Volume 16 Number 1

ACTA CYBERNETICA

Szeged, 2003

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Contributions are accepted for review with the understanding that the same work has not been published elsewhere.

Manuscripts must be in English and should be sent in triplicate to any of the Editors. On the first page, the *title* of the paper, the *name(s)* and *affiliation(s)*, together with the *mailing* and *electronic address(es)* of the author(s) must appear. An *abstract* summarizing the results of the paper is also required. References should be listed in alphabetical order at the end of the paper in the form which can be seen in any article already published in the journal. Manuscripts are expected to be made with a great care. If typewritten, they should be typed double-spaced on one side of each sheet. Authors are encouraged to use any available dialect of TEX.

After acceptance, the authors will be asked to send the manuscript's source TEX file, if any, on a diskette to the Managing Editor. Having the TEX file of the paper can speed up the process of the publication considerably. Authors of accepted contributions may be asked to send the original drawings or computer outputs of figures appearing in the paper. In order to make a photographic reproduction possible, drawings of such figures should be on separate sheets, in India ink, and carefully lettered.

There are no page charges. Fifty reprints are supplied for each article published.

Publication information. Acta Cybernetica (ISSN 0324–721X) is published by the Department of Informatics of the University of Szeged, Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. For 2003 Numbers 1–2 of Volume 16 are scheduled. Subscription prices are available upon request from the publisher. Issues are sent normally by surface mail except to overseas countries where air delivery is ensured. Claims for missing issues are accepted within six months of our publication date. Please address all requests for subscription information to: Department of Informatics, University of Szeged, H-6701 Szeged, P.O.Box 652, Hungary. Tel.: (36)-(62)-420-184, Fax:(36)-(62)-420-292.

URL access. All these information and the contents of the last some issues are available in the Acta Cybernetica home page at http://www.inf.u-szeged.hu/local/acta.

Temporal Logic with Cyclic Counting and the Degree of Aperiodicity of Finite Automata*

Z. Ésik[†] and M. Ito[‡]

Abstract

We define the degree of aperiodicity of finite automata and show that for every set M of positive integers, the class $\mathbf{QA}_M$ of finite automata whose degree of aperiodicity belongs to the division ideal generated by M is closed with respect to direct products, disjoint unions, subautomata, homomorphic images and renamings. These closure conditions define q-varieties of finite automata. We show that q-varieties are in a one-to-one correspondence with literal varieties of regular languages. We also characterize $\mathbf{QA}_M$ as the cascade product of a variety of counters with the variety of aperiodic (or counter-free) automata. We then use the notion of degree of aperiodicity to characterize the expressive power of first-order logic and temporal logic with cyclic counting with respect to any given set M of moduli. It follows that when M is finite, then it is decidable whether a regular language is definable in first-order or temporal logic with cyclic counting with respect to moduli in M.

1 Introduction

The richness of the theory of regular languages is due to the many different characterizations of (subclasses of) regular languages. By the theorem of Büchi and Elgot, a language is regular iff it is definable in monadic second-order logic over words [3, 6] involving the predicate $<$ and a predicate corresponding to each letter of the alphabet. Moreover, by classic results of Schützenberger [14] and Mc

*This paper is a revised version of the author's technical report RS-01-53, BRICS, December, 2001.

†Dept. of Computer Science, University of Szeged, Szeged, Hungary.
Email: `esik@inf.u-szeged.hu`
The results of this paper were obtained during the first author's visit at the Faculty of Science at the Kyoto Sangyo University and at the Department of Computer Science at the University of Aalborg.
The first author was partially supported by Grant-in-Aid 10044098, Japan Society for the Promotion of Science, by BRICS (Basic Research in Computer Science), and by grant no. T30511 from the National Foundation of Hungary for Scientific Research.

‡Dept. of Mathematics, Kyoto Sangyo University, Kyoto, Japan.
Email: `ito@ksuvx0.kyoto-su.ac.jp`
Partially supported by Grant-in-Aid 10044098, Japan Society for the Promotion of Science.

Naughton and Papert [11], a language is star-free iff it is definable in first-order logic iff it is accepted by an aperiodic (or counter-free) finite automaton. Thus, it is decidable for a regular language whether or not it is definable in first-order logic, or has a star-free expression. Moreover, by a classic result of Kamp [10] and Gabbay et al. [8], the logic **LTL** of Linear (Propositional) Temporal Logic over words has the same expressive power as first-order logic.

The above results have been extended in several directions involving, in addition to words, also ω-words, trees and other structures, see [18, 19] for overviews. In order to increase the expressive power of first-order logic on words, two kinds of cyclic counting have been studied: the extension of first-order logic with numerical predicates $C_m^r(x)$ that holds for a position x in a word iff x is congruent to r modulo m, see [1, 16], and the extension with modular quantifiers, cf. [17, 16]. In this paper our concern is the first type of counting. In [1], Barrington, Compton, Straubing and Therien gave a decidable characterization of the languages definable in first-order logic with counting with respect to the predicates $C_m^r(x)$, where the modulus m ranges over all positive integers and r is any nonnegative integer $< m$. However, this characterization does not answer the question that, given a finite set M of moduli, what languages are definable by using only predicates involving moduli in M. Our aim in this paper is to provide an analysis of the above mentioned result of Barrington, Compton, Straubing and Therien that will provide an answer to the previous question. Moreover, we also study an extension of temporal logic yielding the same expressive power.

We define the degree of aperiodicity of finite automata and show that for every set M of positive integers, the class $\mathbf{QA}_M$ of automata whose degree of aperiodicity belongs to the division ideal generated by M is closed with respect to direct products, disjoint unions, subautomata, homomorphic images and renamings. These closure conditions define q-varieties. We show that q-varieties are in a one-to-one correspondence with literal varieties of regular languages. We also characterize $\mathbf{QA}_M$ as the cascade product of a variety of counters with the variety of aperiodic (or counter-free) automata. We then use the notion of degree of aperiodicity to characterize the expressive power of first-order logic and temporal logic with cyclic counting with respect to any given set M of moduli. When M is finite, this characterization is effective.

The paper is organized as follows. In Section 2 we define literal varieties of regular languages, q-varieties of finite automata, and establish an Eilenberg-type correspondence between them. In Section 3, we recall the notion of cascade product of finite automata together with a few basic facts regarding regular languages accepted by cascade products. We also define cascade products $\mathbf{V} \star \mathbf{W}$ of q-varieties. Then, in Section 4, we study q-varieties of finite automata of the form $\mathbf{C}_M \star \mathbf{V}$, where M is a given subset of the positive integers and $\mathbf{C}_M$ is the q-variety generated by all counters whose length belongs to M. Then, in Section 5, we define the degree of aperiodicity of finite automata and show that for every set M as above, the finite automata whose degree of aperiodicity belongs to the division ideal generated by M form a q-variety $\mathbf{QA}_M$ which is the cascade product of $\mathbf{C}_M$ with the q-variety of aperiodic (counter-free) automata. Moreover, we show that the degree of aperiodicity of a finite automaton is computable. We also show that a language can be recognized by an automaton in $\mathbf{QA}_M$ iff it can be constructed from the finite languages and the languages consisting of all words over the underlying alphabet whose length is a multiple of some integer in M by the boolean operations and concatenation. Then, in Section 6 we prove that the very same condition characterizes the languages definable in first-order logic with cyclic counting with respect to moduli in M. When M is empty or M is the set of all positive integers, these results correspond to those of Schützenberger [14], Mc Naughton and Papert [11], and Barrington et al. [1] mentioned above. In Section 7, we provide several extensions of propositional temporal logic with cyclic counting and show that all these are equivalent. Moreover, we show that temporal logic with cyclic counting with respect to any given set M of moduli has the same expressive power as first-order logic with counting with respect to moduli in M. When M is empty, this fact corresponds to the result of Kamp [10] and Gabbay et al. [8]. Section 8 contains a summary of the results obtained and outlines some future results.

We have tried to make the paper accessible for a wider audience.

2 An Eilenberg correspondence

A *finite alphabet*, or just alphabet, for short, is any finite nonempty set whose elements are called *letters*. When Σ is an alphabet, we let Σ^* denote the free monoid of *words* over Σ including the *empty word* ϵ equipped with the operation of *concatenation* as product. For any word $u = a_0 \dots a_{n-1}$, where the a_i are letters, we call the integer n the length of u and denote it by $|u|$. We let Σ^n denote the set of all words in Σ^* of length n. The *prefix order* $\leq$ on words is defined by $u \leq v$ iff there is a word z with $uz = v$, i.e., when u is a prefix of v. Suppose that h is a (monoid) homomorphism $\Sigma^* \to \Delta^*$, where Σ, Δ are finite alphabets. We call h *nonerasing* if $ah \neq \epsilon$ holds for all $a \in \Sigma$. Moreover, we call h a *literal homomorphism* if $ah \in \Delta$ holds for all $a \in \Sigma$.

A *language* (over Σ) is any subset of Σ^*. Languages over Σ are equipped with several operations including the boolean operations $\cup$, $\cap$ and ^{c} (complement), product (or concatenation), Kleene star (*), left and right quotients, homomorphisms, inverse homomorphisms, etc. These are defined in the standard way. When $L \subseteq \Sigma^*$ and $u \in \Sigma^*$, we let $u^{-1}L$ and Lu^{-1} denote the left and right quotients of L with respect to u, respectively:

$$\begin{aligned} u^{-1}L &= \{v \in \Sigma^* : uv \in L\} \\ Lu^{-1} &= \{v \in \Sigma^* : vu \in L\} \end{aligned}$$

We will sometimes identify a word w with the singleton set $\{w\}$ and write w^* for the Kleene star $\{w\}^*$ of the language $\{w\}$.

Recall that a language $L \subseteq \Sigma^*$ is called *regular* if it can be constructed from the finite subsets of Σ^* by the regular operations of union, product and Kleene star. It is well-known that the class of regular languages is closed with respect to all of the operations mentioned above. Moreover, by Kleene's classic theorem, the regular languages are exactly those languages that can be recognized by finite automata.

In this paper, by a *finite automaton*, or just automaton, we mean a system $Q = (Q, \Sigma, \cdot)$ consisting of a finite nonempty set Q of *states*, a finite *input alphabet* Σ and a right *action* of Σ on Q, i.e., a function $\cdot : Q \times \Sigma \to Q$, which is extended to an action of Σ^* on Q in the usual way. Below we will usually write just qu for $q \cdot u$, for all $q \in Q$ and $u \in \Sigma^*$. The function $q \mapsto qu$ is called the *function induced by* u, denoted u^Q. When we want to emphasize that the input alphabet of an automaton is some alphabet Σ, we call it a Σ-*automaton*. Suppose that $L \subseteq \Sigma^*$ and that $Q = (Q, \Sigma, \cdot)$ is a Σ-automaton. We say that L *is recognizable in* Q, or that L *can be recognized by* Q, if there are a state $q_0 \in Q$, the *initial state*, and a set $F \subseteq Q$ of *final states* such that $L = \{u \in \Sigma^* : q_0u \in F\}$. Moreover, a language is called *recognizable* if it can be recognized by some finite automaton. The aforementioned theorem of Kleene equates the recognizable languages with the regular languages.

Recall [5, 12] that a *stream (or class)* $\mathcal{V}$ *of regular languages* is a nonempty collection $\Sigma^*\mathcal{V}$ of regular languages over Σ, for each finite alphabet Σ. Streams of regular languages are ordered by set inclusion: we write $\mathcal{V} \subseteq \mathcal{V}'$ if $\Sigma^*\mathcal{V} \subseteq \Sigma^*\mathcal{V}'$, for all finite alphabets Σ.

Definition 2.1. *A* literal variety (of languages), *or l-variety, for short, is a stream* $\mathcal{V}$ *of regular languages closed with respect to the boolean operations, left and right quotients and inverse literal homomorphisms. Thus, if* $L_1, L_2 \in \Sigma^*\mathcal{V}$ *and* $a \in \Sigma$, *then* $L_1 \cup L_2$, $L_1 \cap L_2$, L_1^c, $a^{-1}L_1$ *and* L_1a^{-1} *are all in* $\Sigma^*\mathcal{V}$. *Moreover, if h is a literal homomorphism* $\Delta^* \to \Sigma^*$, *so that* $\Delta h \subseteq \Sigma$, *then* $L_1h^{-1} \in \Delta^*\mathcal{V}$.

A $*$-variety ($+$-variety, respectively) of languages *is a literal variety which is closed with respect to all (nonerasing, respectively) inverse homomorphisms.*

Example 2.2. It is clear that l-varieties form a complete lattice, in fact, an algebraic lattice. The largest l-variety contains, for each Σ, all the regular languages in Σ^*, and the smallest only the empty language and the language Σ^*. When $\{\mathcal{V}_i : i \in I\}$ is a directed set of l-varieties, the least upper bound $\mathcal{V} = \bigvee_{i \in I} \mathcal{V}_i$ is just the union $\bigcup_{i \in I} \mathcal{V}_i$, so that $\Sigma^*\mathcal{V} = \bigcup_{i \in I} \Sigma^*\mathcal{V}_i$, for each Σ.

Example 2.3. Of course, every $*$-variety or $+$-variety is a literal variety. For each Σ, let $\Sigma^*\mathcal{L}$ consist of all regular languages L in Σ^* such that for all words $u, v \in \Sigma^*$, if $u \in L$ and $|u| = |v|$, then $v \in L$. Then $\mathcal{L}$ is a literal variety which is not a $+$-variety or a $*$-variety.

The l-varieties contained in $\mathcal{L}$ correspond to those boolean algebras of regular languages over the one-letter alphabet closed with respect to quotients. We give some examples of such varieties.

Suppose that $d \geq 1$ is an integer. The l-variety $\mathcal{C}_d$ is that generated by the one-letter regular language $(a^d)^*$, considered as a subset of a^*. It is not hard to see that each language in $\Sigma^*\mathcal{C}_d$ is a finite union of languages of the form $(\Sigma^d)^*\Sigma^i$, where i is an integer in $[d] = \{0, 1, \ldots, d-1\}$.

Suppose that M is a subset of the set Nat of positive integers. Then let $\mathcal{C}_M$ denote the smallest l-variety containing all of the $\mathcal{C}_m$ with $m \in M$. It is clear that

C_M is the union of those C_d where d is contained in the *division ideal* $(M]$ generated by M. (Of course, $(M]$ consists of all divisors of least common multiples of finite families of elements of M.) Thus, $C_M \subseteq C_{M'}$ iff $(M] \subseteq (M']$. We write C for C_{Nat}.

Further examples of literal varieties that are not $*$-varieties or $+$-varieties will be given later.

Remark 2.4. *The $*$-varieties defined above are the same as the $*$-varieties of Eilenberg [5], see also [12]. However, Eilenberg's $+$-varieties [5] are streams of regular languages containing only* nonempty *words closed with respect to the boolean operations, left and right quotients, and nonerasing inverse homomorphisms. If $\mathcal{V}$ is a $+$-variety as defined in Definition 2.1, and if $\Sigma^+\mathcal{W} = \{L \cap \Sigma^+ : L \in \Sigma^*\mathcal{V}\}$, for each Σ, where Σ^+ denotes the free semigroup of all nonempty words over Σ, then $\mathcal{W}$ is an Eilenberg $+$-variety. This mapping $\mathcal{V} \mapsto \mathcal{W}$ is surjective but not injective.*
Suppose that $\mathcal{W}$ is an Eilenberg $+$-variety. For each alphabet Σ, define

$$\Sigma^*\mathcal{V} = \{L, L \cup \epsilon : L \in \Sigma^+\mathcal{W}\}.$$

Then $\mathcal{V}$ is a $+$-variety, as defined in Definition 2.1, which is mapped to $\mathcal{W}$. If for some Σ, there is a finite nonempty set in $\Sigma^+\mathcal{W}$, then this is in fact the unique $+$-variety mapped to $\mathcal{W}$. However, if $\Sigma^*\mathcal{V} = \{\emptyset, \Sigma^*\}$ and $\Sigma^*\mathcal{V}' = \{\emptyset, \epsilon, \Sigma^+, \Sigma^*\}$, for each alphabet Σ, then the same Eilenberg $+$-variety $\mathcal{W}$ corresponds to both $\mathcal{V}$ and $\mathcal{V}'$:

$$\Sigma^+\mathcal{W} = \{\emptyset, \Sigma^+\},$$

for each Σ.

