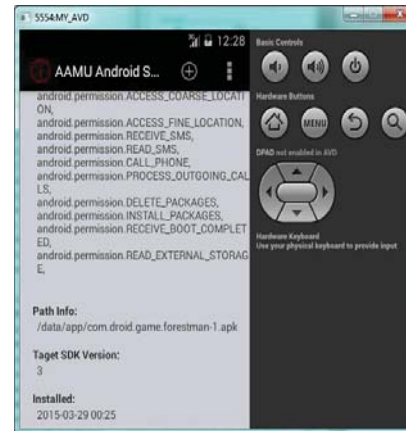


Figure 3. Snapshot of permission ID based security analysis

### 3.4 Man-in-the-middle Attack Exploiting Certificate Verification Flaws In Smartphone Apps

On the smartphone app market, apps are developed by developers with various level of security knowledge and many of them are suspected to be flawed in certificate validation. In this project, a student is expected to conduct a serial of experiments to find flawed apps and further analyze the cause.

### 3.5 WebView Information Stealing

Web applications relies on several TCB (trustworthy computing base) components to achieve security. One important TCB component is the use of trustworthy browsers. However, WebView (embedded browser for mobile apps) changes the picture of TCB for web security. The lacking of security support for cross-application interface embedding in mobile platforms allows a malicious host to eavesdrop on input intended for embedded interface only. In this lab, students are required to mount a password-grabbing attack by using JavaScript injection.

- d) How much important do you evaluate security regarding to a quality software?
- e) Do you think system design place a key role on the software quality?
- f) How much do you know about software testing?

For classes in Spring 2014, it was interviewed at 11 students in CS 401 and 15 CS 561. A statistic results regarding to questions are shown in Fig 4. It is clearly shown from both classes that many students do not consider the security and cyber attacks play a critical role in the reliability of the system development. After this class, the students had changed their mind and the collected data indicated that most considered the



Figure 4. Data collection for courses cs 401 and cs 561.

## 4 Evaluation and Discussion

Class evaluation is done regular each semester. Other the regular class evaluation, for CS 401 and 403 at AAMU, the instructor has developed pre and post set of questions to evaluate these courses.

The pre/post test question for CS 401 is listed as following. The answer is given from 1(no knowledge) to 10 (very much know).

- a) How much do you know about Software Engineering?
- b) How do you consider System quality is critical during software development process?
- c) Do you consider cyber attacks and security holes in the system as one of the key factors that cause software system crash and cost a lot?

security is one of the key concerns of the reliable system design. For CS 561, we have shown the value of each answers pre- and post test, in addition to the line chart. Even some graduate students have shown the knowledge of reliability regarding to security in the system design at the software engineering aspect, there is still an increase of the value regarding to these questions.

We also conducted the pre and post tests of CS 403 and CS 521 (object oriented design and implementation). Since these two courses are less in common, the question sets are a little bit different. Due to the space, the pre/post test questions for CS 403 and CS 521 are not listed.

## 5 Pedagogical Issues

We summarize pedagogical significance regarding to the project – motivation. Motivation is a topic that remains to be challenge to all education researchers how to motivate students in different areas. Project based learning (PBL) is one of the traditional technology that has been proved effective in the student motivation, provided that the topics are very interesting and the ideas are novel, the skills is simple and the knowledge are not too much. On the contrary, students will be easily to loss interest and PBL will not make sense for the purpose of improve student learning outcomes. Regarding to these concerns, and our projects are designed in the large extent to maintain constant student interests on the point of this final results on the Android security is rooted in daily life and the topic is attractive.

## 6 Conclusions and Future Works

In this paper, we presented a study of integrating pervasive and mobile computing to CS curriculum at AAMU and UMD in the past year. Several updated courses, one new developed course, and hands on lab were discussed. Student evaluation was presented. This work demonstrated that properly integrating new cutting edge research projects to CS curriculum can motivate students in pursuing higher degree or continue on the computer science study even if the CS market right now is not taking the lead position of all jobs. Students always love to see new research results and are exciting about the using and applying the ideas to a not complicated project. In the future, on top of current result, we expect to i) to develop more interesting projects that are rooted from current research study and is able to fit for CS or STEM curriculum; ii) conduct more study especially tracking students in the high level grade and/or graduate study.

## Acknowledgements

The research work was supported by National Science Foundation of USA under Grant No. 1419295 and Grant No. 1419280.

## 7 References

- [1] MacFee Report. Mobile Malware Report. June 2014.
- [2] F-Secure Report. Mobile Malware Report. June 2014.
- [3] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. “Semantics-Aware Android Malware Classification using Weighted Contextual API Dependency Graphs”, In Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS'14), November 2014.
- [4] Mu Zhang and Heng Yin, “AppSealer: Automatic generation of vulnerability-specific patches for preventing component hijacking attacks in Android applications”, In

- Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'14), February 2014.
- [5] Steve Rosenbush. Demand for Cyber Security Jobs is Soaring. March 2013. CIO Journal. The Wall Street Journal. Retrieved From: <http://blogs.wsj.com/cio/2013/03/04/demand-for-cyber-security-jobs-is-soaring/>
- [6] Ye Wu, Mei-Hwa Chen, and Jeff Offutt. “UML-based Integration Testing for Component-based Software”. *Proceedings of the Second International Conference on COTS-Based Software Systems*, Lecture Notes In Computer Science; Vol. 2580, pages: 251 – 260, 2003.
- [7] Sagheer, A.M.; Abdulhameed, A.A.; AbdulJabbar, M.A., "SMS Security for Smartphone," In proceedings of *The Sixth International Conference on Developments in eSystems Engineering (DeSE), 2013*, pp.281-285, 16-18 Dec. 2013
- [8] Interceptor-NG. Available from: <http://interceptor.nerf.ru/>
- [9] XDA Developers Forum. Available from: <http://forum.xda-developers.com/showthread.php?p=35159281>
- [10]Z. Aung, and W. Zaw. “Permission-Based Android Malware Detection.” *International Journal of Scientific and Technology Research*, Vol. 2, Issue 3, March 2013.

# Developing of virtual tools to improve classroom performance in physics

Rosete F. Juan C.<sup>1</sup>, Guzmán R. Miguel A.<sup>2</sup>, Estrada R. Felipe<sup>3</sup>

1 Mechanical Engineering Department, Technological Institute of Queretaro, Av. Tecnológico S/N Esq. M. Escobedo, PC 76000, Querétaro, Qro, México  
jc\_rosete@yahoo.com.mx

2 Computer Engineering Department, Technological Institute of Queretaro

3 Computer Engineering Department, Technological Institute of Queretaro

**Abstract.** *-This paper exposes the results of a serial of researches focused to identify the main reasons of a low student performance in physics and classical Newton's mechanics.*

*This research first exposes the percentage of students that have failed to complete a satisfactory lecture of physics and the possible relationship between this students' behavior and classroom conditions in progressive programs of study as mechanics of materials, statics and dynamics from the last 6 years to today.*

*Finally, considering the misconceptions about this topics as one of the potential reasons of the difficulties by the students to link the theoretical concepts with elements of the reality, the paper exposes the actions implemented in order to provide to the students a serial of virtual tools that could help them to establish solid connections and mental structures between the bodies behavior and the lectures.*

**Keywords:** Misconceptions, physics, models.

## 1 Introduction

Desertion and low notes are typical problems that several countries around the world experiment continuously in their day by day educative process. These conditions are normally evaluated as a way to identify the economical amount of resources that every year the country loses for this concept and are the basis to plan strategies oriented to improve the activity into and out of the classroom.