A *stream (or class)* $\mathbf{V}$ *of finite automata* is a nonempty collection $\Sigma\mathbf{V}$ of finite Σ-automata, for each finite alphabet Σ. Streams of finite automata are ordered by set inclusion in the same way as streams of regular languages.

The notions of *subautomaton* and *quotient* (or *homomorphic image*) of an automaton are defined as usual. When $Q = (Q, \Sigma, \cdot)$ and $Q' = (Q', \Sigma, \cdot)$ are automata with the same set of input letters, the *direct product* $Q \times Q' = (Q \times Q', \Sigma, \cdot)$ is equipped with the pointwise action, so that $(q, q') \cdot a = (qa, q'a)$, for all $q \in Q$, $q' \in Q'$ and $a \in \Sigma$. The *disjoint sum* (or *disjoint union*) of Q and Q' is also defined in the standard way: $Q \oplus Q' = (Q \times \{0\} \cup Q \times \{1\}, \Sigma, \cdot)$, where $(q, 0)a = (qa, 0)$ and $(q', 1)a = (q'a, 1)$, for all $q \in Q$ and $q' \in Q'$. Suppose now that $Q = (Q, \Sigma, \cdot)$ and $Q' = (Q', \Delta, \cdot)$, where Σ and Δ are any alphabets. We say that Q can be constructed from Q' by *renaming*, or that Q is a renaming of Q', if $Q = Q'$ and there is a function $h : \Sigma \to \Delta$ such that $qa = q(ah)$, for all $q \in Q$ and $a \in \Sigma$.

Definition 2.5. *A* q-variety of finite automata *is any stream of finite automata closed with respect to the operations of taking subautomata, quotients, direct products, disjoint sums and renamings.*

We use the prefix to distinguish q-varieties from varieties (or pseudo-varieties) that are nonempty classes of automata with the same input alphabet closed with respect to the operations of taking subautomata, quotients, and direct products, and to express that q-varieties are also closed with respect to the *quasi-direct product* [9].

Since a q-variety $\mathbf{V}$ is nonempty and closed with respect to subautomata, quotients, direct product and renaming, closure under disjoint sum is clearly equivalent to the requirement that the two-element *discrete automaton* with a single input letter belongs to $\mathbf{V}$. (A Σ-automaton is called discrete if it is a disjoint sum of trivial, i.e., one-state Σ-automata.)

A $*$*-variety* ($+$*-variety*) *of finite automata* is a q-variety that is also closed with respect to the operation $Q \mapsto Q^*$ ($Q \mapsto Q^+$). Here, the operation $Q \mapsto Q^*$ is defined as follows. Let $Q = (Q, \Sigma, \cdot)$, say, and let $M(Q)$ denote the *monoid of* Q. Thus, the elements of $M(Q)$ are the functions $u^Q : Q \to Q$ induced by the words $u \in \Sigma^*$, and the product operation in $M(Q)$ is function composition written left-to-right. Now Q^* is $(Q, M(Q), \cdot)$, where for each $q \in Q$ and $u \in \Sigma^*$, $q \cdot u^Q$ is just $qu = q \cdot u$, the image of q under u^Q. The automaton Q^+ is defined in the same way except that its alphabet is $S(Q) = \{u^Q : u \in \Sigma^+\}$, the *semigroup of* Q.

Remark 2.6. *It is clear that* $*$*-varieties of finite automata correspond in a bijective manner to varieties of finite monoids as defined in [5, 12]. Given a* $*$*-variety* $\mathbf{V}$ *of finite automata, the corresponding variety of finite monoids consists of all monoids that are isomorphic to the monoid of some automaton in* $\mathbf{V}$. *However, a similar function mapping* $+$*-varieties of finite automata to varieties of finite monoids is only surjective, but not injective. See also Remark 2.4.*

Example 2.7. The set of all q-varieties equipped with set inclusion is an algebraic lattice. The largest q-variety contains, for each Σ, all Σ-automata, and the smallest one only the discrete Σ-automata. When $\{\mathbf{V}_i : i \in I\}$ is a directed set of q-varieties, the least upper bound $\bigvee_{i \in I} \mathbf{V}_i$ is just the union $\bigcup_{i \in I} \mathbf{V}_i$.

Example 2.8. For each Σ, the q-variety $\mathbf{L}$ consists of all *autonomous* Σ-automata, i.e., all the automata $Q = (Q, \Sigma, \cdot)$ such that $qa = qb$, for all $q \in Q$ and $a, b \in \Sigma$.

Given an integer $d \geq 1$, the q-variety $\mathbf{C}_d$ has, as its members in $\Sigma\mathbf{C}_d$, all the Σ-automata that are disjoint sums of Σ-*counters* of length a divisor of d. A Σ-counter is an automaton $(Q, \Sigma, \cdot)$ such that each letter in Σ induces the *same* cyclic permutation $Q \to Q$. The length of the counter is $|Q|$, the number of states in Q. Note that $\mathbf{C}_d$ is contained in $\mathbf{L}$.

When M is a set of positive integers, then we define $\mathbf{C}_M = \bigvee_{m \in M} \mathbf{C}_m$, so that $\mathbf{C}_M$ is the least q-variety containing all of the $\mathbf{C}_m$ with $m \in M$. Note that $\mathbf{C}_M$ is just the union of the $\mathbf{C}_d$ with d any integer in $(M]$. Thus, $\mathbf{C}_M \subseteq \mathbf{C}_{M'}$ iff $(M] \subseteq (M']$. We denote $\mathbf{C}_{\mathsf{Nat}}$ by $\mathbf{C}$.

Suppose that $\mathbf{V}$ is a q-variety. The corresponding stream $\mathcal{V}$ of regular languages contains those languages in $\Sigma^*\mathcal{V}$ that can be recognized by an automaton in $\Sigma\mathbf{V}$ (by a suitable initial state and a set of final states). Thus, a language $L \subseteq \Sigma^*$ belongs to $\Sigma^*\mathcal{V}$ if and only if there is an automaton $Q = (Q, \Sigma, \cdot)$ in $\mathbf{V}$, a state $q_0 \in Q$ and a set $F \subseteq Q$ such that the language recognized by Q with initial state

q_0 and final states F is L. Alternatively, a (regular) language $L \subseteq \Sigma^*$ belongs to $\Sigma^*\mathcal{V}$ if and only if the *minimal* automaton recognizing L is in $\Sigma\mathbf{V}$.

The following variant of Eilenberg's variety theorem [5, 12] follows by standard arguments.

Theorem 2.9. *The correspondence* $\mathbf{V} \mapsto \mathcal{V}$ *is an order isomorphism from the lattice of q-varieties of finite automata onto the lattice of l-varieties of regular languages. The same correspondence establishes an order isomorphism between $*$-varieties ($+$-varieties) of finite automata and $*$-varieties ($+$-varieties) of regular languages.*

Proof. We briefly sketch the proof of the first statement. If L is in $\Sigma^*\mathcal{V}$, then L is accepted by an automaton in $\mathbf{V}$ by a suitable initial state and a set of final states. By taking the same initial state and the complement of the set of final states, the same automaton accepts L^c. It is also known that any quotient of L can be accepted by the same automaton with suitable initial and final states. Closure with respect to set union follows from the fact that the union of languages accepted by Q_1 and Q_2 can be accepted by the direct product of Q_1 and Q_2. It is clear that $\mathbf{V}_1 \subseteq \mathbf{V}_2$ implies $\mathcal{V}_1 \subseteq \mathcal{V}_2$. Suppose now that $\mathcal{V}_1 \subseteq \mathcal{V}_2$. Assume that $Q = (Q, \Sigma, \cdot) \in \mathbf{V}_1$ is generated by a single state q_0, so that each state $q \in Q$ is of the form q_0u, for some $u \in \Sigma^*$. For each state $q \in Q$, let L_q denote the language accepted by Q with initial state q_0 and single final state q. Since $L_q \in \mathcal{V}_1$ and $\mathcal{V}_1 \subseteq \mathcal{V}_2$, there exists an automaton $Q_q \in \mathbf{V}_2$ accepting L_q with some initial state i_q and some set of final states F_q. Now the direct product of the Q_q contains a subautomaton that can be mapped homomorphically onto Q: take those tuples of the direct product accessible by a word from that tuple whose components are the respective initial states i_q. It follows that each state $s = (s_q)_{q \in Q}$ has a unique component s_q with $s_q \in F_q$, and that the map taking s to this component s_q is a homomorphism onto Q. Since $\mathbf{V}_2$ is closed with respect to direct product, subautomata and homomorphic images, it follows that Q is in $\mathbf{V}_2$. If $Q \in \mathcal{V}_1$ is not generated by a single state, then Q is a quotient of the disjoint sum of its (maximal) one-generated subautomata. Since q-varieties are closed with respect to disjoint sum, it follows by the above argument that $Q \in \mathbf{V}_2$. Finally, the fact that the assignment $\mathbf{V} \mapsto \mathcal{V}$ is surjective can be seen as follows. Given an l-variety $\mathcal{V}$, consider the stream $\mathbf{V}$ of automata that only accept languages in $\mathcal{V}$, so that $Q = (Q, \Sigma, \cdot) \in \mathbf{V}$ iff for each $q_0 \in Q$ and $F \subseteq Q$ it holds that the language accepted by Q with initial state q_0 and set of final states F is in $\mathcal{V}$. Then $\mathbf{V}$ is a q-variety mapped to $\mathcal{V}$. Indeed, the closure properties of $\mathcal{V}$ guarantee that $\mathbf{V}$ is a q-variety. Moreover, every language $L \in \Sigma^*\mathcal{V}$ can be accepted by an automaton in $\mathbf{V}$, namely the minimal automaton Q_L corresponding to L, since any language accepted by this automaton is a boolean combination of quotients of L. $\square$

Example 2.10. The l-variety corresponding to $\mathbf{L}$ is the variety $\mathcal{L}$ defined in Example 2.3. For each M, the l-variety corresponding to $\mathbf{C}_M$ is $\mathcal{C}_M$.

Example 2.11. We call a finite automaton $Q = (Q, \Sigma, \cdot)$ *nilpotent* if there is an integer n such that $qu = qv$ holds for all words $u, v \in \Sigma^*$ of length $\geq n$. (Note that the usual definition of nilpotent automata [9] requires that $qu = q'v$ holds for all states q, q' and words $u, v \in \Sigma^*$ of length at least n.) Nilpotent automata form a $+$-variety denoted $\mathbf{N}$. The corresponding $+$-variety $\mathcal{N}$ of languages consists of all finite and cofinite languages in Σ^*, for each alphabet Σ.

Example 2.12. A finite automaton $Q = (Q, \Sigma, \cdot)$ is called *definite* if there exits some $n \geq 0$ such that for all $q \in Q$ and $u, v \in \Sigma^*$, if the suffixes of u and v of length at most n agree, then $qu = qv$. (Again, the usual definition of definite automata [9] requires more.) For example, any *shift register* $(\Sigma^n, \Sigma, \cdot)$ with $u \cdot a$ being the length n suffix of ua, for each $u \in \Sigma^n$ and $a \in A$, is definite.

Definite automata form a $+$-variety $\mathbf{D}$ with corresponding $+$-variety of languages denoted $\mathcal{D}$. We call $\mathcal{D}$ the $+$-variety of definite languages. For each Σ and $L \subseteq \Sigma^*$, we have $L \in \Sigma^*$ iff there is an integer $n \geq 0$ such that for all words $u, v \in \Sigma^*$ such that u and v have the same suffixes of length at most n, it holds that $u \in L$ iff $v \in L$. (See [5].)

Example 2.13. A finite automaton Q is called *aperiodic*, or *counter-free* [5], if $M(Q)$ (or $S(Q)$) contains only trivial subgroups. Aperiodic automata form a $*$-variety $\mathbf{A}$ with corresponding language variety $\mathcal{A}$. We have that $\mathbf{N} \subset \mathbf{D} \subset \mathbf{A}$ and $\mathcal{N} \subset \mathcal{D} \subset \mathcal{A}$.

3 Cascade product

We call a function $\tau : \Sigma^* \to \Delta^*$ *sequential* if τ preserves prefixes, i.e., for all words u and v in Σ^*, if $u \leq v$ in the prefix order then $\tau(u) \leq \tau(v)$. It then follows that for each word $u \in \Sigma^*$ there is a (unique) function, in fact a sequential function $\tau_u : \Sigma^* \to \Delta^*$ with $\tau(uv) = \tau(u)\tau_u(v)$. If in addition τ preserves the length of the words, then we call τ a *literal sequential function.*

Sequential functions are known to be the functions inducible by sequential transducers, and literal sequential functions by Mealy automata [9], which are a restricted type of transducers. The (literal) sequential functions $\tau : \Sigma^* \to \Delta^*$ that can be induced by finite transducers obey the condition that the functions τ_u, $u \in \Sigma^*$ form a finite set. Such (literal) sequential functions are said to be of *finite state.* Note that any (literal) homomorphism is a finite state (literal) sequential function.

Suppose that $Q = (Q, \Sigma, \cdot)$ is a finite automaton. A *Mealy automaton* [9] over Q is the extension of Q by an output alphabet Δ and an output function $\mu : Q \times \Sigma \to \Delta$. We let $Q(\Delta, \mu)$ denote this extension. Clearly, each state $q \in Q$ may be used to induce a finite state literal sequential function $\mu_q : \Sigma^* \to \Delta^*$ defined by $\mu_q(\epsilon) = \epsilon$ and $\mu_q(ua) = \mu_q(u)\mu(qu, a)$. We use Mealy automata extensions to define *cascade products.*

Suppose that $Q = (Q, \Sigma, \cdot)$ and $R = (R, \Delta, \cdot)$ are finite automata and suppose that we are given a Mealy automaton extension $Q(\Delta, \mu)$ of Q. Then the cascade product of Q with R determined by μ is defined to be the automaton $Q \times_\mu R = (Q \times R, \Sigma, \cdot)$, where $(q,r) \cdot a = (qa, r\mu(q,a)) = (qa, r\mu_q(a))$, for all $q \in Q$ and $r \in R$. Note that it follows by induction that $(q,r) \cdot u = (qu, r\mu_q(u))$, for all $u \in \Sigma^*$.

The semigroup theoretic concepts corresponding to the cascade product are the *semidirect product* and the *wreath product*, cf. [5, 12]. The following fundamental fact is a variant of Straubing's "wreath product principle" [4] to the cascade product.

Proposition 3.1. *A language is recognized by a cascade product $Q \times_\mu R$ with initial state (q_0, r_0) iff it is a finite union of languages of the form $K \cap \mu_{q_0}^{-1}(L)$, where K is a language recognized by Q with initial state q_0 and L is a language recognized by R with initial state r_0.*

The cascade product may be extended to q-varieties.

Definition 3.2. *Suppose that* $\mathbf{V}$ *and* $\mathbf{W}$ *are q-varieties. The q-variety* $\mathbf{V} \star \mathbf{W}$ *is that generated by all cascade products $Q \times_\mu R$ with Q an automaton in* $\Sigma\mathbf{V}$, *R an automaton in* $\Delta\mathbf{W}$, *and $Q(\Delta, \mu)$ a Mealy automaton extension of Q.*

It is immediate to prove that when both $\mathbf{V}$ and $\mathbf{W}$ are +-varieties (*-varieties, respectively), then so is $\mathbf{V} \star \mathbf{W}$.

The l-variety corresponding to $\mathbf{V} \star \mathbf{W}$ has the following description. The result is an adaptation of a similar characterization of languages recognizable by semigroups in the wreath product of two semigroup varieties, see [12].

Theorem 3.3. *Suppose that* $\mathbf{V}$ *and* $\mathbf{W}$ *are q-varieties with corresponding l-varieties $\mathcal{V}$ and $\mathcal{W}$. Then for each Σ, the l-variety $\mathcal{V} \star \mathcal{W}$ corresponding to* $\mathbf{V} \star \mathbf{W}$ *contains exactly those languages in Σ^* that are finite unions of languages of the form $K \cap \mu^{-1}(L)$, where $K \in \Sigma^*\mathcal{V}$, $L \in \Delta^*\mathcal{W}$ and where $\mu : \Sigma^* \to \Delta^*$ is a sequential function induced by some state of a Mealy automaton extension of an automaton in* $\mathbf{V}$.

We may as well require that the same finite state literal sequential function μ appears in all terms of the finite union. Theorem 3.3 relies on Proposition 3.1 and the following fact.

Theorem 3.4. *For any q-varieties* $\mathbf{V}$ *and* $\mathbf{W}$ *and any Σ, an automaton Q is in* $\Sigma(\mathbf{V} \star \mathbf{W})$ *iff Q is a quotient of a subautomaton of a cascade product $R \times_\mu S$, where $R \in \Sigma\mathbf{V}$ and $S \in \Delta\mathbf{W}$ such that $R(\Delta, \mu)$ is a Mealy automaton extension of R.*

Proof. Let $\mathbf{K}$ denote the stream determined by those automata Q that can be constructed as quotients of subautomata of cascade products of automata $R \in \mathbf{V}$ and $S \in \mathbf{W}$. It is clear that $\mathbf{K} \subseteq \mathbf{V} \star \mathbf{W}$. Also, $\mathbf{K}$ is easily shown to be closed with respect to subautomata, quotients, direct products and renaming. Moreover, $\mathbf{K}$ clearly contains all discrete automata. Hence, $\mathbf{K}$ is closed with respect to disjoint sum. It follows that $\mathbf{V} \star \mathbf{W} \subseteq \mathbf{K}$. □

We say that a q-variety $\mathbf{V}$ is *closed with respect to the cascade product* if for any cascade product $Q \times_\mu R$ with $Q, R \in \mathbf{V}$, it holds that $Q \times_\mu R \in \mathbf{V}$. For example,

$\mathbf{N}, \mathbf{D}, \mathbf{A}$ are all closed with respect to the cascade product, cf. [5]. Moreover, for any set M of positive integers, $\mathbf{C}_M$ is closed with respect to the cascade product, as is *any* q-variety of autonomous automata.

We omit the straightforward proofs of the following facts.

Proposition 3.5. *Any q-variety contained in* $\mathbf{L}$ *is closed with respect to the cascade product. If* $\mathbf{V}$ *and* $\mathbf{W}$ *are q-varieties such that* $\mathbf{V}$ *is contained in* $\mathbf{L}$ *and* $\mathbf{W}$ *is closed with respect to the cascade product, then* $\mathbf{V} \star \mathbf{W}$ *is also closed with respect to the cascade product.*

Proposition 3.6. *Suppose that* $\{\mathbf{V}_i : i \in I\}$ *is a directed set of q-varieties and* $\mathbf{V} = \bigcup_{i \in I} \mathbf{V}_i$. *Then for any q-variety* $\mathbf{W}$, *we have* $\mathbf{V} \star \mathbf{W} = \bigcup_{i \in I} \mathbf{V}_i \star \mathbf{W}$. *Suppose that* $\mathcal{V}_i$ *denotes the l-variety corresponding to* $\mathbf{V}_i$, *for each* $i \in I$, *and suppose that* $\mathcal{V}$ *denotes the l-variety corresponding to* $\mathbf{V}$. *Then for any l-variety* $\mathcal{W}$, *it holds that* $\mathcal{V} \star \mathcal{W} = \bigcup_{i \in I} (\mathcal{V}_i \star \mathcal{W})$.

Thus, the $\star$ operation is continuous in its first argument. In a similar way, it is continuous in its second argument.

As an immediate application of Proposition 3.6 we have that

$$\mathbf{C}_M \star \mathbf{V} = \bigcup_{d \in (M]} \mathbf{C}_d \star \mathbf{V}$$

and

$$\mathcal{C}_M \star \mathcal{V} = \bigcup_{d \in (M]} \mathcal{C}_d \star \mathcal{V},$$

for all q-varieties $\mathbf{V}$ and l-varieties $\mathcal{V}$, and for all $M \subseteq \mathsf{Nat}$.

4 Varieties $\mathbf{C}_M \star \mathbf{V}$

In this section, we study q-varieties of the form $\mathbf{C}_d \star \mathbf{V}$ and $\mathbf{C}_M \star \mathbf{V}$, and the corresponding l-varieties $\mathcal{C}_d \star \mathcal{V}$ and $\mathcal{C}_M \star \mathcal{V}$.

Definition 4.1. *For any automaton* $Q = (Q, \Sigma, \cdot)$ *and integer* $d > 0$, *let* $Q^{(d)}$ *denote the automaton* $(Q, \Sigma^{(d)}, \cdot)$, *where* $\Sigma^{(d)}$ *consists of all* letters $\langle u \rangle$, *where* u *is any word of length* d *in* Σ^*, *i.e., any element of* Σ^d, *and where*

$$q \cdot \langle u \rangle = qu,$$

for all $q \in Q$ *and* $u \in \Sigma^d$.

Thus, $Q^{(d)}$ arises from Q by letting the words in Σ^* of length d be the input letters. For each $u \in \Sigma^d$, the function induced by $\langle u \rangle$ in $Q^{(d)}$ is the same as the function induced by u in the automaton Q. Besides $Q^{(d)}$, we will also use the automaton $Q_1^{(d)}$, which is the extension of $Q^{(d)}$ by a letter a_0 inducing the identity function $Q \to Q$. Thus, $Q_1^{(d)} = (Q, \Sigma^{(d)} \cup \{a_0\}, \cdot)$, where $q \cdot a_0 = q$, for all $q \in Q$, and where for all $q \in Q$ and $u \in \Sigma^d$, $q \cdot \langle u \rangle$ is defined as above. Note that the monoids $M(Q^{(d)})$ and $M(Q_1^{(d)})$ are both isomorphic to the submonoid $M_d(Q)$ of the monoid $M(Q)$ of automaton Q consisting of all functions $Q \to Q$ induced by those words in Σ^* whose length is a multiple of d.

Proposition 4.2. *Each automaton Q is a homomorphic image of a cascade product of an automaton which is a direct product of a counter of length d with a shift register, and the automaton $Q_1^{(d)}$.*

Proof. Suppose that $Q = (Q, \Sigma, \cdot)$, so that $Q_1^{(d)}$ is $(Q, \Sigma^{(d)} \cup \{a_0\}, \cdot)$ defined above.

Let C_d denote the counter of length d whose input alphabet is Σ and whose states are the integers in $[d]$, so that $i \cdot a = i + 1 \bmod d$, for all $i \in [d]$ and $a \in \Sigma$. Let D_{d-1} denote the shift register of length $d - 1$ over Σ. Thus the states of D_{d-1} are the words in Σ^{d-1}, and the transition is defined so that for each $u \in \Sigma^{d-1}$ and $a \in \Sigma$, state $u \cdot a$ is the suffix of ua of length $d - 1$. Define

$$\mu : ([d] \times \Sigma^{d-1}) \times \Sigma \quad \to \quad \Sigma^{(d)} \cup \{a_0\}$$

by

$$\mu((i,u),a) \quad = \quad \begin{cases} a_0 & \text{if } i \neq d-1 \\ \langle ua \rangle & \text{otherwise.} \end{cases}$$

We thus obtain the cascade product $Q' = (C_d \times D_{d-1}) \times_\mu Q_1^{(d)}$. We claim that there is a surjective homomorphism $h : Q' \to Q$. Indeed, for each state $((i,u),q)$ of Q', define

$$((i,u),q)h \quad = \quad qv,$$

where v denotes the suffix of u of length i. In particular, $((0,u),q)h = q$, for all $u \in \Sigma^{d-1}$ and $q \in Q$, so that h is surjective. We show that h is a homomorphism. Assume that $((i,u),q)$ is a state of Q' and $a \in \Sigma$. If $i \neq d - 1$ then

$$\begin{aligned} ((i,u),q)ah &= ((i+1,u'a),q)h \\ &= qva \\ &= (((i,u),q)h)a, \end{aligned}$$

where v denotes the suffix of u of length i and u' the suffix of u of length $d - 1$. When $i = d - 1$, we have

$$\begin{aligned} ((d-1,u),q)ah &= ((0,u'a),qua)h \\ &= qua \\ &= (((d-1,u),q)h)a, \end{aligned}$$

where u' is the same as above. □

Remark 4.3. *The same argument proves the following stronger version of Proposition 4.2. Suppose that R is a subautomaton of $Q_1^{(d)}$ such that for each $q \in Q$ there exists a state $r \in R$ and a word $u \in \Sigma^*$ with $|u| < d$ such that $ru = q$ holds in Q. Then automaton Q is a homomorphic image of a cascade product of an automaton which is the direct product of a counter of length d with a shift register, and the automaton R. Indeed, if we replace $Q_1^{(d)}$ with R in the above proof, the same argument works. The assumption that each $q \in Q$ be of the form ru with $r \in R$ and $|u| < d$ is needed to show that h is surjective.*

Recall that $\mathbf{D}$ denotes the $+$-variety of definite automata, and that $\mathcal{D}$ denotes the corresponding $+$-variety of definite languages. Note that for any $*$-variety $\mathbf{V}$ of automata and for any automaton Q and $d \geq 1$, we have $Q^{(d)} \in \mathbf{V}$ iff $Q_1^{(d)} \in \mathbf{V}$.

Corollary 4.4. *Suppose that $\mathbf{V}$ is a q-variety such that $\mathbf{D} \star \mathbf{V} \subseteq \mathbf{V}$. Then for any integer $d \geq 1$ and automaton Q, if $Q_1^{(d)} \in \mathbf{V}$ then $Q \in \mathbf{C}_d \star \mathbf{V}$.*

We now want to prove a certain converse of the above result.

Proposition 4.5. *Suppose that $\mathbf{V}$ is a $*$-variety of automata and $d \geq 1$. If $Q \in \mathbf{C}_d \star \mathbf{V}$, then $Q^{(d)}$, and thus $Q_1^{(d)}$, is in $\mathbf{V}$.*

Proof. First assume that Q is 1-generated, i.e., there exists a state q_0 in Q such that each state is accessible from q_0 by an input word. If $Q \in \mathbf{C}_d \star \mathbf{V}$ then, by Theorem 3.4, Q is a quotient of a subautomaton R' of a cascade product of an automaton C in $\mathbf{C}_d$ and an automaton R in $\mathbf{V}$. Since Q is 1-generated, without loss of generality we may assume that so is R'. But in that case C may be chosen to be 1-generated as well, so that C is a counter in $\mathbf{C}_d$ and is thus a quotient of a counter of length d. We conclude that Q is a homomorphic image, with respect to a homomorphism h, of a subautomaton $R' = (R', \Sigma, \cdot)$ of a cascade product $C_d \times_\mu R$, where $C_d = ([d], \Sigma, \cdot)$ is the counter of length d with $ia = i + 1 \bmod d$, for all $i \in [d]$ and $a \in \Sigma$, and $R = (R, \Delta, \cdot)$ is an automaton in $\mathbf{V}$. For each $i \in [d]$, let R_i denote the set of all states $r \in R$ such that $(i, r) \in R'$. It is clear that $R_i \neq \emptyset$. Moreover, let $h_i : R_i \to Q$ be defined by $r \mapsto h((i, r))$, for all $r \in R_i$. We turn each R_i into an automaton $R_i = (R_i, \Sigma^{(d)}, \cdot)$ with input letters in the set $\Sigma^{(d)}$. For each $r \in R_i$ and $u \in \Sigma^d$, let $r \cdot \langle u \rangle = r\mu_i(u)$, the image of r with respect to the word which is the image of u with respect to the sequential function induced by state i of the Mealy extension $C_d(\Delta, \mu)$. Since $\mathbf{V}$ is a $*$-variety and $R \in \mathbf{V}$, it follows that each R_i is in $\mathbf{V}$. Indeed, R_i can be constructed from R^* by renaming and taking subautomata. Also, each h_i is a homomorphism $R_i \to Q^{(d)}$, and since h is surjective, each state in Q appears as the image of some state in $\bigcup_{i \in [d]} R_i$. Thus, the disjoint sum of the R_i can be mapped homomorphically onto $Q^{(d)}$, proving that $Q^{(d)}$ is in $\mathbf{V}$ (since $\mathbf{V}$ is closed with respect to disjoint sum).

In the general case, Q is a quotient of the disjoint sum of its 1-generated subautomata $Q_1, \ldots, Q_n$. If $Q \in \mathbf{C}_d \star \mathbf{V}$ then each Q_i belongs to $\mathbf{C}_d \star \mathbf{V}$. Thus, by the above argument, we have $Q_i^{(d)} \in \mathbf{V}$, for each i. Since $\mathbf{V}$ is closed with respect to disjoint sum, it follows that the disjoint sum of the $Q_i^{(d)}$ is also in $\mathbf{V}$. But $Q^{(d)}$ is a quotient of this disjoint sum, so that $Q^{(d)} \in \mathbf{V}$. □

Call a q-variety $\mathbf{V}$ *decidable* if there is an algorithm to decide for any given automaton Q whether or not Q belongs to $\mathbf{V}$. Similarly, call an l-variety $\mathcal{V}$ decidable if there is an algorithm to decide whether or not a regular language (given by an automaton or a regular expression) belongs to $\mathcal{V}$. From Corollary 4.4 and Proposition 4.5 we have:

Theorem 4.6. *For any $*$-variety $\mathbf{V}$ of automata with $\mathbf{D} \star \mathbf{V} \subseteq \mathbf{V}$, and for any $d \geq 1$ and automaton Q, we have that $Q \in \mathbf{C}_d \star \mathbf{V}$ iff $Q^{(d)} \in \mathbf{V}$. Thus, if $\mathbf{V}$ is decidable, then so is $\mathbf{C}_d \star \mathbf{V}$.*

A first characterization of the languages in the variety $\mathcal{C}_d \star \mathcal{V}$, where $\mathcal{V}$ is any l-variety of languages, may be obtained from the wreath product principle. Let Σ denote an alphabet and consider the Σ-counter $C_d = ([d], \Sigma, \cdot)$ with $i \cdot a = i + 1 \bmod d$, for all $i \in [d]$ and $a \in \Sigma$. Consider the alphabet $[d] \times \Sigma$ and the identity function $\pi_d : [d] \times \Sigma \to [d] \times \Sigma$. Let σ_d denote the literal sequential function induced by the Mealy extension $C_d([d] \times \Sigma, \pi_d)$ in state 0. Then any literal sequential function $\sigma : \Sigma^* \to \Delta^*$ induced by a state of a Mealy extension of an automaton in $\mathbf{C}_d$ can be factorized as the composition of σ_d with a literal homomorphism $\tau : ([d] \times \Sigma)^* \to \Delta^*$. Thus, by the wreath product principle we get:

Proposition 4.7. *A language $L \subseteq \Sigma^*$ belongs to $\mathcal{C}_d \star \mathcal{V}$ iff L can be written as*

$$L \quad = \quad \bigcup_{i \in [d]} (\Sigma^d)^* \Sigma^i \cap \sigma_d^{-1}(K_i),$$

for some languages $K_i \in ([d] \times \Sigma)^* \mathcal{V}$, $i \in [d]$.