In Mexico, this situation is an important point of interest from the high education authorities, this interest obeys to the obtained results in the last analysis made by the OECD and where Mexico got a low performance note.

Several government strategies have been developed to stop the condition of low performance into the classroom and low understanding of basic knowledge. Some of the actions and policies followed into the Universities include continuous monitoring of the class desertion, quality models developed that could assure a number of minimal requirements covered into the classroom, student – student teaching and the empowering of virtual tools that could be applied by the students as distance teaching and virtual lectures.

The Technological Institute of Queretaro is immersed into this national strategy for higher education in order to identify the main reasons of school desertion and low profile of an important number of new engineers.

The first activity deployed for the Mechanical Engineering Department was to establish the lectures that have the largest number of students with low notes and the level of impact with the progressive lectures. From this analysis it was possible to conclude that Physics has been the study program with the largest number of students of mechanical engineering, followed by Statics and Dynamics lectures.

Given that this 3 programs represent the foundations into mechanical engineering by the programs of mechanics of materials, mechanical design and mechanical vibrations, it is a priority

of the department to understand the reasons and to design assertive improvement programs

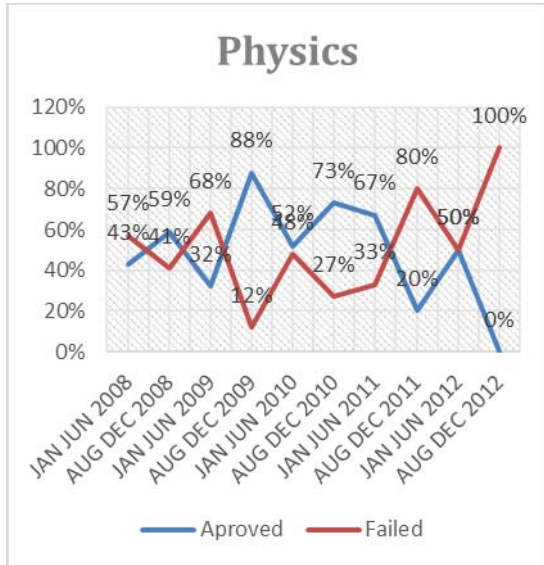


Figure 1 Physics lecture behavior in Mechanical Engineering

One the remarkable situations from the study is the up & down behavior of the notes obtained for the students where one semester has high level of low notes and the next shows a better behavior only followed for another low level semester, the 2 2012 semester is significant cause the 100% of the students failed in the lecture.

Statics and dynamics lectures exhibit a close behavior as the next graphs denote.

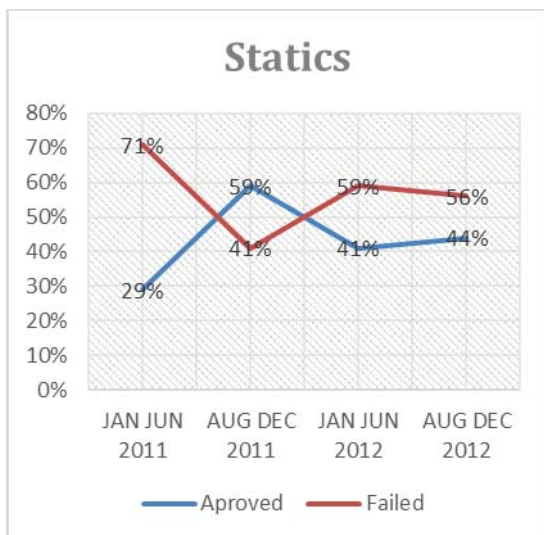


Figure 2 Statics lecture behavior in Mechanical Engineering

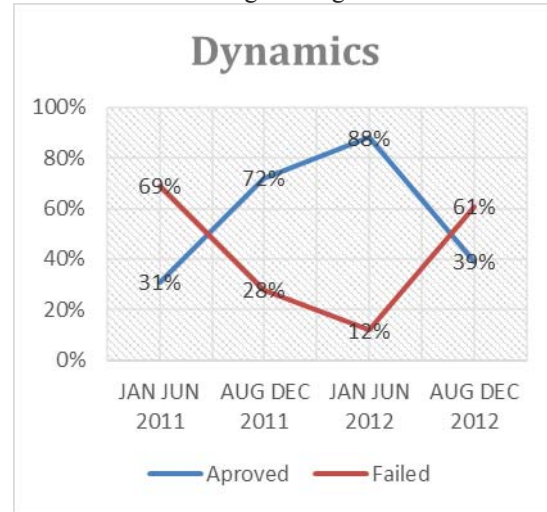


Figure 3 Dynamics lecture behavior in Mechanical Engineering

It is not clear at this moment the relationship between the low notes in physics and the notes obtained in statics or dynamics, one condition that does difficult to identify a main reason is the fact that physics is not an exclusive lecture of mechanical engineering and in a classroom it is possible to find students from all the programs of engineering and some teachers are not titular to these lectures.

Desertion from the Institute is another problem that can be involved with the physics problem. It is really that this lecture does not represent the only reason of desertion but certainly it can be an important factor to consider if we desire to improve the number of graduates into the institute, but more important of that is the fact that the engineering departments must be working in the academic level improvement if we desire competitive engineers.

The next graphs show the percentage of students that have interrupted their studies after 5 or seven semesters. Unfortunately at this point it has not been possible to identify the total reasons that are involved in the decision as low academic level, illness, a pregnant condition or insufficient incomes to pay the studies.