When $\mathcal{V}$ corresponds to a $*$-variety $\mathbf{V}$ with $\mathbf{D} \star \mathbf{V} \subseteq \mathbf{V}$, we can use Theorem 4.6 to derive an alternative characterization of the languages in $\mathcal{C}_d \star \mathcal{V}$.

Suppose that $L \subseteq \Sigma^*$ and $d \geq 1$. We define

$$L^{(d)} \quad = \quad \{\langle u_0 \rangle \ldots \langle u_{k-1} \rangle : u_0 \ldots u_{k-1} \in L, \ u_i \in \Sigma^d, \ i \in [k]\},$$

so that $L^{(d)} \subseteq (\Sigma^{(d)})^*$. Moreover, for each $u \in \Sigma^*$ with $|u| < d$, we define $L^{(d,u)} = (Lu^{-1})^{(d)}$. Thus, $L^{(d)}$ and each $L^{(d,u)}$ is a language in $(\Sigma^{(d)})^*$, moreover, $L^{(d)} = L^{(d,\epsilon)}$.

Theorem 4.8. *Suppose that $\mathbf{V}$ is a $*$-variety of automata with $\mathbf{D} \star \mathbf{V} \subseteq \mathbf{V}$, and suppose that $\mathcal{V}$ denotes the language variety corresponding to $\mathbf{V}$. Then for any integer $d \geq 1$ and language $L \subseteq \Sigma^*$, if $L \in \mathcal{C}_d \star \mathcal{V}$ then $L^{(d,u)} \in \mathcal{V}$, for all $u \in \Sigma^*$ with $|u| < d$. Moreover, if $L^{(d,u)} \in \mathcal{V}$, for all $u \in \Sigma^*$ with $|u| < d$, and if $\mathcal{V}$ is closed with respect to right (or left) concatenation with letters, then $L \in \mathcal{C}_d \star \mathcal{V}$.*

Proof. Suppose first that L is in $\Sigma^*(\mathcal{C}_d \star \mathcal{V})$. Then L can be recognized by an automaton Q in $\mathbf{C}_d \star \mathbf{V}$. By Theorem 4.6 we have that $Q^{(d)} \in \mathbf{V}$. But each of the languages $L^{(d,u)}$, where $u \in \Sigma^*$ with $|u| < d$ can be recognized by $Q^{(d)}$. For if L is recognized by $Q = (Q, \Sigma, \cdot)$ with initial state q_0 and final states F, then $L^{(d,u)}$

is recognized by $Q^{(d)}$ with initial state q_0 and final states $F_u = \{q \in Q : qu \in F\}$. Thus, each $L^{(d,u)}$ belongs to $\mathcal{V}$.

Suppose now that each $L^{(d,u)}$ belongs to $\mathcal{V}$, for any $u \in \Sigma^*$ with $|u| < d$, so that each $L^{(d,u)}$ can be recognized by some automaton Q_u in $\Sigma^{(d)}\mathbf{V}$. For each u, let $R_u = Q_u \times (\cup_{k \in [d]} \Sigma^k)$. We turn R_u into a Σ-automaton $(R_u, \Sigma, \cdot)$ by defining, for each $(q, v) \in R_u$ and $a \in \Sigma$,

$$(q,v)\cdot a \quad = \quad \begin{cases} (q, va) & \text{if } |v| < d-1 \\ q\langle va\rangle & \text{otherwise.} \end{cases}$$

Let $Q'_u = (q, \epsilon)$, $q \in Q_u$. Then Q'_u determines a subautomaton of $R_u^{(d)}$ which is isomorphic to Q_u. Moreover, $(q, v) = (q, \epsilon)v$, for each $(q, v) \in R_u$. Thus, by Remark 4.3 and the assumption $\mathbf{D} \star \mathbf{V} \subseteq \mathbf{V}$, it follows that R_u belongs to $\mathbf{C}_d \star \mathbf{V}$. Now for every u, the language $L_u = (Lu^{-1}) \cap (\Sigma^d)^*$ can be recognized by R_u, so that $L_u \in \mathcal{C}_d \star \mathcal{V}$. Since $L = \bigcup_{u \in \Sigma^*,\ |u| < d} L_u u$, it follows now that L is in $\mathcal{C}_d \star \mathcal{V}$. □

Corollary 4.9. *Under the assumption of Theorem 4.8, if $\mathcal{V}$ is decidable, then so is $\mathcal{C}_d \star \mathcal{V}$.*

Proof. This follows either from Theorem 4.8 or from Theorem 4.6. □

Corollary 4.10. *Suppose that $M \subseteq$ Nat and $\mathbf{V}$ is a q-variety with corresponding l-variety $\mathcal{V}$. Suppose that $\mathbf{D} \star \mathbf{V} \subseteq \mathbf{V}$ and that $\mathcal{V}$ is closed with respect to right concatenation by letters. An automaton Q is in $\mathbf{C}_M \star \mathbf{V}$ iff there is some $d \in (M]$ with $Q^{(d)} \in \mathbf{V}$. Moreover, a language $L \subseteq \Sigma^*$ is in $\mathcal{C}_M \star \mathcal{V}$ iff there is some $d \in (M]$ such that $L^{(d,u)} \in \mathcal{V}$ for each $u \in \Sigma^*$ with $|u| < d$.*

Remark 4.11. *Suppose that $\mathbf{V}$ is a q-variety with corresponding language variety $\mathcal{V}$. If $\mathbf{V} \star \mathbf{D} \subseteq \mathbf{V}$, then $\mathcal{V}$ is closed with respect to right concatenation by letters. To see this, suppose that $Q = (Q, \Sigma, \cdot)$ is an automaton in $\mathbf{V}$ that accepts the language L with initial state q_0 and set of final states F. Moreover, suppose that a_0 is a letter in Σ. We turn the set $R = \{e\} \cup (Q \times \Sigma)$ into a $(Q \times \Sigma)$-automaton by defining*

$$x(q,a) \quad = \quad (q,a),$$

for all $x \in R$ and $(q, a) \in Q \times \Sigma$. It is clear that R is a definite automaton, in fact a reset automaton. *Then let Q' be the Σ-automaton*

$$Q \times_\mu R,$$

where μ is the identity function $Q \times \Sigma \to Q \times \Sigma$. It is an easy matter to show that the language accepted by Q' with initial state (q_0, e) and final states $Q \times (F \times \{a_0\})$ is La_0.

In particular, if $\mathbf{V}$ contains $\mathbf{D}$ and is closed with respect to the cascade product, then the language variety corresponding to $\mathbf{V}$ is closed with respect to right concatenation by letters.

5 Degree of aperiodicity

Following [1, 16], we call an automaton $Q = (Q, \Sigma, \cdot)$ *quasi-aperiodic* if there is no n such that $M(Q)$ (or $S(Q)$) contains a nontrivial group *all of whose members can be induced by length n words.* (In the terminology of [7], Q is quasi-aperiodic if no nontrivial group divides $M(Q)$ in "equal lengths".)

It is clear that any aperiodic automaton is quasi-aperiodic. On the other hand, a counter of length > 1 is quasi-aperiodic, but not aperiodic. Let $\mathbf{QA}$ denote the stream of quasi-aperiodic automata. The following theorem is a rephrasing of a result due to Barrington, Compton, Straubing and Therien. In its original formulation, the theorem involved the wreath product instead of the cascade product.

Theorem 5.1. (Barrington et al. [1]) $\mathbf{QA} = \mathbf{C} \star \mathbf{A}$. *Thus* $\mathbf{QA}$ *is a q-variety.*

It is a well-known consequence of the Krohn-Rhodes theorem [5, 16] that $\mathbf{A} \star \mathbf{A} \subseteq \mathbf{A}$, in fact equality holds. Thus, by Proposition 3.5, $\mathbf{QA}$ is also closed with respect to the cascade product. Moreover, since $\mathbf{D} \subseteq \mathbf{A}$, we have that $\mathbf{D} \star \mathbf{A} \subseteq \mathbf{A}$. Thus, by Theorem 4.6 we have:

Corollary 5.2. *For any $d \geq 1$ and automaton Q, we have $Q \in \mathbf{C}_d \star \mathbf{A}$ iff $Q^{(d)} \in \mathbf{A}$.*

Corollary 5.3. *An automaton Q is quasi-aperiodic iff there is some integer $d \geq 1$ such that $Q^{(d)}$ is aperiodic.*

Proof. If Q is quasi-aperiodic, then by Theorem 5.1, Q is in $\mathbf{C} \star \mathbf{A}$. But since $\mathbf{C}$ is the union of the $\mathbf{C}_n$ where n is any positive integer, it follows that Q is in $\mathbf{C}_d \star \mathbf{A}$, for some $d \geq 1$. Thus, by Corollary 5.2, $Q^{(d)}$ is in $\mathbf{A}$, so that $Q^{(d)}$ is aperiodic.

Assume now that $Q^{(d)}$ is aperiodic, for some $d \geq 1$. Then, by Corollary 5.2 and Theorem 5.1, Q is in $\mathbf{C}_d \star \mathbf{A} \subseteq \mathbf{QA}$. □

Remark 5.4. *Of course, it is possible to prove Corollary 5.3 without using Theorem 5.1 and Corollary 5.2. Assume that $Q^{(d)}$ is aperiodic for some $d \geq 1$. Then it cannot be the case that for some n, the set of all functions in $M(Q)$ that can be induced by the length n words contains a nontrivial group G, since otherwise each element of G would be induced by a word of length dn, so that $Q^{(d)}$ would not be aperiodic. The other direction can be verified by following the argument given in the proof of Theorem 5.10.*

Proposition 5.5. *Suppose that Q is an automaton such that both $Q^{(m)}$ and $Q^{(n)}$ are aperiodic, where $m, n \geq 1$. If m and n are relative primes, then also Q is aperiodic.*

Proof. If Q is not aperiodic, then $M(Q)$ contains a cyclic subgroup $G = \{g_0, \ldots, g_{p-1}\}$ of prime order $p > 1$, where $g_0 = e$ denotes the unit. Unless $g_1^m = e$, it follows that each element of G can be induced by a word whose length is a multiple of m. (Indeed, if $g_1^m = g_i$, where $i \neq 0$, then g_i can be induced by a word whose length is a multiple of m. Since g_i is a generator element of G, the same holds for any other group element.) But since $Q^{(m)}$ is aperiodic, this is impossible.

We conclude that $g_1^m = e$. In the same way, $g_1^n = e$. But then p divides both m and n, a contradiction. □

Corollary 5.6. *Suppose that Q is an automaton such that both $Q^{(m)}$ and $Q^{(n)}$ are aperiodic. If d denotes the g.c.d. of m and n, then $Q^{(d)}$ is also aperiodic.*

Corollary 5.7. *An automaton Q is quasi-aperiodic iff there is a least integer $d \geq 1$ such that $Q^{(d)}$ is aperiodic. Moreover, for an integer $n \geq 1$ we have that $Q^{(n)}$ is aperiodic iff this integer d is a divisor of n.*

Definition 5.8. *The* degree of aperiodicity, *or* aperiodicity degree *of an automaton Q is the least integer d such that $Q^{(d)}$ is aperiodic, if such an integer exists. Otherwise the degree of aperiodicity of Q is ∞.*

Thus, by Corollary 5.7, the aperiodicity degree of Q is finite iff Q is quasi-aperiodic.

For any set M of positive integers, we let $\mathbf{QA}_M$ denote the stream of automata whose aperiodicity degree is finite and belongs to $(M]$. In particular, $\mathbf{A} = \mathbf{QA}_{\{1\}} = \mathbf{QA}_\emptyset$ and $\mathbf{QA} = \mathbf{QA}_{\mathsf{Nat}}$. We also denote $\mathbf{QA}_d = \mathbf{QA}_{\{d\}}$, for each $d \geq 1$.

Theorem 5.9. *Suppose that M is a set of positive integers. Then $\mathbf{QA}_M = \mathbf{C}_M \star \mathbf{A}$. Thus, $\mathbf{QA}_M$ is a q-variety closed with respect to the cascade product.*

Proof. Suppose that the aperiodicity degree d of Q is finite and is contained in $(M]$. Then $Q^{(d)}$ is aperiodic, so that $Q \in \mathbf{C}_d \star \mathbf{A}$, by Corollary 5.2. But $\mathbf{C}_d \subseteq \mathbf{C}_M$, thus $Q \in \mathbf{C}_M \star \mathbf{A}$.

Suppose now that $Q \in \mathbf{C}_M \star \mathbf{A}$. Then since $\mathbf{C}_M$ is the union of all varieties $\mathbf{C}_d$, where d belongs to $(M]$, it follows by Proposition 3.6 that $Q \in \mathbf{C}_d \star \mathbf{A}$, for some such d. Thus, by Corollary 5.2, $Q^{(d)}$ is aperiodic. But then the aperiodicity degree of Q divides d, so that it also belongs to $(M]$. □

Theorem 5.10. *There exists an algorithm to compute the aperiodicity degree of an automaton.*

Proof. Barrington, Compton, Straubing and Therien showed in [1] how to decide for an automaton whether or not it belongs to $\mathbf{QA}$. (See also [7].) Our result follows by a slight modification of their argument. Given $Q = (Q, \Sigma, \cdot)$, let $M_{=m}(Q)$ denote the set of all functions $Q \to Q$ induced by the words in Σ^m, for each $m \geq 0$. Then, compute the sets $M_{=1}(Q), M_{=2}(Q), \ldots$ until a repetition occurs, i.e., until $M_{=m}(Q) = M_{=n}(Q)$, for some $m < n$. Then also $M_{=m+r}(Q) = M_{=n+r}(Q)$, for all $r \geq 1$. In particular, we have $M_{=d}(Q) = M_{=d+n-m}(Q)$ for some $m \leq d < n$ such that $n - m$ divides d. Thus, $M_{=d}(Q) = M_{=2d}(Q)$, showing that $M_{=d}(Q)$ is a subsemigroup of $M(Q)$. In fact, $M_{=d}(Q)$ is the semigroup of all functions inducible by words whose length is a positive multiple of d. If Q is quasi-aperiodic, then, by definition, this semigroup contains no nontrivial group. It follows that $Q^{(d)}$ is aperiodic. Thus, to compute the aperiodicity degree of Q it suffices to find the least divisor d' of d such that $Q^{(d')}$ is aperiodic. On the other hand, if $M_{=d}$ does contain a nontrivial group, then Q is not quasi-aperiodic and thus its aperiodicity degree is ∞. □

Corollary 5.11. *Suppose that $(M]$ is a recursive set. Then $\mathbf{QA}_M$ is decidable.*

Remark 5.12. *The opposite direction is immediate: if $\mathbf{QA}_M$ is decidable then $(M]$ is recursive.*

Since $\mathbf{QA}_M$ is a q-variety, there is a corresponding l-variety that we denote by $\mathcal{QA}_M$. We also denote $\mathcal{QA}_{\{d\}}$ by $\mathcal{QA}_d$. In particular, $\mathcal{QA}_{\{1\}} = \mathcal{QA}_1 = \mathcal{A}$ and $\mathcal{QA}_{\mathsf{Nat}} = \mathcal{QA}$, the l-variety corresponding to $\mathbf{QA}$. Since $\mathbf{QA}_M$ is the union of the varieties $\mathbf{QA}_d$, where d is any element of the division ideal $(M]$ generated by M, also $\mathcal{QA}_M$ is the union of the $\mathcal{QA}_d$, where d is any member $(M]$. The languages belonging to $\mathcal{A}$ have been characterized by Schützenberger as the *star-free languages.*

Theorem 5.13. (Schützenberger [14]) *A language $L \subseteq \Sigma^*$ belongs to $\mathcal{A}$ iff L can be constructed from the finite subsets of Σ^* by the operations of set union, complement and concatenation.*

A similar characterization of $\mathcal{QA}$ was obtained in [1].