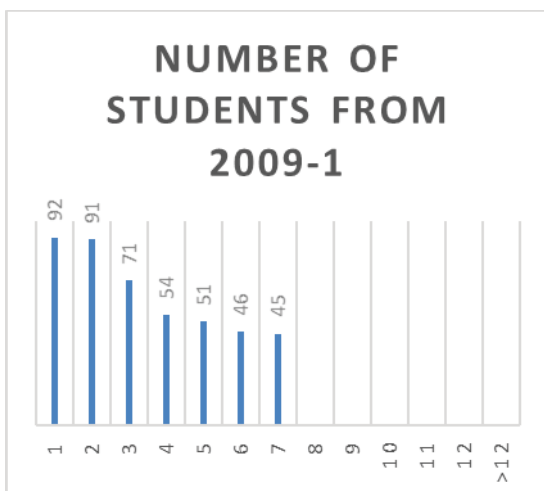


Figure 4 Number of remaining students in 7<sup>th</sup> semester from a total of 92 in the first semester

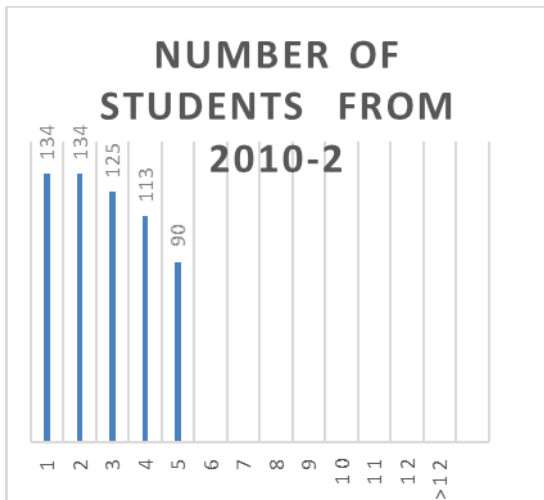


Figure 5 Number of remaining students in 5<sup>th</sup> semester from a total of 134 in the first semester

When the topic of study is physics, a large number of students, in their own opinion, think this basic science is easier than math and they consider themselves with a higher dominion of the topics related to the discipline. However, in spite of this point of view from the students, the rate of failing is close to the 50% in all engineering programs [2].

This condition is the result of several factors acting simultaneously, including large number of lectures per day, the cultural influence of mass communication media, a lack of criteria to discern the quality of information found in the internet, and lots of misconceptions related with basic Newton's law amongst others factors. All of these

are typical reasons for a student to fail a course, and they are the cause that the students aren't able to establish logical connections between the theory learned at school and their experiences in real life.

In the particular case of physics lectures, there are three elements that are common: a low level in math, a low development of abilities to establish connections between theoretical concepts and real life situations [3] and misconceptions about a topic that acts as an inhibitor of the learning process [4].

Considering the necessity of a long time to clarify the reasons linked with a low level in math and the total factors to produce effective mental connection between the real life and the theoretical program of math and physics, it was decided to evaluate the level of influence of misconceptions with the number of poor notes in classroom, starting a research program with students of mechanical engineering and computing engineering, both programs with different background in the topic and different future applications.

In order to identify what kind of wrong previous ideas our students of engineering have, it was applied the "Force Concept Inventory" (FCI) [5], a multiple choice test developed in 1992 whose main objective is to evaluate the level of comprehension of a student about the concepts of movement and force in physics. Most of those who answered the test failed, and they are not even aware of it.

## 2 Misconceptions detection

### 2.1 Structure of the test

The first step was to apply a serial of tests to students of computer and mechanical engineering with 10 questions selected from the "Force Concept Inventory", about the bodies' movement understanding.

The test is designed to explain a possible situation of a body under movement and the student must think about the logical answer that could join the situation with the information and experiences that he has previously obtained.

Every one of the questions contain the description of one of such situations, most of the times a picture that shows several options to describe the behavior of the body and a total of 5 options to choose one of them as the right answer. In every case the student is told to disregard the effects of air friction for the bodies in movement.

The next example took from the FCI test can explain the nature of the questions and the level of misunderstanding about the topic in physics.

Example:

“A canon ball is launched at 0 degrees. What option can describe best the movement of the ball after it is shot out of the canon?”

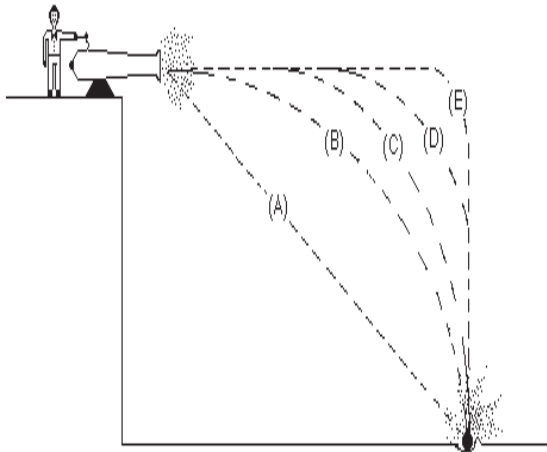


Figure 6 Question example

With the next results

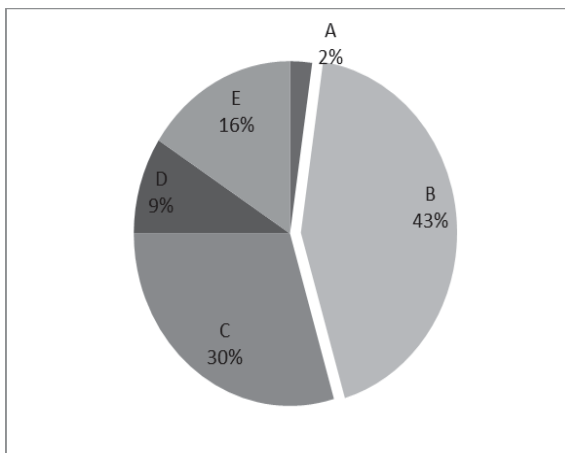


Figure 7 Question number 4 selections

The graphic shows the number of students (43 %) that chose the correct option (B), it corresponds to one parabolic movement given that the combination of movement vectors along X axis with constant velocity and the vector along Y axis with accelerated conditions because the action of the gravity.

It is important to check the high number of students that chose option E nonetheless the impossibility of such movement trajectory under the stated conditions. This number of responses can suggest that there are several factors involved that led the students to consider some impossible trajectories as valid possibilities, in spite of the knowledge acquired at school.

### 2.2 Validating the proposal

At this point it was necessary to validate if the previous ideas can influence the cognitive procedures of one individual to consider possible movement conditions that the serial of experiences in our life have shown that are not valid.

Another question from the FCI requests the movement behavior of a body that is dropped from an airplane in flight at a constant speed from the perspective of a standing observer. The 5 options show different combination of movement but only option D is correct.

The next 2 figures define the quest conditions and the percentage of students that selected it. The results show several difficulties for the individuals to understand the behavior of bodies under this kind of movement.

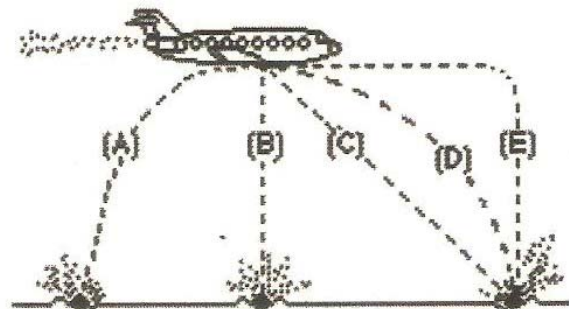


Figure 8. Dropped body movement

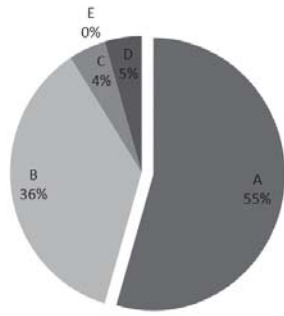


Figure 9 Plane question and results with students of higher levels before taking the physics course

With this information it is obvious that a large number of students have trouble to establish proper connections with a proposed condition and the physics laws studied in the classroom, this sentence is formulated because the high number of answers A and B and a really low percentage answered the right option D. Maybe the influence of mass media as TV cartoons or movies could be directly responsible.

### 2.2.1 Knowledge in physics

Another hypothesis formulated to define the potential reason of this kind of mistakes is that most of the students in second semester have not matriculated the course of physics and it can be one reason of the obtained results.

Again, the same test was applied to different groups of students of computing engineering but now with the difference that they had completed and covered the conditions to validate the lecture. This condition should have improved the number of correct answers in the same test, because of the serial of experiences and academic resources applied into and out of the classroom.

The next graph shows the result of the plane question with students that had approved the physics course:

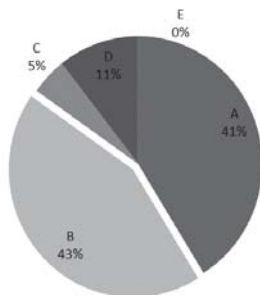


Figure 10 Plane question and results with students of higher levels after taking the physics course

This means that the impact of the lectures in the understanding of physics laws is poor, because only 11% of students concluded that D was the right option in comparison with the original 5%, and the number of students that selected option B increased in 7%.

The final results from the test let us to conclude that the information provided in a typical program of physics is not enough to change the understanding level of the topic and it is a certainty that this condition will affect the performance of the student in consecutive courses given that serial of mistakes that the individual is carrying on.

In order to identify the more important reasons of this condition a secondary research was done. It was found that most of the students that were wrong in the first study have a great influence of the concepts and situations from the mass media such as TV, video games and movies.

### 2.2.2 Applying software and hardware to develop experiments

In order to identify the most common errors in physics a virtual platform was developed; this time, the totality of the questions from the FCI were included as multiple choice test, registering in addition the engineering program that the participant was studying and whether the student had accredited the physics course.

Some questions are only sentences about physics concepts, other as it illustrates in figure 13 include small animations made in flash to provide more information about the situation that must be studied. A timer of 30 minutes is included to identify the clarity of the questionnaire and the capabilities of the students.

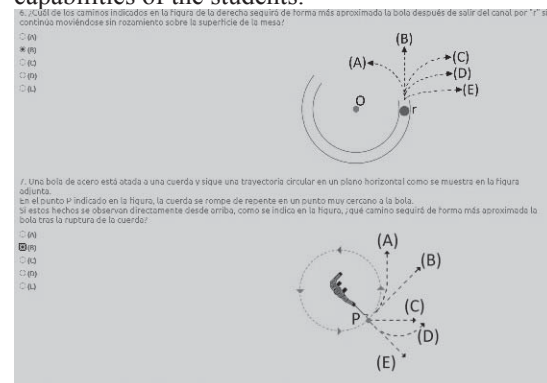


Figure 11 Multiple option questions with visual help

Just when the student concludes the FCI's questions, a simple graph is displayed to notify the percentage of properly answered questions and mistakes done.

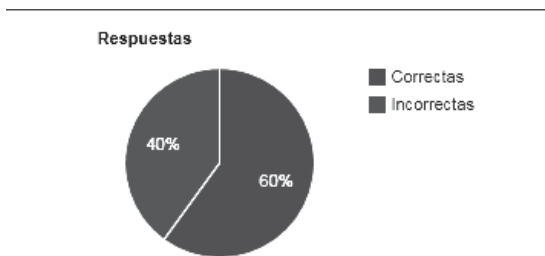


Figure 12 Graph displayed to the student after the test notifying the percentage of mistakes

The teacher has access to the total mistakes of the group deploying the global graphs of every question as an option to modify the classroom actions or if the questionnaire is applied before the lecture, it could be an interesting reference about the potential performance of the group, and help the teacher to design better didactic strategies

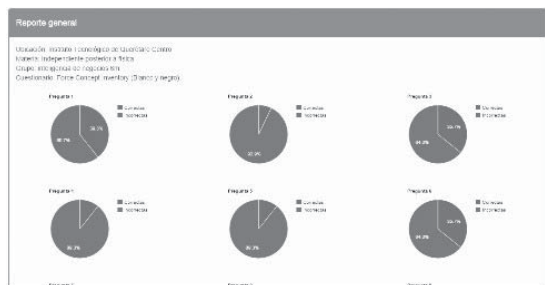


Figure 13 Teacher back information about group's performance

The test will be applied to the students that belong to the main campus and the students that are established in the counties and rural areas in a distance education engineering program. The goal is to complete a statistically significant enough number of tests to validate our hypothesis about the most common misconceptions amongst students and at the same time to provide a powerful tool for the teachers of physics to do a diagnosis of the group before to start activities in the classroom.

### 2.2.3 Obtained results

The obtained results with the tool were totally satisfactory at this moment. All the groups of students were involved in the solution of an engineering problem that required the development of a series of abilities and competences, not only in physics. Because all people involved in the test could notice about the number of mistakes it had the opportunity to think about the potential misunderstanding of the concepts about the movement and forces between bodies.

## 3 Conclusions

There is an abundance of digital resources that allow the students to get information about all kinds of topics; unfortunately, more information doesn't mean more understanding of the topics from the students [6].

It is important for the teachers in the classroom to adapt to the use of the new technologies in the learning process, looking for a way to improve the performance of the students, but at the same time, the teachers must be interested about the particular situation of a group to deploy a series of strategies to improve the performance in the topic.

Both virtual and real life experiments are being designed to understand the best way to teach physics and tending conceptual bridges to theoretical information as seen in the classroom. Our main objective, for the moment, is to get sound information about the most prevalent misconceptions in classical mechanics and to provide educational tools to eliminate them, improving the comprehension of this important topic for engineering.

## 4 References

1. OECD (2014). Education at glance: OECD Indicators. Retrieved from <http://www.oecd.org/edu/Mexico-EAG2014-Country-Note.pdf>
2. Jimenez G. M. A. (2011) Rate of non satisfactory results into a physics course. *Crónica Naranja*, 6 (25), pp 41
3. Stahl, G., Koschmann, T., Suthers D, (2006) Computer – supported collaborative learning: An historical

- perspective. Cambridge handbook of the learning sciences, pp 409 – 426
4. Kind, V. (2004) Beyond appearances: Students' misconceptions about basic ideas, 2<sup>nd</sup> edition.
  5. Hestens, D., Wells, M., Swackhammer, G. (1992), Force Concept Inventory. The physics teacher, 30, pp. 141-158
  6. Hay D., Kinchin, (2008) Making learning visible: the role of concept mapping in higher education, Studies in higher education, 33 (3), pp. 295 – 311

# A Short Survey on Statistical Network Analysis

Matthias Dehmer

Universität der Bundeswehr München  
Department of Computer Science  
Werner-Heisenberg-Weg 39  
85577 Neubiberg, Germany  
matthias.dehmer@unibw.de

Stefan Pickl

Universität der Bundeswehr München  
Department of Computer Science  
Werner-Heisenberg-Weg 39  
85577 Neubiberg, Germany  
stefan.pickl@unibw.de

Zhonglin Wang

Universität der Bundeswehr München  
Department of Computer Science  
Werner-Heisenberg-Weg 39  
85577 Neubiberg, Germany  
zhonglin.wang@unibw.de

**Abstract**—In this paper, we perform a mini review on statistical network analysis. We survey recent papers in this field and highlight those that are based on using statistical methods when analyzing structural properties of complex networks.