Theorem 5.14. (Barrington, Compton, Straubing and Therien [1]) *A language $L \subseteq \Sigma^*$ belongs to $\mathcal{QA}$ iff L can be constructed from the finite languages in Σ^* and the languages $(\Sigma^d)^*$, $d \geq 1$, by the operations of union, complement and concatenation.*

In the rest of this section we prove a refinement of these results.

Theorem 5.15. *Let M denote any subset of the set of positive integers. A language $L \subseteq \Sigma^*$ belongs to $\mathcal{QA}_M$ iff L can be constructed from the finite languages in Σ^* and the languages $(\Sigma^m)^*$, where $m \in M$, by the operations of union, complement and concatenation.*

In our argument, we will make use of the following characterization of $\mathcal{QA}_d$, which is an immediate consequence of Theorem 4.8 and the fact that $\mathcal{A}$ is closed with respect to right concatenation by letters (in fact, by Schützenberger's theorem, $\mathcal{A}$ is closed with respect to concatenation).

Corollary 5.16. *For any integer $d \geq 1$ and language $L \subseteq \Sigma^*$, if $L \in \mathcal{QA}_d$ then $L^{(d,u)} \in \mathcal{A}$, for all $u \in \Sigma^*$ with $|u| < d$. Moreover, if $L^{(d,u)} \in \mathcal{A}$, for all $u \in \Sigma^*$ with $|u| < d$, then $L \in \mathcal{QA}_d$.*

Proof of Theorem 5.15. First note that the language $(\Sigma^d)^*$, where d is any member of the division ideal generated by M can be constructed from the finite languages and the languages $(\Sigma^m)^*$, $m \in M$ by the operations of union, complement, and concatenation. This follows from the following two facts. If m_1 and m_2 are positive integers and m denotes their least common multiple (l.c.m.), then $(\Sigma^m)^* = (\Sigma^{m_1})^* \cap (\Sigma^{m_2})^*$. Moreover, if d is a divisor of m, then for some finite F, $(\Sigma^d)^* = (\Sigma^m)^* F$. Thus, since $\mathcal{QA}_M = \bigcup_{d \in (M]} \mathcal{QA}_d$, in the rest of the argument we may assume that M is itself a division ideal.

Suppose first that $L \in QA_M$. Since QA_M is the union of the QA_m with $m \in M$, there exists an integer $d \in M$ with $L \in QA_d$. Thus, by Corollary 5.16, all the languages $L^{(d,u)}$, $u \in \Sigma^*$, $|u| < d$ are in $\mathcal{A}$. By Schützenberger's theorem, Theorem 5.13, it follows that each $L^{(d,u)}$ with $u \in \Sigma^*$, $|u| < d$ can be constructed from the finite languages in $(\Sigma^{(d)})^*$ by using the operations of union, complement and concatenation. Hence, each language $K_u = Lu^{-1} \cap (\Sigma^d)^*$, where $u \in \Sigma^*$ with $|u| < d$ can be constructed from the finite languages in Σ^* and the language $(\Sigma^d)^*$ by the operations of union, complement and concatenation. (Take complement relatively to $(\Sigma^d)^*$.) Since $L = \bigcup_{u\in\Sigma^*,\ |u|<d} K_u u$, the same holds for L.

Suppose now that L can be constructed from the finite subsets of Σ^* and the languages $(\Sigma^m)^*$, where $m \in M$ by the operations of union, complement and concatenation. Let d denote the l.c.m. of those integers m for which $(\Sigma^m)^*$ is used in the construction of L. If we can show that $L^{(d,v)}$ belongs to $\mathcal{A}$, for each $v \in \Sigma^*$ with $|v| < d$, then it follows by Corollary 5.16 that $L \in QA_d$, and thus that $L \in QA_M$. We will show that for each $u, v \in \Sigma^*$ with $|u|, |v| < d$, the language in $(\Sigma^{(d)})^*$

$$L^{(d,u,v)} = \{\langle x_0\rangle \dots \langle x_{k-1}\rangle : k \geq 0,\ ux_0 \dots x_{k-1}v \in L\}$$

is in $\mathcal{A}$. Now this follows by a straightforward induction argument using Schützenberger's theorem, Theorem 5.13, and the following facts. Let $u, v \in \Sigma^*$ with $|u|, |v| < d$, and let $L, L_1, L_2 \subseteq \Sigma^*$.

1. If L is finite, then so is $L^{(d,u,v)}$.
2. $(L_1 \cup L_2)^{(d,u,v)} = L_1^{(d,u,v)} \cup L_2^{(d,u,v)}$.
3. $(L^c)^{(d,u,v)} = (L^{(d,u,v)})^c$.
4. If the length of each word in L_1 is at least $|u|$ and the length of each word in L_2 is at least $|v|$, then $(L_1L_2)^{(d,u,v)} = \bigcup_{|wz|=d} L_1^{(d,u,w)} \langle wz\rangle L_2^{(d,z,v)}$.
5. If the length of each word in L_1 is less than $|u|$ and the length of each word in L_2 is at least $|v|$, then $(L_1L_2)^{(d,u,v)} = \bigcup_{wz=u,\ w\in L_1} L_2^{(d,z,v)}$.
6. If the length of each word in L_1 is at least $|u|$ and the length of each word in L_2 is less than $|v|$, then $(L_1L_2)^{(d,u,v)} = \bigcup_{zw=v,\ w\in L_2} L_1^{(d,u,z)}$.
7. If the length of each word in L_1 is less than $|u|$ and the length of each word in L_2 is less than $|v|$, then $(L_1L_2)^{(d,u,v)}$ is finite.

□

Corollary 5.17. *Suppose that $(M]$ is a recursive set. Then there exists an algorithm to decide for a regular language $L \subseteq \Sigma^*$ whether or not L can be constructed from the finite languages and the languages $(\Sigma^m)^*$ with $m \in M$ by the operations of union, complement and concatenation.*

Remark 5.18. *The converse of the above corollary is immediate. If there exists an algorithm to decide for a regular language $L \subseteq \Sigma^*$ whether or not L can be constructed from the finite languages and the languages $(\Sigma^m)^*$ with $m \in M$ by the operations of union, complement and concatenation, then $(M]$ is a recursive set.*

Remark 5.19. *By the first part of the proof of Theorem 5.15, it follows that a language $L \subseteq \Sigma^*$ is in $\mathcal{Q}A_M$ iff L is a finite union*

$$L = \bigcup_{u \in \Sigma^*,\ |u|<m} L_u \cap (\Sigma^m)^* u,$$

where each L_u is in $\mathcal{A}$ and $m \in (M]$.

6 First-order logic

The expressive power of first-order logic on words with a unary predicate corresponding to each letter of the alphabet and $<$ as the only numerical predicate was characterized by McNaughton and Papert [11]. We let $\mathbf{FO}[<]$ denote this logic. Thus, for any fixed alphabet Σ, the *atomic formulas* of $\mathbf{FO}[<]$ are the propositions $P_a(x)$ and $x < y$, where a is any letter of Σ and x and y are variables. *Formulas* can be constructed from the atomic formulas by the boolean connectives $\vee$ and $\neg$, denoting disjunction and negation, and existential quantification. The other boolean connectives and universal quantification can be introduced as abbreviations. *Free and bound variables* are defined as usual. We may assume that no variable is bound two or more times in a formula, or in a finite set of formulas, and that any free variable is different from any bound variable. Below we will denote *syntactic equality* by $\equiv$.

Suppose that φ is a formula with free variables in X, and suppose that $w \in \Sigma^*$ and $\lambda : X \to [|w|]$, i.e., λ maps variables in X to "positions" in w. We say that (w, λ) *satisfies* φ, denoted $(w, \lambda) \models \varphi$, if

- $\varphi \equiv P_a(x)$ and the letter in w at position $x\lambda$ is a, or
- $\varphi \equiv x < y$ and $x\lambda < y\lambda$, or
- $\varphi \equiv \varphi_1 \vee \varphi_2$ and $(w, \lambda) \models \varphi_1$ or $(w, \lambda) \models \varphi_2$, or
- $\varphi \equiv \neg\psi$ and $(w, \lambda) \not\models \psi$, or
- $\varphi \equiv (\exists x)\psi$ and there exists a function $\lambda' : X \cup \{x\} \to [|w|]$ which agrees with λ on X such that $(w, \lambda') \models \psi$. (Here, by our conventions, we may assume without loss of generality that $x \notin X$.)

When X is empty, so that φ is a *sentence*, i.e., φ has no free variables, we write $w \models \varphi$ and call the set $\{w \in \Sigma^* : w \models \varphi\}$ the *language defined by* φ. Moreover, we say that a language $L \subseteq \Sigma^*$ is *definable in* $\mathbf{FO}[<]$ if there is a sentence φ which defines L.

As before, we let $\mathbf{A}$ denote the $*$-variety of aperiodic automata, and let $\mathcal{A}$ denote the corresponding $*$-variety of languages.

Theorem 6.1. (McNaughton and Papert [11]) *A language $L \subseteq \Sigma^*$ is definable in* $\mathbf{FO}[<]$ *iff $L \in \Sigma^*\mathcal{A}$.*

We refer the reader to [11], and in particular to [16], for detailed proofs of Theorem 6.1.

Subsequently, Barrington, Compton, Straubing and Therien [1] considered the extension of first-order logic by atomic propositions of the form $C_d^r(x)$, $d \geq 1$, $r \in [d]$ meaning that position x in the word satisfies $x \equiv r \bmod d$. Thus, using the above notations, $(w, \lambda) \models C_d^r(x)$ if and only if $x\lambda$ is congruent to r mod d. Since this logic is equivalent to the extension of **FO**$[<]$ by all regular numerical predicates, see [15], we denote it by **FO[R]**. As before, let $\mathcal{QA}$ denote the l-variety corresponding to the q-variety **QA** of quasi-aperiodic automata.

Theorem 6.2. (Barrington et al. [1]) *A language $L \subseteq \Sigma^*$ is definable in* **FO[R]** *iff $L \in \Sigma^*\mathcal{QA}$.*

For an integer $d \geq 1$, let **FO**$[d]$ denote the fragment of **FO[R]** where only atomic propositions associated to the letters of the alphabet and propositions of the form $x < y$ and $C_d^r(x)$ are allowed. (It would be sufficient to allow only $x < y$ and $C_d^0(x)$.) Moreover, for a set M of the positive integers, let **FO**$[M]$ denote the union of the **FO**$[d]$ with $d \in M$. Thus, **FO[R]** = **FO**[Nat] and **FO**$[<]$ = **FO**$[\emptyset]$.

Below we will write $x \leq y$ as an abbreviation for $\neg(y < x)$, $x = y + 1$ for $x < y \wedge \neg(\exists z)(x < z \wedge z < y)$, $\mathsf{Last}(x)$ for $(\forall y)(y \leq x)$, True for $\varphi \vee \neg\varphi$, where φ is a fixed sentence, and False for $\neg$True.

Proposition 6.3. *A language $L \subseteq \Sigma^*$ is definable in* **FO**$[M]$ *iff L is definable in* **FO**$[(M)]$.

Proof. This follows by the following two observations.

1. If d is a divisor of m, say $dk = m$, then $C_d^0(x)$ can be expressed as $C_m^0(x) \vee C_m^d(x) \vee \ldots C_m^{d(k-1)}(x)$. Moreover, for every $r \in [d-1]$, $C_m^{r+1}(x)$ can be expressed by $(\exists y)(x = y + 1 \wedge C_m^r(y))$.
2. If $m_1, m_2 \geq 1$ and m denotes the l.c.m. of m_1 and m_2, then $C_m^0(x)$ can be expressed as $C_{m_1}^0(x) \wedge C_{m_2}^0(x)$.

□

By our previous results we can prove the following common extension of Theorems 6.1 and 6.2.

Theorem 6.4. *Suppose that M is any set of the positive integers. Then a language $L \subseteq \Sigma^*$ is definable in* **FO**$[M]$ *iff $L \in \Sigma^*\mathcal{QA}_M$.*

The proof of Theorem 6.4 will be completed at the end of the section.

Proposition 6.5. *Suppose that $L \subseteq \Sigma^*$ and $d \geq 1$. If $L^{(d)}$ is definable in* **FO**$[<]$, *then $L \cap (\Sigma^d)^*$ is definable in* **FO**$[d]$.

Proof. First we prove that for all $\varphi \in$ **FO**$[<]$ with free variables in X there exists some $\varphi' \in$ **FO**$[d]$ with free variables in X such that for all $w \in (\Sigma^{(d)})^*$ and $\lambda : X \to [|w|]$,

$$(w, \lambda) \models \varphi \quad \textit{iff} \quad (wh, \kappa) \models \varphi',$$

where h denotes the homomorphism $(\Sigma^{(d)})^* \to \Sigma^*$ defined by $\langle u \rangle \mapsto u$, for all $u \in \Sigma^d$, and where $x\kappa = d(x\lambda)$, the product of the integers d and $x\lambda$, for all $x \in X$. We prove this claim by induction on the structure of φ.

- $\varphi \equiv P_{\langle u \rangle}(x)$, where $u = a_0 \dots a_{d-1}$. Then, writing x_0 for x, we define

$$\varphi' \quad \equiv \quad (\exists x_1)\dots(\exists x_{d-1}) \left[\bigwedge_{j=0}^{d-2} x_{j+1} = x_j + 1 \wedge \bigwedge_{j=0}^{d-1} P_{a_j}(x_j) \right].$$

- $\varphi \equiv x < y$. Then $\varphi' \equiv x < y$.
- $\varphi \equiv \varphi_1 \vee \varphi_2$. Then $\varphi' \equiv \varphi_1' \vee \varphi_2'$.
- $\varphi \equiv \neg\varphi_1$. Then $\varphi' \equiv \neg\varphi_1'$.
- $\varphi \equiv (\exists x)\varphi_1$. Then $\varphi' \equiv (\exists x)(C_d^0(x) \wedge \varphi_1')$.

We now complete the proof of Proposition 6.5. Suppose that $L^{(d)}$ is defined by sentence φ in **FO**$[<]$. Then $L \cap (\Sigma^d)^*$ is defined by $\varphi' \wedge (\forall x)(\mathsf{Last}(x) \to C_d^{d-1}(x))$. □

Corollary 6.6. *Suppose that $L \subseteq \Sigma^*$ and $d \geq 1$. If $L^{(d,u)}$ is definable in* **FO**$[<]$*, for all $u \in \Sigma^*$ with $|u| < d$, then L is definable in* **FO**$[d]$.

Proof. For each $u \in \Sigma^*$ with $|u| < d$ we have that $L^{(d,u)} = (Lu^{-1})^{(d)}$. By Proposition 6.5, it follows that if $L^{(d,u)}$ is definable in **FO**$[<]$, then $K_u = Lu^{-1} \cap (\Sigma^d)^*$ is definable in **FO**$[d]$, for each $u \in \Sigma^*$, $|u| < d$. But then, using the formula $L = \bigcup_{u \in \Sigma^*,\ |u|<d} K_u u$, it follows easily that L is definable in **FO**$[d]$. Indeed, if K_u is defined by φ_u, where $u = a_0 \dots a_{n-1} \in \Sigma^*$ with $|u| = n < d$, then $K_u u$ is defined by the formula ψ_u

$$(\exists x_0)\dots(\exists x_{n-1}) \left[\bigwedge_{i=0}^{n-2} x_{i+1} = x_i + 1 \wedge \bigwedge_{i=0}^{n-1} P_{a_i}(x_i) \wedge \mathsf{Last}(x_{n-1}) \wedge \varphi_u[< x_0] \right],$$

where $\varphi_u[< x_0]$ is the *relativization* of φ_u defined in the usual manner, cf. [16]. (If $n = 0$, so that there is no x_0, by this formula we mean φ_u.) Finally, L is defined by $\bigvee_{u \in \Sigma^*,\ |u|<d} \psi_u$. □

Proposition 6.7. *If $L \subseteq \Sigma^*$ is definable in* **FO**$[d]$*, then $L^{(d)}$ is definable in* **FO**$[<]$.

Proof. We prove the following claim. For all φ in **FO**$[d]$ with free variables in X and for all functions $\rho : X \to [d]$ there exists a formula $\varphi'_\rho \in$ **FO**$[<]$ with free variables in X such that for all words $w \in (\Sigma^{(d)})^*$ and functions $\lambda : X \to [|w|]$,

$$(w, \lambda) \models \varphi'_\rho \quad \textit{iff} \quad (wh, \kappa_\rho) \models \varphi,$$

where h denotes the homomorphism $(\Sigma^{(d)})^* \to \Sigma^*$ given by $\langle u \rangle \mapsto u$, for all $u \in \Sigma^d$, and where $x\kappa_\rho = (x\lambda)d + x\rho$, for all $x \in X$. We prove this claim by induction on the structure of φ.

- $\varphi \equiv P_a(x)$. Then φ'_ρ is the disjunction of all of the $P_{\langle u \rangle}(x)$ such that the letter of u on the $(x\rho)$th position is a.
- $\varphi \equiv x < y$. Then

$$\varphi'_\rho \equiv \begin{cases} x < y & \text{if } x\rho \geq y\rho \\ x \leq y & \text{if } x\rho < y\rho. \end{cases}$$

- $\varphi \equiv C_d^r(x)$. Then

$$\varphi'_\rho \equiv \begin{cases} \mathsf{True} & \text{if } x\rho = r \\ \mathsf{False} & \text{if } x\rho \neq r. \end{cases}$$

- $\varphi \equiv \varphi_1 \vee \varphi_2$. Then $\varphi'_\rho = (\varphi'_1)_\rho \vee (\varphi'_2)_\rho$..
- $\varphi \equiv \neg\psi$. Then $\varphi'_\rho = \neg\psi'_\rho$.
- $\varphi \equiv (\exists x)\psi$. Here we may assume that x is not in the set X. For each $i \in [d]$, let $\rho[x \mapsto i]$ denote that function $X \cup \{x\} \to [d]$ which agrees with ρ on X and such that $x\rho = i$. Then we define

$$\varphi'_\rho \equiv (\exists x) \bigvee_{i \in [d]} \psi'_{\rho[x \mapsto i]}.$$

We now complete the proof of Proposition 6.7. Suppose that $L \subseteq \Sigma^*$ is defined by the sentence φ in $\mathbf{FO}[d]$. Let φ' be the corresponding sentence of $\mathbf{FO}[<]$ defined above. Then for all $w \in (\Sigma^{(d)})^*$,

$$w \models \varphi' \quad \textit{iff} \quad wh \models \varphi.$$

(Note that ρ is the empty function.) Thus, φ' defines $L^{(d)}$. □

Corollary 6.8. *If $L \subseteq \Sigma^*$ is definable in* $\mathbf{FO}[d]$, *then for each $u \in \Sigma^*$ with $|u| < d$, $L^{(d,u)}$ is definable in* $\mathbf{FO}[<]$.

Proof. Use the fact that $L^{(d,u)} = (Lu^{-1})^{(d)}$ and that if L is definable in $\mathbf{FO}[d]$, then so is Lu^{-1}. □

We are now in the position to complete the proof of Theorem 6.4.

Proof of Theorem 6.4. By Corollaries 6.6 and 6.8, a language $L \subseteq \Sigma^*$ is definable in $\mathbf{FO}[d]$ iff $L^{(d,u)}$ is definable in $\mathbf{FO}[<]$, for each $u \in \Sigma^*$ with $|u| < d$. Thus, by the theorem of McNaughton and Papert, Theorem 6.1, and by Corollary 5.16, L is definable in $\mathbf{FO}[d]$ iff $L \in \mathcal{QA}_d$. Since a language is definable in $\mathbf{FO}[M]$ iff it is definable in $\mathbf{FO}[d]$, for some d in the division ideal generated by M, and since $\mathcal{QA}_M$ is the union of the $\mathcal{QA}_d$ where d is any integer in the division ideal generated by M, the result follows. □

Corollary 6.9. *Suppose that $(M]$ is a recursive set. Then it is decidable for a regular language L whether or not L can be defined in* $\mathbf{FO}[M]$.

Again, the converse direction holds obviously.

7 Temporal logic

The language **LTL** of Linear (Propositional) Temporal Logic [13] over an alphabet Σ has, as atomic formulas (or atomic propositions), the propositional constants p_a associated with the letters $a \in \Sigma$. Formulas can be constructed from the atomic formulas by the boolean connectives $\vee$ and $\neg$, and the modalities X (next) and U (until). Other boolean connectives may be introduced as usual. Suppose that $u \in \Sigma^*$ and that φ is a formula. We say that u satisfies φ, denoted $u \models \varphi$, if

1. $\varphi \equiv p_a$ and $u = av$, for some $a \in \Sigma$ and $v \in \Sigma^*$, or
2. $\varphi \equiv \varphi_1 \vee \varphi_2$, for some φ_1 and φ_2, and $u \models \varphi_1$ or $u \models \varphi_2$, or
3. $\varphi \equiv \neg\psi$, for some formula ψ, and it is not the case that $u \models \psi$, or
4. $\varphi \equiv X\psi$, for some ψ, and u is of the form av with $a \in \Sigma$ and $v \in \Sigma^*$ such that $v \models \psi$, or
5. $\varphi \equiv \varphi_1 U \varphi_2$, for some φ_1 and φ_2, and there exist $v, w \in \Sigma^*$ such that $u = vw$, $w \models \varphi_2$, moreover, $z \models \varphi_1$ for all suffixes z of u properly including w.

In this section we study the extension of **LTL** by a sort of modular counting which is different from the one considered in [2].

Suppose that $M \subseteq \mathsf{Nat}$. For an alphabet Σ, the atomic formulas of **LTL**$[M]$ are those of **LTL** together with an additional propositional constant $\mathsf{lg}_{d,r}$, for each $d \in M$ and $r \in [d]$. Formulas are constructed from the atomic formulas as above, so that if φ and ψ are formulas, then so are $\varphi \vee \psi$, $\neg\varphi$, $X\varphi$ and $\varphi U \psi$. For all $d \in M$ and $r \in [d]$, we define $u \models \mathsf{lg}_{d,r}$ iff the length of u is congruent to r modulo d. The semantics of the other constructs of **LTL**$[M]$ are defined as above. When $M = \{d\}$, for some positive integer d, we write just **LTL**$[d]$ for **LTL**$[M]$. Note that **LTL**$[\emptyset]$ is just **LTL**.

We say that a language $L \subseteq \Sigma^*$ is definable in **LTL**$[M]$ if there is a formula φ of **LTL**$[M]$ (with propositional constants corresponding to the letters of Σ) such that $L = L_\varphi = \{u \in \Sigma^* : u \models \varphi\}$.

Example 7.1. For any $m, n > 0$ and $u \in \Sigma^*$, we have that $u \models \mathsf{lg}_{m,0}$ and $u \models \mathsf{lg}_{n,0}$ iff $u \models \mathsf{lg}_{k,0}$, where k denotes the least common multiple of m and n. Moreover, $u \models \mathsf{lg}_{m,r}$, for $r \in [m]$, iff $u \models X^r \mathsf{lg}_{m,0}$, where X^r is $X \dots X$ with X appearing r times. Also, if n divides m, then $u \models \mathsf{lg}_{n,0}$ iff $u \models \bigvee_{i \in [m/n]} \mathsf{lg}_{m,in}$.

By the above example, we have that **LTL**$[M]$ is exactly as expressive as **LTL**$[(M]]$, i.e., a language is definable in **LTL**$[M]$ iff it is definable in **LTL**$[(M]]$. Moreover, when M is not empty, then a language is definable in **LTL**$[M]$ iff it is definable in **LTL**$[d]$, for some $d \in (M]$.

The logic **LTL**$[M]$ allows for several counting versions of the until modality. For any formulas φ and ψ, and for any $d \in M$ and $r \in [d]$, define $\varphi U^{(d,0)} \psi$ to be the formula

$$\bigvee_{i \in [d]} [\mathsf{lg}_{d,i} \wedge (\varphi U (\psi \wedge \mathsf{lg}_{d,i}))],$$

and define $\varphi U^{(d,r)}\psi$, $r > 0$ as

$$\varphi \wedge X\varphi \wedge \ldots \wedge X^{r-1}\varphi \wedge X^r(\varphi U^{(d,0)}\psi).$$

Then we have $u \models \varphi U^{(d,r)}\psi$ iff u has a decomposition $u = vw$ such that $w \models \psi$, $|v|$ is congruent to r modulo d, moreover, for all x, z with $xz = u$ such that w is a proper suffix of z, it holds that $z \models \varphi$.

A second counting version of the until modality can now be defined as follows. For all φ, ψ and d, r as before, let $\varphi U_1^{(d,0)}\psi$ be the formula

$$\bigvee_{i\in[d]} [\mathsf{lg}_{d,i} \wedge (\neg\mathsf{lg}_{d,i} \vee \varphi)U^{(d,0)}\psi].$$

Moreover, when $r > 0$, let $\varphi U_1^{(d,r)}\psi$ be the formula

$$X^r(\varphi U_1^{(d,0)}\psi).$$

We now have $u \models \varphi U_1^{(d,r)}\psi$, for u a word in Σ^*, iff u has a decomposition $u = vw$ such that $w \models \psi$, $|v|$ is congruent to r modulo d, moreover, for all x, z with $xz = u$ such that w is a proper suffix of z and $|x|$ is congruent to r modulo d, it holds that $z \models \varphi$.

A last version of until involves several formulas. Suppose, as before, that $d \in M$, and suppose that $\varphi_0, \ldots, \varphi_{d-1}, \psi$ are formulas of **LTL**$[M]$. We define $(\varphi_0, \ldots, \varphi_{d-1})U_2^{(d,0)}\psi$ as the formula

$$\bigvee_{i\in[d]} \left\{ \mathsf{lg}_{d,i} \wedge \left[\bigwedge_{j,k\in[d],\ j-k\equiv i \bmod d} (\neg\mathsf{lg}_{d,j} \vee \varphi_k) \right] U^{(d,i)}\psi) \right\}$$

Thus, for all words $u \in \Sigma^*$, we have $u \models (\varphi_0, \ldots, \varphi_{d-1})U_2^{(d,0)}\psi$ iff u has a decomposition $u = vw$ such that $w \models \psi$, $|v|$ is congruent to r modulo 0, moreover, for all x, z and $i \in [d]$ with $xz = u$ such that w is a proper suffix of z and $|x|$ is congruent to i modulo d, it holds that $z \models \varphi_i$. The modalities $U_2^{(d,r)}$ with $r \in [d]$, $r \neq 0$, which have a similar semantics, can be introduced in the obvious way. Of course, the propositional constants $\mathsf{lg}_{d,r}$ can in turn be defined using either version of until.

Remark 7.2. *The last version of the until modality shows that the extension of* **LTL** *by counting is a particular case of Wolper's extension of temporal logic by grammar (or finite automaton) operators, cf. [22, 21].*

We introduce several abbreviations. First, let $\mathsf{True} = p_a \vee \neg p_a$, where a is any letter in Σ, and let $\mathsf{False} = \neg\mathsf{True}$. Moreover, let End denote the formula $\bigwedge_{a\in\Sigma} \neg p_a$, so that for all $u \in \Sigma^*$, we have $u \models \mathsf{End}$ iff $u = \epsilon$. Finally, for any formula φ, let $\Diamond^{(d,r)}\varphi$ stand for $\mathsf{True}\, U^{(d,r)}\varphi$ and $\Box^{(d,r)}\varphi$ for $\neg\Diamond^{(d,r)}\neg\varphi$. The modalities $\Diamond$ and $\Box$ are defined as usual.

Example 7.3. Let $\Sigma = \{a, b\}$. If φ is the formula $\Diamond^{(2,0)}\mathsf{End}$, then L_φ, the language defined by φ is $(\Sigma^2)^*$. Moreover, if $\psi \equiv p_a U^{(2,0)}(\Box(p_b \vee \mathsf{End}))$, then L_ψ is the language $(a^2)^* b^*$.

In his thesis [10], Kamp proved that temporal logic with past and future modalities is *expressively complete* in the sense that it can express every first-order property of words. Subsequently, it has been shown in [8] that future (or past) modalities alone suffice. An algebraic proof of this result, based on the Krohn-Rhodes decomposition theorem for finite semigroups and automata [5, 16], was later given by Cohen, Perrin and Pin in [4]. See also Th. Wilke, [20].

Theorem 7.4. (Kamp [10], Gabbay et al. [8]) *A language $L \subseteq \Sigma^*$ is definable in* **LTL** *iff L is definable in* $\mathbf{FO}[<]$.

Hence, L is definable in **LTL** iff L is in $\mathcal{A}$. Our aim is to prove the following counting extension of Kamp's theorem.

Theorem 7.5. *For any set M of positive integers, a language $L \subseteq \Sigma^*$ is definable in* $\mathbf{LTL}[M]$ *iff L is definable in* $\mathbf{FO}[M]$.

In our proof of Theorem 7.5, we will use:

Proposition 7.6. *Suppose that $L \subseteq \Sigma^*$, $d \geq 1$ and $v \in \Sigma^*$ with $|v| < d$. If $L^{(d,v)}$ is definable in* **LTL**, *then $L \cap (\Sigma^d)^* v$ is definable in* $\mathbf{LTL}[d]$.

Proof. First we show that for every formula φ of **LTL** there is a formula φ' of $\mathbf{LTL}[d]$ such that for all words $w \in (\Sigma^{(d)})^*$ it holds that $w \models \varphi$ iff $(wh)v \models \varphi'$, where h denotes the homomorphism $(\Sigma^{(d)})^* \to \Sigma^*$ defined by $\langle w \rangle \mapsto w$, all $w \in \Sigma^d$. We construct φ' by induction.

- $\varphi \equiv p_{\langle u \rangle}$, where $u = a_0 \dots a_{d-1}$. Then
$$\varphi' \equiv p_{a_0} \wedge X p_{a_1} \wedge \dots \wedge X^{d-1} p_{a_{d-1}},$$
where $X^n \varphi$ is $X \dots X\varphi$ with X appearing n times.
- $\varphi \equiv \varphi_1 \vee \varphi_2$. Then $\varphi' \equiv \varphi_1' \vee \varphi_2'$.
- $\varphi \equiv \neg\psi$. Then $\varphi' \equiv \neg\psi'$.
- $\varphi \equiv X\psi$. Then $\varphi' \equiv X^d \psi'$.
- $\varphi \equiv \varphi_1 U \varphi_2$. Then $\varphi' \equiv \varphi_1' U_1^{(d,0)} \varphi_2'$.

Suppose now that $L^{(d,v)}$ is defined by φ. Then the formula
$$\varphi' \wedge \Diamond^{(d,0)}(p_{a_0} \wedge X p_{a_1} \wedge \dots \wedge X^{i-1} p_{a_{i-1}} \wedge X^i \mathsf{End})$$
defines $L \cap (\Sigma^d)^* v$, where $v = a_0 \dots a_{i-1}$ and φ' denotes the formula constructed above. □

Corollary 7.7. *Suppose that $L \subseteq \Sigma^*$ and $d \geq 1$. If $L^{(d,u)}$ is definable in* **LTL**, *for each $u \in \Sigma^*$ with $u \in \Sigma^*$, $|u| < d$, then L is definable in* $\mathbf{LTL}[d]$.