## I. INTRODUCTION

When analyzing complex networks, statistical techniques such as resampling [11], bootstrapping [28], randomization [31] have been proven useful. For instance, typical network structures represent technical networks, biological networks, and social networks. Technical networks appear for example in communication technology, transportation, and energy. In computational biology, several biological networks such as gene networks have been investigated. Emmert-Streib et al. [12], [6] found that biological networks are often non-deterministic, that means they cannot be inferred deterministically. Therefore statistical methods to analyze such networks have been crucial, see [12], [6].

In complex network analysis, there are several problems to deal with. For example, it has been challenging to collect network data properly such that statistical methods become applicable. Another problem that has been often demanding relates to visualize and analyze networks meaningfully [13]. To analyze networks, structural graph measures have been often used, see, e.g., see [12], [6]. Examples are the clustering coefficient, cohesion measures, connectivity measures and other topological indices to map graphs to real numbers [8]. The latter method relates to determine the complexity of networks. By employing statistical methods, important applications in network theory such as subgraph sampling and link mining. Link mining techniques have been also applied in Web Structure Mining [5] and related disciplines. Another striking problem in statistical network analysis is network inference [3].

In the age of big data, analyzing massive data sets become more and more important. In case of structural data (networks), tackling the complexity of the data sets has been often challenging. Either the networks are very large or one needs to deal with a huge number of graphs, see [7]. In both cases, applying statistical methods has been fruitful. The main contribution of the paper is to survey recent work on statistical network analysis. The survey aims to highlight the interdisciplinary character of the field and, hence, the paper can be useful

for those who want to tackle problems in statistical network analysis and related disciplines.

## II. REVIEW

In the following we start surveying recent contributions. We begin with a paper due to Eldardiry and Neville [11] dealing with network sampling. Eldardiry and Neville [11] proposed a novel subgraph resampling approach which cannot only generate pseudo samples with sufficient global variance. The method also maintains local relational dependencies and link structures. The algorithm is based on a two-phase relational subgraph resampling technique. The first phase selects subgraphs of same size from the original relational data, and the second phase links up the selected subgraphs. Finally the authors applied two different relational settings to evaluate the resampling method.

In [9], Dehmer and Basak explored several statistical methods to analyze complex networks. Also, the book deals with explaining machine learning methods for networks such as graph classification. Importantly [9] shows that graph theory, machine learning and statistical techniques have been applied in an interdisciplinary manner.

It is known that bootstrapping is a well-known data-driven approach used to create random pseudo samples with just one empirical observation. In this context Tremblay et al. [28] presented an approach for statistical resampling based on bootstrapping of nodes under constraints. The whole network was used to do the analysis and the aim of their study was to design a statistical test and find acceptance intervals for various null hypotheses concerning relevant observable features of groups of nodes in a given network. They demonstrated the performance of the network by using real data sets.

Detecting communities in graphs has been useful to identify functional sub units of a system and to reveal the similarities among vertices. In [15], Fortunato explored the problem of community detection by defining the problem and discussing issue regarding the significance of the method. This method is mainly based on bayesian inference where the best fit is obtained through the maximization of likelihood (generative models). The observations are used by using Bayesian inference to estimate the probability to verify whether the given hypothesis is true or false.

Djidjev et al. [10] analyzed traffic patterns by using statistical network analysis. Djidjev represented large computer traffic networks as time-labeled graphs and made use of temporal characteristics to partition the graph in subgraphs where they called them telescoping subgraphs. The statistical analysis aimed to explore characteristics of the subgraphs statistically. As statistical techniques, Djidjev et al. [10] applied methods from supervised learning.

Lancichinetti et al. [20] introduced a new measure called C-score aiming at quantifying the statistical significance of communities in networks. The C-score is the probability of occurrence of a community that has the same number of nodes with the same degree sequence and the same internal connections under two hypothesis: The first is that the nodes are randomly connected in the network, and the second is that the group is chosen. In order to predict the statistics associated with individual clusters, statistical measures have been used. The proposed measure of C-score has been successfully tested on several networks such as the random graphs, artificial networks and real networks.

Guillaume [17] studied p2p query graph by using statistical graph analysis. To to so, Guillaume used complex network analysis methods to analyze large traces of queries and exchanges processed in such p2p query systems. The authors defined a labeled weighted bipartite graph called the query graph representing the query information. In this query graph, the time-evolution of degrees has been mainly studied. As demonstrated in [17], degree distributions of query graph follow power laws.

A similar analysis using randomization techniques have been performed in [31]. In this paper, two edge-based randomization techniques have been introduced. More precisely, Ying and Wu [31] developed spectrum preserving randomization methods. The proposed method has been proven useful when preserve graph properties meaningfully. In [31], Ying and Wu mainly focused on two important eigenvalues of graph spectrum, namely the largest eigenvalue of the adjacency matrix and the second largest eigenvalue of the Laplacian matrix respectively.

In [21], community structures have been explored too. Here, the problem has been explored based on the local optimization of a fitness function that expresses the statistical significance of clusters. Also Lancichinetti et al. [21] the Order Statistics Local Optimization method (OSLOM) to detect clusters in large complex networks. In order to give evidence, mathematical properties of OSLOM have been explored and the methods have been demonstrated by employing real world data sets, see [21].

Albert and Barabási [1] performed significant work in statistical network analysis dealing with exploring dynamics of complex networks. In [1], important network models are discussed which include random graphs, small-world networks, scale-free networks and evolving networks. Those have been analyzed statistically, see [1].

Simpson et al. [24] explored functional brain networks statistically. First they performed a survey on the statisti-

cal methods to analyze these networks and for exploring functional magnetic resonance imaging (fMRI) network data. Moreover, they also discussed techniques for modeling and inferring brain networks.

In order to model social networks by using statistical models, exponential random graph models (ERGMs) have been used. Snijders et al. [25] explored convergence problems of estimation algorithms and inference problems using these ERGMs. To tackle this problem they used new specifications of ERGMs allowing to represent structural properties such as transitivity and heterogeneity of complex social networks. As result, they have demonstrated that their new model outperform classical techniques.