We are now ready to prove Theorem 7.5.

Proof of Theorem 7.5. It is well-known that temporal logic can be embedded in first order logic. Thus, any language definable in **LTL** is definable in **FO**$[<]$. The proof goes by formula induction, essentially by formalizing the definition of the semantics of **LTL** in first-order logic. It is easy to show in the same way that any language definable in **LTL**$[M]$ is definable in **FO**$[M]$.

Suppose now that L is definable in **FO**$[M]$. Then L is definable in **FO**$[d]$, for some $d \in (M]$. Thus, by Corollary 6.8, $L^{(d,u)}$ is definable in **FO**$[<]$, for each $u \in \Sigma^*$ with $|u| < d$. Thus, by Theorem 7.4 and Corollary 7.7, L is definable in **LTL**$[d]$, hence in **LTL**$[(M]]$ and in **LTL**$[M]$. □

8 Summary and future results

Our main results can be summarized in a single statement that establishes the equivalence between four descriptions of the same class of languages.

Corollary 8.1. *Suppose that M is a set of the positive integers. The following conditions are equivalent for a language $L \subseteq \Sigma^*$:*

1. *L can be constructed from the finite subsets of Σ^* and the languages $(\Sigma^m)^*$, where $m \in M$, by the Boolean operations and concatenation.*
2. *L can be defined by a formula of* **LTL**$[M]$.
3. *L can be defined by a formula of* **FO**$[M]$.
4. *L can be accepted by a finite automaton whose degree of aperiodicity belongs to $(M]$ (or equivalently, the minimal automaton accepting L is finite with aperiodicity degree contained in $(M]$).*

As mentioned above, this result is a common extension of those obtained in [1, 8, 10, 11, 14]. In fact, we have shown that Corollary 8.1 is easily derivable from the classical results of Schützenberger [14], McNaughton and Papert [11], Kamp [10] and Gabbay et al. [8], using Corollary 4.10, which is in turn based on Theorem 4.8 and Theorem 4.6. (Of course, it is possible to prove Corollary 4.10 without using Theorem 4.6.)

Some of the implications of Corollary 8.1 are quite obvious. It is clear that the second condition implies the third as does the first. The fact that the second condition implies the first can be proved by generalizing an argument from [4] which concerns the case when M is empty. That the third condition implies the fourth can also be shown directly using Ehrenfeucht-Fraisse games, following the usual argument establishing the fact that any language definable in **FO** has an aperiodic syntactic monoid. In the classical case, i.e., when $M = \emptyset$, there are also known direct arguments establishing that the last condition implies the second. One argument is based on (a weak form of) the Krohn-Rhodes decomposition theorem, and can be found in [4]. A more elementary argument is given in [20]. Both arguments can be generalized to any given set M of moduli.

Theorem 4.8 and Theorem 4.6 are also very useful in the characterization of the expressive power of other variants of first-order and temporal logic. Various fragments of **LTL** have been studied in [4] and [20]. In a forthcoming paper, we will characterize the expressive power of the extension of most of these fragments by counting. In [16, 17], the expressive power of first-order logic with modular quantifiers with respect to any given set of moduli has been characterized, as well as the expressive power of first-order logic with modular quantifiers and the predicates $C_m^r(x)$, where m is any positive integer and $r \in [m]$. Using Theorem 4.8 and Theorem 4.6, we can give a characterization of the expressive power of the extension of first-order logic with any collection of modular quantifiers and any collection of predicates $C_m^r(x)$. A further natural research topic is to extend these results to ω-languages.

9 Acknowledgments

The first author would like to thank the members of the Department of Mathematics of Kyoto Sangyo University and the members of the DSS group of the Department of Computer Science of Aalborg University for their hospitality.

References

[1] D. A. M. Barrington, K. Compton, H. Straubing and D. Therien, Regular languages in NC^1, *J. Comput. Sys. Sci.,* 44(1992), 478–499.

[2] A. Bazirambawo, P. McKenzie and D. Therien, Modular temporal logic, *Proc. 1999 IEEE Conf. LICS, Trento, Italy,* IEEE Press, 1999, 344-351.

[3] R. Büchi, Weak second-order arithmetic and finite automata, *Z. Math. Logik Grundlag. Math.*, 6(1960), 66–92.

[4] J. Cohen, D. Perrin and J.-E. Pin, On the expressive power of temporal logic, *J. Comp. Sys. Sci.,* 46(1993), 271–294.

[5] S. Eilenberg, *Automata, Languages and Machines,* v. A and B., Academic Pr*ess, 1974 and 1976.

[6] C. C. Elgot, Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98(1961), 21–51.

[7] Z. Ésik, Results on homomorphic representation of automata by α_0-products, *Theoret. Comp. Sci.*, 87(1991), 229-249.

[8] D. M. Gabbay, A. Pnueli, S. Shelah and J. Stavi, On the temporal analysis of fairness, in: proc. *12th ACM Symp. Principles of Programming Languages,* Las Vegas, 1980, 163–173.

[9] F. Gécseg and I. Peák, *Algebraic Theory of Automata*, Akadémiai Kiadó, Budapest, 1972.

[10] J. A. Kamp, *Tense Logic and the Theory of Linear Order*, Ph. D. Thesis, UCLA, 1968.

[11] R. McNaughton and S. Papert, *Counter-Free Automata*, MIT Press, 1971.

[12] J.-E. Pin, *Varieties of Formal Languages*, North Oxford Academic, 1986.

[13] A. Pnueli, The temporal logic of programs, in: proc. *18th IEEE Symp. Foundations of Computer Science*, Providence, RI, 1977, 46–57.

[14] M. P. Schützenberger, On finite monoids having only trivial subgroups, *Inform. Comp.*, 8(1965), 190–194.

[15] H. Straubing, Constant-depth periodic circuits, *Int. J. Algebra and Computation*, 1(1991), 45–88.

[16] H. Straubing, *Finite Automata, Formal Logic and Circuit Complexity*, Birkhauser, 1994.

[17] H. Straubing, D. Therien and W. Thomas, Regular languages defined with generalized quantifiers, *Information and computation*, 118(1995), 289–301.

[18] W. Thomas, Automata on infinite objects, in: *Handbook of Theoretical Computer Science*, Vol. B, Elsevier, Amsterdam, 1990, 133–191.

[19] W. Thomas, Languages, automata, and logic, in: *Handbook of Formal Language Theory*, Vol. III, (G. Rozenberg, A. Salomaa, Eds.), Springer-Verlag, New York, 1997, 389-455.

[20] Th. Wilke, Classifying discrete temporal properties, in: proc. *STACS 99*, Trier, 1999, LNCS 1563, Springer, 1999, 32–46.

[21] M. Y. Vardi and P. Wolper, Reasoning about infinite computation, *Information and Computation*, 115(1994), 1–37.

[22] P. Wolper, Temporal logic can be more expressive, *Information and Control*, 56(1983), 72–99.

Received August, 2002

On D0L systems with finite axiom sets

Juha Honkala*

Abstract

We give a new solution for the language equivalence problem of D0L systems with finite axiom sets by using the decidability of the equivalence problem of finite valued transducers on HDT0L languages proved by Culik II and Karhumäki.

1 Introduction

The language equivalence problem for D0L systems with finite axiom sets was solved in [4]. The problem turns out to be much more difficult for DF0L systems than for D0L systems. The main idea in [4] is to decompose a given DF0L language in a canonical way into finitely many parts such that no part contains two words with equal Parikh vectors. This makes it possible to use ideas from [8]. The resulting algorithm gives a lot of information concerning the structures of the languages generated by two equivalent DF0L systems. Also the equivalence problem for DF0L power series over a computable field is solved in [4].

The purpose of this paper is to give a new solution of the DF0L language equivalence problem. The new proof for the decidability of the problem avoids many difficulties in [4] but fails to give precise information about language equivalent DF0L systems. In that respect it resembles the solutions of the D0L equivalence problem based on Hilbert's basis theorem which also are short but do not, for example, give any bounds for the problem (see [3]).

Our new solution again uses methods from [8] which in turn use ideas from [1]. In addition, we use the decidability of the equivalence problem of finite valued transducers proved by Culik II and Karhumäki [2]. In this way we obtain a solution of the DF0L language equivalence problem which is essentially based on commutative methods (see [5]).

For further background and motivation we refer to [6, 7, 8, 9, 10, 4]. It is assumed that the reader is familiar with the basics concerning D0L systems and their generalizations such as HDT0L systems, see [6, 7].

*Department of Mathematics, University of Turku, FIN-20014 Turku, Finland. Email: juha.honkala@utu.fi
Research supported by the Academy of Finland.

2 Definitions and earlier results

Let $X = \{x_1, \ldots, x_k\}$ be an alphabet with $k \geq 1$ letters. The *Parikh mapping* $\psi : X^* \longrightarrow \mathbb{N}^k$ is defined by

$$\psi(w) = (\#_{x_1}(w), \ldots, \#_{x_k}(w)),$$

for $w \in X^*$. Here $\#_{x_i}(w)$ is the number of occurrences of the letter x_i in the word w. The *length* of a word w is denoted by $|w|$. The length of the empty word ε equals zero.

A *D0L system* is a triple $G = (X, h, w)$ where X is a finite alphabet, $h : X^* \longrightarrow X^*$ is a morphism and $w \in X^*$ is a word. A DF0L system is obtained from a D0L system by replacing the word w by a finite set F. Hence, a *DF0L system* is a triple $G = (X, h, F)$ where X is a finite alphabet, $h : X^* \longrightarrow X^*$ is a morphism and $F \subseteq X^*$ is a finite set.

The *sequence* $S(G)$ and the *language* $L(G)$ of the D0L system $G = (X, h, w)$ are given by

$$S(G) = (h^n(w))_{n \geq 0}$$

and

$$L(G) = \{h^n(w) \mid n \geq 0\}.$$

The *language* $L(G)$ of the DF0L system $G = (X, h, F)$ is defined by

$$L(G) = \{h^n(w) \mid w \in F,\ n \geq 0\}.$$

Below we will discuss also DT0L and HDT0L systems. By definition, a *DT0L system* is a construct $(X, h_1, \ldots, h_n, w)$ such that $n \geq 1$ is an integer and (X, h_i, w) is a D0L system for $1 \leq i \leq n$. An *HDT0L system* is a construct $G = (X, Y, h_1, \ldots, h_n, h, w)$ such that $(X, h_1, \ldots, h_n, w)$ is a DT0L system (called the *underlying DT0L system* of G), Y is a finite alphabet and $h : X^* \longrightarrow Y^*$ is a morphism.

Let $G = (X, Y, h_1, \ldots, h_n, h, w)$ be an HDT0L system and let $Z_n = \{z_1, \ldots, z_n\}$ be an alphabet with n letters. Then the *sequence* of G is the mapping $S(G) : Z_n^* \longrightarrow Y^*$ defined by

$$S(G)(z_{i_1} \ldots z_{i_m}) = h h_{i_m} \ldots h_{i_1}(w)$$

for $m \geq 0$, $1 \leq i_1, \ldots, i_m \leq n$. The *sequence* of a DT0L system $(X, h_1, \ldots, h_n, w)$ equals the sequence of the HDT0L system $(X, X, h_1, \ldots, h_n, g, w)$ where the morphism $g : X^* \longrightarrow X^*$ is defined by $g(x) = x$ for all $x \in X$.

A *finite transducer* is a construct $\tau = (Q, \Sigma, \Delta, s_0, F, E)$ where Q is the finite set of states, Σ and Δ are the input and output alphabets, respectively, $s_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $E \subseteq Q \times \Sigma^* \times \Delta^* \times Q$ is the finite set consisting of the transitions of τ. If $u \in \Sigma^*$ and $v \in \Delta^*$ we write $v \in \tau(u)$ if there is an accepting computation of τ having input u and output v. Let k be a nonnegative integer. A transducer τ is called *k-valued* if for all $u \in \Sigma^*$ the set

$\tau(u)$ contains at most k words. Finally, a transducer τ is called *finite valued* if it is k-valued for some k.

The following important result is due to Culik II and Karhumäki, [2]. Here two transducers τ_1 and τ_2 are called equivalent on a language L if $\tau_1(u) = \tau_2(u)$ for all $u \in L$.

Theorem 1. *It is decidable whether two finite valued finite transducers are equivalent on a given HDT0L language.*

3 The HDT0L covering problem

In this section we discuss the HDT0L covering problem which is a useful tool in the study of the DF0L language equivalence problem. It would suffice to consider the D0L covering problem but this would not simplify the discussion.

Let $H_i = (X_i, Y_i, h_{i1}, \ldots, h_{in}, h_i, w_i)$, $1 \le i \le k+1$, be HDT0L systems. Then we say that the first k sequences $S(H_i)$ *cover* the last sequence $S(H_{k+1})$ if

$$S(H_{k+1})(u) \in \{S(H_i)(u) \mid 1 \le i \le k\}$$

for all $u \in Z_n^*$. If $k = 1$, then $S(H_1)$ covers $S(H_2)$ if and only if H_1 and H_2 are sequence equivalent. If $k > 1$, the covering relation generalizes sequence equivalence by allowing finitely many alternatives for each term of $S(H_{k+1})$.

Let H_i, $1 \le i \le k+1$, be as above. By the *HDT0L covering problem* we understand the problem of deciding whether or not $S(H_i)$, $1 \le i \le k$, cover $S(H_{k+1})$. To reduce the covering problem to the equivalence problem of finite valued transducers one lemma is required.

Lemma 2. *Let $H_i = (X_i, Y_i, h_{i1}, \ldots, h_{in}, h_i, w_i)$, $1 \le i \le k$, be HDT0L systems. Then there is a DT0L system $H = (X, f_1, \ldots, f_n, w)$ and finite valued finite transducers τ_I for $I \subseteq \{1, \ldots, k\}$ such that*

$$\tau_I(S(H)(u)) = \{S(H_i)(u) \mid i \in I\} \tag{1}$$

for all $u \in Z_n^*$.

Proof. We may assume that the alphabets X_i, $1 \le i \le k$, are pairwise disjoint. Denote $X = X_1 \cup \ldots \cup X_k$, $Y = Y_1 \cup \ldots \cup Y_k$ and let $f_j : X^* \longrightarrow X^*$ be the morphism such that

$$f_j(x) = h_{ij}(x)$$

whenever $x \in X_i$, $1 \le i \le k$, $1 \le j \le n$. Denote $w = w_1 \ldots w_k$ and consider the DT0L system $H = (X, f_1, \ldots, f_n, w)$.

Let $\overline{H}_i = (X_i, h_{i1}, \ldots, h_{in}, w_i)$ be the underlying DT0L system of H_i, $1 \le i \le k$. Then we have

$$S(H)(u) = S(\overline{H}_1)(u) \ldots S(\overline{H}_k)(u) \tag{2}$$

for $u \in Z_n^*$.