### III. SUMMARY AND CONCLUSION

In this paper we performed a brief survey of the recent literature on statistical analysis of networks. For instance, we reviewed contributions dealing with statistical properties of complex networks like the degree distribution, the clustering coefficient, and other statistical analysis techniques such as resampling, bootstrapping, randomization and so forth. We see that those statistical techniques are suitable to investigate so-called non-deterministic networks. That means, we refer to networks that cannot be inferred deterministically as in graph theory. Therefore we believe that these approaches complement classical ones meaningfully and, hence, we continue doing research in this field.

### IV. ACKNOWLEDGEMENTS

We gratefully acknowledge financial support from the German Federal Ministry of Education and Research (BMBF) (project RiKoV, Grant No. 13N12304).

### REFERENCES

- [1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.
- [2] Jeffrey D Allen, Yang Xie, Min Chen, Luc Girard, and Guanghua Xiao. Comparing statistical methods for constructing large scale gene networks. *PloS one*, 7(1):e29348, 2012.
- [3] Gökmen Altay and Frank Emmert-Streib. Inferring the conservative causal core of gene regulatory networks. *BMC Systems Biology*, 4:132, 2010.
- [4] Vladimir Boginski, Sergiy Butenko, and Panos M Pardalos. Statistical analysis of financial networks. *Computational statistics & data analysis*, 48(2):431–443, 2005.
- [5] M. Dehmer. *Strukturelle Analyse web-basierter Dokumente*. Multimedia und Telekooperation. Deutscher Universitäts Verlag, Wiesbaden, 2006.
- [6] M. Dehmer, F. Emmert-Streib, A. Graber, and A. Salvador, editors. *Applied Statistics for Network Biology*. Quantitative and Network Biology. Wiley-Blackwell, 2011.
- [7] M. Dehmer, M. Grabner, A. Mowshowitz, and F. Emmert-Streib. An efficient heuristic approach to detecting graph isomorphism based on combinations of highly discriminating invariants. *Advances in Computational Mathematics*, 39:311–325, 2012.
- [8] M. Dehmer, M. Grabner, and K. Varmuza. Information indices with high discriminative power for graphs. *PLoS ONE*, 7:e31214, 2012.
- [9] Matthias Dehmer and Subhash C Basak. *Statistical and machine learning approaches for network analysis*. Wiley Online Library, 2012.
- [10] Hristo Djidjev, Gary Sandine, Curtis Storlie, and Scott Vander Wiel. Graph based statistical analysis of network traffic. In *Proceedings of the Ninth Workshop on Mining and Learning with Graphs*, 2011.

- [11] Hoda Eldardiry and Jennifer Neville. A resampling technique for relational data graphs. In *Proceedings of the 2nd SNA workshop, 14th ACM SIGKDD conference on knowledge discovery and data mining*, 2008.
- [12] F. Emmert-Streib and M. Dehmer. *Analysis of Microarray Data: A Network-Based Approach*. Wiley-VCH, 2008. Weinheim, Germany.
- [13] F. Emmert-Streib, R. De Matos Simoes, S. Tripathi, G. V. Glazko, and M. Dehmer. Bayesian analysis of the chromosome architecture of human disorders by integrating reductionist data. *Scientific Reports (Nature Publishing)*, 2, 2012.
- [14] James Fairbanks, David Ediger, Rob McColl, David A Bader, and Eric Gilbert. A statistical framework for streaming graph analysis. In *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, pages 341–347. IEEE, 2013.
- [15] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.
- [16] Anna Goldenberg, Alice X Zheng, Stephen E Fienberg, and Edoardo M Airolidi. A survey of statistical network models. *Foundations and Trends® in Machine Learning*, 2(2):129–233, 2010.
- [17] Jean-Loup Guillaume, Matthieu Latapy, and Stevens Le-Blond. Statistical analysis of a p2p query graph based on degrees and their time-evolution. In *Distributed Computing-IWDC 2004*, pages 126–137. Springer, 2005.
- [18] Steve Hanneke, Wenjie Fu, Eric P Xing, et al. Discrete temporal models of social networks. *Electronic Journal of Statistics*, 4:585–605, 2010.
- [19] Mark Huisman and Tom AB Snijders. Statistical analysis of longitudinal network data with changing composition. *Sociological methods & research*, 32(2):253–287, 2003.
- [20] Andrea Lancichinetti, Filippo Radicchi, and José J Ramasco. Statistical significance of communities in networks. *Physical Review E*, 81(4):046110, 2010.
- [21] Andrea Lancichinetti, Filippo Radicchi, José J Ramasco, and Santo Fortunato. Finding statistically significant communities in networks. *PLoS one*, 6(4):e18961, 2011.
- [22] Sang Hoon Lee, Pan-Jun Kim, and Hawoong Jeong. Statistical properties of sampled networks. *Physical Review E*, 73(1):016102, 2006.
- [23] Constantine Manikopoulos and Symeon Papavassiliou. Network intrusion and fault detection: a statistical anomaly approach. *Communications Magazine, IEEE*, 40(10):76–82, 2002.
- [24] Sean L Simpson, F DuBois Bowman, and Paul J Laurienti. Analyzing complex functional brain networks: fusing statistics and network science to understand the brain. *Statistics surveys*, 7:1, 2013.
- [25] Tom AB Snijders, Philippa E Pattison, Garry L Robins, and Mark S Handcock. New specifications for exponential random graph models. *Sociological methodology*, 36(1):99–153, 2006.
- [26] Augustin Soule, Kavé Salamatian, and Nina Taft. Combining filtering and statistical methods for anomaly detection. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 31–31. USENIX Association, 2005.
- [27] Marina Thottan and Chuanyi Ji. Anomaly detection in ip networks. *Signal Processing, IEEE Transactions on*, 51(8):2191–2204, 2003.
- [28] Nicolas Tremblay, Alain Barrat, Cary Forest, Mark Norberg, Jean-François Pinton, and Pierre Borgnat. Bootstrapping under constraint for the assessment of group behavior in human contact networks. *Physical Review E*, 88(5):052812, 2013.
- [29] JR Van Dorp and MR Duffey. Statistical dependence in risk analysis for project networks using monte carlo methods. *International Journal of Production Economics*, 58(1):17–29, 1999.
- [30] Biao Xing and Mark J van der Laan. A statistical method for constructing transcriptional regulatory networks using gene expression and sequence data. *Journal of Computational Biology*, 12(2):229–246, 2005.
- [31] Xiaowei Ying and Xintao Wu. Randomizing social networks: A spectrum preserving approach. In *SDM*, volume 8, pages 739–750. SIAM, 2008.



**SESSION**  
**POSTER PAPERS**

**Chair(s)**

**TBA**



# A Dependable Language for Low Power Embedded Systems

Yuan-Ming Chang, Chia-Chen Hsu, and Jenq-Kuen Lee

Department of Computer Science, National Tsing-Hua University, Taiwan

**Abstract**—*Minimizing the power consumption is crucial for embedded systems. Previous researches have successes for power optimization with stateless components in embedded processors. Recently, researches have started to design the architecture in minimizing the supply voltage for stateful components. However, lowering voltage also increases the risks of reliability. In this paper, we present a dependable language, which defines several expanded syntax rules. With this language, developers can describe the region of critical data and the region hoping for stored in low voltage region of memories. The language provides ways for programmers to participate in exploiting the variability and reliability issues of hardware designs.*

## 1. Introduction

Minimizing the power consumption is crucial for embedded systems[1][2]. One method to reduce overall power consumption is lowering the supply voltage. For example, Abdel-Majeed *et al.*[3] propose a drowsy state which uses retention voltage to keep data alive in a lower voltage. While lowering supply voltage can save power consumption, it also hurts the reliability of the system. To be exact, the probability of error increases as the supply voltage is lowered[4][5]. There are researches which are motivated to detect soft error and present approaches to recover faults. Gao *et al.*[6] proposes explicit output comparison to identify faults. However, these researches are lack of flexibility, such as deciding which parts of the programs should be executed in low power mode or should be protected.

To enable programmers to participate in exploiting the variability and reliability issues of hardware designs, we propose a dependable language. By proposed pragma, programmer can decide whether the data/functions be protected, or whether the data/functions be put in low power mode. With cooperated architecture, our system can partially execute low power mode and guarantee the reliable of certain instructions.

## 2. Syntax Rules for Dependable Language

We propose several dependable pragma, which can support all kinds of data type, such as char, int, short, long. Besides, they can also support several conditional statements, such as for, while, if, do. In the following, we introduce proposed dependable language and their meanings.

- reliable**  
 reliable means declared value/function needs to be protected. Following are some examples:  
*reliable int value* means the declared value is a reliable integer type and it needs to be protected.  
*reliable for (...)* means the declared for-loop is a reliable for-loop, and all values in the loop scope need to be protected.  
*reliable( output | count ) for (...)* means variable "output" and "count" in the loop scope need to be protected, and others remain normal.
- dllpRegion**  
 dllpRegion means declared value/function would be put in memory with low supply voltage or in certain memory region. Following are some examples:  
*dllpRegion int value* means the declared value is a dllpRegion integer type, and it is stored with low support voltage.  
*dllpRegion for (...)* means the declared for-loop is a reliable for-loop, and all values in loop scope would be stored with low voltage.  
*dllpRegion( output | count > \$r1 ) for (...)* means that, in the for loop, variable "output" and "count" would be stored in assigned region, \$r1.
- reliable dllpRegion**  
 reliable dllpRegion means declared value/ function would not only be put in certain memory (low voltage region or other certain region) but also be protected. For example:  
*reliable dllpRegion int value* means the declared value is a reliable dllpRegion integer type, and it needs to be protected and be stored with low support voltage.  
*reliable dllpRegion for (...)* means the declared for-loop is a reliable dllpRegion for-loop, and all values in loop scope needs to be protected and be stored with low support voltage.  
*reliable( output | count ) dllpRegion( output | count > \$r1 ) for (...)* means variable "output" and "count" in the loop scope need to be protected and be stored in region1 \$r1.  
*reliable dllpRegion( output > \$r0 | count > \$r1 ) for (...)* means the declared for-loop is a reliable for-loop, so all variables in the loop should be protected. Also, variable "output" and "count" in the loop scope need to be stored in region 0(\$r0) and region 1(\$r1)

respectively.

*reliable( output | count ) dllpRegion( count > \$r1 | \$other > \$r0 ) for (...)* means variable "output" and "count" in the loop scope need to be protected. Moreover, "count" value should be stored in region 1(\$r1), and other values are stored in region 0(\$r0).

To illustrate how to use proposed language, we take a *Low Power Smart Trash* as an example. The trash keeps in sleep mode most of time. In sleep mode, the support voltage of memory is kept low to save power. When people throw the garbage to it, the trash occurs interrupt and detects whether the trash is filled. The sample code of smart trash is shown in Listing 1 and the interrupt function is shown in Listing 2. In Listing 1, *inPin0* and *inPin1* are the input ports of the trash(GPIO pins of sensor). We should guarantee the accuracy of the input ports because the system relies on it to get inputs. Since the input ports of a system are frequently used and important, we put them in regular memory. If inputs were put in low voltage region of memory, the system is difficult to be recovered when the error occurs. *Threshold*, declared in line 6, is the value based on it the system judge whether the trash is filled. We declare *Threshold* in *dllpRegion* type rather than putting in the memory of regular voltage because it can be recovered once the error occurs. While we also declare it in *reliable* type because *reliable* promises the system would detect errors, if any, and solve them. Without declaring in *reliable* type, the systems never check the correctness of the data. *lightSensor* and *pressureSensor* are the value of current pressure and light. When people throw trash, the interrupt is triggered (as shown in Listing 2). We have *lightSensor / pressureSensor* get data from reading *inPin0 / inPin1*. Since *lightSensor* and *pressureSensor* refresh every time when calling interrupt function, they are relatively less important than *inPin0 / inPin1*. Therefore we declare them in *dllpRegion* type, so does the *Text*. By this example, we present how to save energy without lose accuracy by flexibly using proposed dependable language.

```

1  /* pushbutton connected to digital pin 7, 8. */
2
3  int inPin0 = 7;  int inPin1 = 8;
4
5  /* The declared value needs protection, and it is
6     stored other region. */
7  reliable dllpRegion int threshold = 80;
8
9  /* The declared value is loaded and stored other
10     region. */
11
12 dllpRegion int lightSensor, pressureSensor;
13 dllpRegion char Text[20] = "Trash is full";

```

Listing 1: Example of IoT Smart Trash

```

1 void interrupt_handler(){
2
3     /* read the input pin. */
4     lightSensor = digitalRead(inPin0);
5     pressureSensor = digitalRead(inPin1);
6
7     if (pressureSensor > threshold
8         && lightSensor){
9         sendData(Text);
10
11         ...
12     }
13 }

```

Listing 2: Interrupt function

### 3. Conclusion

We propose a prototype of the dependable language which allows users to feasibly store data in low voltage region of memories and protect critical data. We also use an example of smart trash to demonstrate how to use this language. The language is our attempt to provide programmers ways to exploit the variability and reliability issues of hardware designs.

### Acknowledgment

The work is supported in part by III and by Taiwan MOST.

### References

- [1] Yi-Ping You, Chung-Wen Huang, and Jenq Kuen Lee, "Compilation for Compact Power-Gating Controls," ACM Transactions on Design Automation of Electronic Systems, Vol. 12, Issue 4, Article 51, ACM, New York, September 2007.
- [2] Wen-Li Shih, Yi-Ping You, Chung-Wen Huang, and Jenq-Kuen Lee, "Compiler Optimization for Reducing Leakage Power in Multi-thread BSP Programs," ACM Transactions on Design Automation of Electronic Systems, Vol. 20, Issue 1, Article 9, November, 2014.
- [3] Mohammad Abdel-Majeed, Murali Annavaram, "Warped Register File: A Power Efficient Register File for GPGPUs," HPCA '13 Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), 2013.
- [4] Mark Gottscho, Abbas BanaiyanMofrad, Nikil Dutt, Alex Nicolau, Puneet Gupta, "Power / Capacity Scaling: Energy Savings With Simple Fault-Tolerant Caches," DAC '14 Proceedings of the 51st Annual Design Automation Conference Pages 1-6 ACM New York, NY, USA 2014
- [5] Nikil Dutt, Puneet Gupta, Alex Nicolau, Abbas BanaiyanMofrad, Mark Gottscho, Majid Shoushtari, "Multi-Layer Memory Resiliency," DAC '14: Proceedings of the 51st Annual Design Automation Conference, June 2014
- [6] Gao. Yue, Gupta. Sandeep K, Breuer. Melvin A, "Using Explicit Output Comparisons for Fault Tolerant Scheduling (FTS) on Modern High- Performance Processors," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013

# COMPARING GRAPHS REPRESENTING CHRONOLOGICALLY ORDERED EVENTS

Katia Mayfield

Department of Mathematics, Computer and Natural Sciences, Athens State Univ., Athens, AL USA

**Abstract** – *The use of graphs to represent chronologically ordered events may be justified in several different applications. Two graphs from different sources representing the same sets of events may need to be compared to verify their similarities or correctness if one of the graphs is assumed to be the expected representation. This study discusses possible accuracy scoring systems that can be applied in such situations.*

**Keywords:** graph, similarity, accuracy, compassion, LCS.

## 1 INTRODUCTION

There are many scenarios and situations where the occurrence of events can be readily represented by graphs. In such graphs, edges are used to represent dependency between events. Consider for example a sequence of historical facts, beginning with Abraham Lincoln being elected president in 1860, his anti-slavery outlook caused South Carolina along with six other states to secede from the Union[8]. In the example, one event became the cause for the next that occurred. At the same time, other events may occur that are independent of one another. For example, Lincoln becomes President; the Civil War takes place as a result of his presidency; and Lincoln invents a tool to lift riverboats stuck on sandbars [4].

Sometimes researchers involved with historical events have to estimate a sequence of events. A comparison between these estimates and extant documentation may result in ascertaining the accuracy of the estimation. A similar situation happens with a literary critic trying to find the evolutionary path of a text, going through different versions. In order to be able to determine the accuracy of the estimation

method, a numerical scoring system has to be applied.

## 2 BACKGROUND

Woon and Wong proposed the use of a new scoring system for their study in text versions restricted to graphs consisting of a single linear path and establishing windows of comparison along such path, which allowed one correct result to be counted multiple times, according to the size of the window, when a node preceded any of its actual successors in the path [9]. In the field of Biology, comparison of tree structures is commonly applied to the analysis of the evolution of species [1]. Some of these algorithms address the graph or tree topology, focusing on leaf nodes organized in quartets: groups of four labeled nodes divided by two internal nodes [3].

In the field of Applied Mathematics, a number of graph comparison techniques are focused on the node placement in the graph. Some of those methods are variations of the Levenshtein distance metric and the Hamming distance methodologies applied to strings representing a Depth-First traversal of the graph [2]. A more suitable algorithm for our problem, known as the Partition Metric, is also used in the field of Biology and considers both, topology and ordering of the nodes in a path, without requiring labeled edges [7].

## 3 SCORING ALGORITHMS

In this study we evaluated a slightly modified version of the Partitioning Metric algorithm to measure the accuracy of the results of a version evolution estimator tool. The algorithm compares two graphs, which contain the same set of nodes, by searching for edges that create equal partitions of nodes in both graphs. A one

point score is awarded to every edge that creates matched partitions, basically measuring the similarity between the graphs.

Table 1 shows the accuracy of the algorithm working with Kruskal's Minimum Spanning Tree (MST) and a modified version of the Hamiltonian path, Single Path Evolution (SPE), options in the estimation of evolutionary versions of a text [6]. As it can be seen, under this scoring system the algorithm is able to correctly identify the sequence in several cases. However, some results seem to show a low accuracy, which is a problem already identified by other researchers with this scoring system where a single node mismatch may cause a high difference, low score, in the graph topology depending on its new placement [7]. Exploring the possibility of utilizing the solution based on the distance metric applied to strings representing a Depth-First traversal of the graph, a new scoring system can be developed where the similarity of the graphs could be measured by using the Longest Common Substring algorithm [2, 6].

The scoring obtained through this method is less sensitive to single nodes mismatch and therefore closer to the actual measurement of the similarities of the graphs. The LCS algorithm produced exactly the same results for the test cases shown in Table 1. An extra test case based on William Shakespeare's "*Henry V*", which causes the mismatch node anomaly in the Partition method, gave LCS a better accuracy representation score.

#### 4 ANALYSIS AND CONCLUSION

An in-depth analysis of the test case results with low accuracy showed that the lower results were a consequence of a backward path, preceded by a jump from the original to the last version.

These works, in which the evolutionary sequence can be verified, allowed for the establishment of a benchmark in the discovery of evolutionary paths.

Table 1 Partition Metric Results

| Test Case   | Minimum Spanning Tree | SPE  |
|-------------|-----------------------|------|
| EBB Child   | 100%                  | 100% |
| EBB Bettine | 66%                   | 66%  |
| EBB Sea     | 33%                   | 33%  |
| EBB Loved   | 100%                  | 100% |
| EBB Clouds  | 100%                  | 100% |
| EBB Dog     | 50%                   | 50%  |
| EBB Mitford | 100%                  | 100% |
| WW Leaves   | 100%                  | 100% |
| WS Hamlet   | 66%                   | 66%  |

#### 5 REFERENCES

1. Albright, E., Hessel, J., Hiranuma, N., Wan, C., Goings, S. "A Comparative Analysis of Popular Phylogenetic Reconstruction Algorithm," MICS 2014 Proceedings, 2014.
2. Cao, B., Li, Y., Yin, J. "Measuring Similarity between Graphs Based on the Levenshtein Distance," Appl. Math, Inf. Sci., pp 169-175, 2013.
3. Christiansen, C., Mailund, T., Pedersen, C.N., Randers, M. "Computing the quartet distance between trees of arbitrary degree," Springer, 2005.
4. Edwards, O. "Abraham Lincoln: the Ingenious Inventor," Smithsonian Magazine, October 2006.
5. Irace, K O., *The First Quarto of "Hamlet."* Cambridge: Cambridge University Press (The New Cambridge Shakespeare), 1998.
6. Neapolitan, R., Naimipour, K. "Foundations of Algorithms Using C++ Pseudocode," Jones and Bartlett, Inc., New York, 2004.
7. Penny, D., Hendy, M.D., "The Use of Tree Comparison Metrics," Systematic Zoology, vol. 34, No. 1, March 1985, pp. 75-82.
8. (2014, May). Saving America's Civil War Battlefields: Civil War Trust [Online] available: <http://www.civilwar.org/>
9. Woon, W.L., Wong, K.D. "String alignment for automated document versioning," Knowledge Information Systems, 2009, pp. 293-309.