Let now $I \subseteq \{1,\dots,k\}$ be a nonempty set and let τ_I be a transducer defined as follows. The input alphabet of τ_I is X and the output alphabet of τ_I is Y. The state set of τ_I is $\{q_0\} \cup \{q_i \mid i \in I\}$ where q_0 is the initial state and $\{q_i \mid i \in I\}$ is the final state set. The set E of transitions is defined by

$$\begin{aligned} E &= \{(q_0,\varepsilon,\varepsilon,q_i) \mid i \in I\} \cup \\ &\quad \{(q_i,x,h_i(x),q_i) \mid i \in I \text{ and } x \in X_i\} \cup \\ &\quad \{(q_i,x,\varepsilon,q_i) \mid i \in I \text{ and } x \notin X_i\}. \end{aligned}$$

Then τ_I is finite valued and (2) implies (1) for all $u \in Z_n^*$. □

Theorem 3. *The HDT0L covering problem is decidable.*

Proof. Let $H_i = (X_i, Y_i, h_{i1},\dots,h_{in},h_i,w_i)$, $1 \le i \le k+1$, be HDT0L systems. Denote $I = \{1,\dots,k\}$ and $J = \{1,\dots,k+1\}$. By Lemma 2 there exist a DT0L system $H = (X, f_1,\dots,f_n,w)$ and finite valued finite transducers τ_I and τ_J such that

$$\tau_I(S(H)(u)) = \{S(H_i)(u) \mid i \in I\}$$

and

$$\tau_J(S(H)(u)) = \{S(H_j)(u) \mid j \in J\}$$

for all $u \in Z_n^*$. Now

$$\tau_I(S(H)(u)) = \tau_J(S(H)(u)) \text{ for all } u \in Z_n^* \tag{3}$$

if and only if

$$S(H_{k+1})(u) \in \{S(H_i)(u) \mid 1 \le i \le k\} \text{ for all } u \in Z_n^*.$$

The claim follows because by Theorem 1 we can decide the validity of (3). (Here we use Theorem 1 for DT0L languages.) □

4 The DF0L language equivalence problem

Let X be an alphabet with $k \ge 1$ letters and let $\psi : X^* \longrightarrow \mathbf{N}^k$ be the Parikh mapping. If $K \subseteq \mathbf{N}^k$ we denote

$$\psi^{-1}(K) = \{w \in X^* \mid \psi(w) \in K\}.$$

Lemma 4. *Let $G = (X,h,F)$ be a DF0L system and let $u \in F$. Assume that $\{h^i(u) \mid i \ge 0\}$ is an infinite set. Then there exist an integer $s \ge 0$, integers $n_1,\dots,n_s$ and words $u_1,\dots,u_s \in F$ such that*

$$\psi^{-1}(\psi h^n(u)) \cap L(G) = \{h^{n+n_1}(u_1),\dots,h^{n+n_s}(u_s)\}$$

for almost all $n \ge 0$.

Proof. We will show that if $v \in F$ then either

$$\psi^{-1}(\psi h^n(u)) \cap \{h^i(v) \mid i \geq 0\} = \emptyset \tag{4}$$

for almost all $n \geq 0$ or, otherwise, there exists an integer m such that

$$\psi^{-1}(\psi h^n(u)) \cap \{h^i(v) \mid i \geq 0\} = \{h^{n+m}(v)\} \tag{5}$$

for almost all $n \geq 0$. (Here and in the sequel we say that a property holds for almost all n if there is an integer n_0 such that the property holds for all $n \geq n_0$.)

First, if $\{h^i(v) \mid i \geq 0\}$ is a finite set then (4) holds for almost all $n \geq 0$. Suppose $\{h^i(v) \mid i \geq 0\}$ is infinite. Then

$$\psi h^i(v) \neq \psi h^j(v) \quad \text{if} \quad i \neq j.$$

Now, if there exist integers m_1 and m_2 such that

$$\psi h^{m_1}(u) = \psi h^{m_2}(v) \tag{6}$$

then (5) holds for almost all $n \geq 0$ if we set $m = m_2 - m_1$. Finally, if (6) holds for no values of m_1 and m_2 then (4) holds for all $n \geq 0$. □

Let $G = (X, h, F)$ be a DF0L system. A word sequence $(w_n)_{n \geq 0}$ is called a *subsequence* of G if there exist $w \in L(G)$ and a positive integer a such that

$$w_n = h^{an}(w)$$

for all $n \geq 0$. In Section 3 we have explained what it means that a given D0L sequence is covered by finitely many given D0L sequences. We now define this notion for DF0L systems.

Let $G_i = (X, h_i, F_i)$, $i = 1, 2$, be DF0L systems. Then G_2 is said to *cover* G_1 if for all $u \in F_1$ there exist a nonnegative integer r and a positive integer k such that for all integers j, $0 \leq j < k$, the sequence $(h_1^{kn+j+r}(u))_{n \geq 0}$ is covered by finitely many subsequences of G_2.

Lemma 5. *Let $G_i = (X, h_i, F_i)$, $i = 1, 2$, be DF0L systems. Assume that $L(G_1) = L(G_2)$ and that alph$(w) = X$ for all $w \in L(G_1)$. Then G_1 and G_2 cover each other.*

Proof. Let $G_i = (X, h_i, F_i)$, $i = 1, 2$, be DF0L systems such that $L(G_1) = L(G_2)$ and alph$(w) = X$ for all $w \in L(G_1)$. If $L(G_1)$ is finite the claim holds. Assume that $L(G_1)$ is infinite. Without restriction assume also that card$(F_1) =$ card(F_2). (If necessary, we replace F_1 by the set $\{h_1^j(u) \mid u \in F_1, 0 \leq j <$ card$(F_2)\}$ and F_2 by the set $\{h_2^j(v) \mid v \in F_2, 0 \leq j <$ card$(F_1)\}$.) Denote $t =$ card(F_1),

$$F_1 = \{u_0, \ldots, u_{t-1}\}$$

and

$$F_2 = \{v_0, \ldots, v_{t-1}\}.$$

Further, denote $k = \mathrm{card}(X)$ and let $P(x_1, \ldots, x_k)$ be a polynomial with nonnegative integer coefficients such that the mapping $P : \mathbf{N}^k \longrightarrow \mathbf{N}$ is injective (see [8]). Define the mappings $f : \mathbf{N} \longrightarrow \mathbf{N}$ and $g : \mathbf{N} \longrightarrow \mathbf{N}$ by

$$f(ti + j) = P(\psi h_1^i(u_j))$$

and

$$g(ti + j) = P(\psi h_2^i(v_j))$$

for $i \geq 0$ and $0 \leq j < t$. Then f and g are D0L growth functions (see [8]) and

$$\{f(n) \mid n \in \mathbf{N}\} = \{g(n) \mid n \in \mathbf{N}\}.$$

Hence there exist integers $a \geq 1$, $r \geq 0$, $x_k \geq 1$ and $y_k \geq 0$ for $0 \leq k < a$ such that

$$f(an + k + r) = g(x_k n + y_k)$$

for $n \geq 0$, $0 \leq k < a$ (see [1]). Without restriction we assume that t divides a and that t divides x_k for all $0 \leq k < a$. Denote $a = bt$. Fix $u \in F_1$. It follows that there is an integer $\beta \geq 0$ such that for all integers α, $0 \leq \alpha < b$, there exist $v_{j_\alpha} \in F_2$ and integers $q_\alpha \geq 1$, $p_\alpha \geq 0$ such that

$$\psi h_1^{bn+\alpha+\beta}(u) = \psi h_2^{q_\alpha n + p_\alpha}(v_{j_\alpha})$$

for $n \geq 0$. Because $L(G_1) = L(G_2)$ we have

$$h_1^{bn+\alpha+\beta}(u) \in \psi^{-1}(\psi h_2^{q_\alpha n+p_\alpha}(v_{j_\alpha})) \cap L(G_2)$$

for $n \geq 0$.

Next, fix α, $0 \leq \alpha < b$. Because $\mathrm{alph}(v_{j_\alpha}) = X$, the set $\{h_2^i(v_{j_\alpha}) \mid i \geq 0\}$ is infinite. By Lemma 4 there exist an integer $s \geq 0$, integers $n_1, \ldots, n_s$ and words $w_1, \ldots, w_s \in F_2$ such that

$$\psi^{-1}(\psi h_2^{q_\alpha n+p_\alpha}(v_{j_\alpha})) \cap L(G_2) = \{h_2^{q_\alpha n+p_\alpha+n_1}(w_1), \ldots, h_2^{q_\alpha n+p_\alpha+n_s}(w_s)\}$$

for almost all $n \geq 0$. Hence

$$h_1^{bn+\alpha+\beta}(u) \in \{h_2^{q_\alpha n+p_\alpha+n_1}(w_1), \ldots, h_2^{q_\alpha n+p_\alpha+n_s}(w_s)\}$$

for almost all $n \geq 0$. In other words, G_2 covers G_1. It is seen similarly that G_1 covers G_2. □

Theorem 6. *It is decidable whether or not two given DF0L systems are language equivalent.*

Proof. It suffices to consider DF0L systems $G = (X, h, F)$ such that $\mathrm{alph}(w) = X$ for all $w \in L(G)$ (see [8]). The claim follows because there exists a semialgorithm for equivalence and there exists a semialgorithm for nonequivalence. The existence of a semialgorithm for equivalence follows by Theorem 3 and Lemma 5. (Here we use Theorem 3 for D0L systems.) The existence of a semialgorithm for nonequivalence is clear. □

References

[1] J. Berstel and M. Nielsen, The growth range equivalence problem for D0L systems is decidable, in A. Lindenmayer and G. Rozenberg (eds.): *Automata, Languages, Development* (North-Holland, Amsterdam, 1976) 161-178.

[2] K. Culik II and J. Karhumäki, The equivalence of finite valued transducers (on HDT0L languages) is decidable, *Theoret. Comput. Sci.* **47** (1986) 71-84.

[3] J. Honkala, A short solution for the HDT0L sequence equivalence problem, *Theoret. Comput. Sci.* **244** (2000) 267-270.

[4] J. Honkala, The equivalence problem for DF0L languages and power series, *J. Comput. System Sci.* **65** (2002) 377-392.

[5] J. Honkala, A note on systems of alternative word equations, *Bull. EATCS* **78** (2002) 237-240.

[6] G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems* (Academic Press, New York, 1980).

[7] G. Rozenberg and A. Salomaa (eds.): *Handbook of Formal Languages*, Vol. 1-3 (Springer, Berlin, 1997).

[8] K. Ruohonen, The decidability of the F0L-D0L equivalence problem, *Inform. Process. Letters* **8** (1979) 257-260.

[9] K. Ruohonen, On a variant of a method of Berstel's and Nielsen's, *Fund. Inform.* **4** (1981) 369-400.

[10] K. Ruohonen, Equivalence problems for regular sets of word morphisms, in G. Rozenberg and A. Salomaa (eds.), *The Book of L* (Springer, Berlin, 1986) 393-401.

Received August, 2002

On directable nondeterministic trapped automata*

B. Imreh[†] Cs. Imreh[†] and M. Ito[‡]

Abstract

A finite automaton is said to be directable if it has an input word, a directing word, which takes it from every state into the same state. For nondeterministic (n.d.) automata, directability can be generalized in several ways. In [8], three such notions, D1-, D2-, and D3-directability, are introduced. In this paper, we introduce the trapped n.d. automata, and for each $i = 1, 2, 3$, present lower and upper bounds for the lengths of the shortest Di-directing words of n-state Di-directable trapped n.d. automata. It turns out that for this special class of n.d. automata, better bounds can be found than for the general case, and some of the obtained bounds are sharp.

1 Introduction

An input word w is called a *directing* (or *synchronizing) word* of an automaton $\mathcal{A}$ if it takes $\mathcal{A}$ from every state to the same state. Directable automata have been studied exstensively, we mention only some of the related works (see *e.g.* [3],[4],[5],[7],[10],[12]). Directable n.d. automata have received less attention. Directability of n.d. automata can be defined in several meaningful ways. The following three nonequivalent definitions are introduced and studied in [8]. An input word w of an n.d. automaton $\mathcal{A}$ is said to be

(1) D1-*directing* if it takes $\mathcal{A}$ from every state to the same singleton set,

(2) D2-*directing* if it takes $\mathcal{A}$ from every state to the same fixed set A', where $\emptyset \subseteq A' \subseteq A$, and

(3) D3-*directing* if there is a state c such that $c \in aw$, for every $a \in A$.

The D1-directability of complete n.d. automata was investigated by Burkhard [1]. He gave a sharp exponential bound for the lengths of minimum-length D1-directing words of complete n.d. automata. In [6] on games of composing relations over a finite set Goralčik *it et al.*, in effect, studied D1- and D3-directability and they proved that neither for D1- nor for D3-directing words, the bound can be polynomial

*This work has been supported by a collaboration between the Hungarian Academy of Science and the Japan Society for the Promotion of Science, and the Hungarian National Foundation for Science Research, Grant T037258.

[†]Department of Informatics, University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary

[‡]Dept. of Mathematics, Faculty of Science, Kyoto Sangyo University, Kyoto 603, Japan

for n.d. automata. Carpi [2] considered a particular class of n.d. automata, the class of unambigous n.d. automata, and presented $O(n^3)$ bounds for the lengths of their shortest D1-directing words.

Here we study trapped n.d. automata that have a trap state, *i.e.*, a state which is stable for any input symbol, and present lower and upper bounds for the lengths of their shortest directing words of the three different types.

2 Preliminaries

Throughout this paper X always denotes a finite nonempty alphabet. The set of all finite words over X is denoted by X^* and λ denotes the empty word. The length of a word $w \in X^*$ is denoted by $|w|$. For any $p, q \in X^*$, the word p is called a *prefix* of q if there exists a word $s \in X^*$ such that $ps = q$. For the sake of simplicity, we use the notation $[n]$ for the set $\{1, \ldots, n\}$.

By a *nondeterministic (n.d.) automaton* we mean a system $\mathcal{A} = (A, X)$, where A is a nonempty finite set of *states*, X is the *input alphabet*, and each input symbol $x \in X$ is realized as a binary relation $x^{\mathcal{A}} (\subseteq A \times A)$. For any $a \in A$ and $x \in X$, let

$$ax^{\mathcal{A}} = \{b : b \in A \text{ and } (a, b) \in x^{\mathcal{A}}\}.$$

Moreover, for every $B \subseteq A$, we denote by $Bx^{\mathcal{A}}$ the set $\bigcup\{ax^{\mathcal{A}} : a \in B\}$. Now, for any word $w \in X^*$ and $B \subseteq A$, $Bw^{\mathcal{A}}$ can be defined inductively as follows:

(1) $B\lambda^{\mathcal{A}} = B$,

(2) $Bw^{\mathcal{A}} = (Bp^{\mathcal{A}})x^{\mathcal{A}}$ for $w = px$, where $p \in X^*$ and $x \in X$.

If $w = x_1 \ldots x_m$ and $a \in A$, then let $aw^{\mathcal{A}} = \{a\}w^{\mathcal{A}}$. This yields that $w^{\mathcal{A}} = x_1^{\mathcal{A}} \ldots x_m^{\mathcal{A}}$. If there is no danger of confusion, then we write simply aw and Bw for $aw^{\mathcal{A}}$ and $Bw^{\mathcal{A}}$, respectively.

An n.d. automaton $\mathcal{A} = (A, X)$ is *complete* if $ax \neq \emptyset$ holds, for all $a \in A$ and $x \in X$. Complete n.d. automata are called c.n.d. automata for short. A state of an n.d. automaton $\mathcal{A}$ is called a *trap* if it is stable for any input symbol, *i.e.*, $ax = \{a\}$, for every input symbol x of $\mathcal{A}$. An n.d. automaton is called *trapped* if it has a trap. Let us denote the class of trapped n.d. automata by $\mathbf{T}$. Regarding some recent results on trapped deterministic automata, we refer to the works [9],[11],[12]. Following [8] we define the directability of n.d. automata as follows. Let $\mathcal{A} = (A, X)$ be an n.d. automaton. For any word $w \in X^*$, let us consider the following conditions:

(D1) $(\exists c \in A)(\forall a \in A)(aw = \{c\})$,

(D2) $(\forall a, b \in A)(aw = bw)$,

(D3) $(\exists c \in A)(\forall a \in A)(c \in aw)$.

For any $i = 1, 2, 3$, if w satisfies Di, then w is called a Di-*directing word* of $\mathcal{A}$ and in this case $\mathcal{A}$ is said to be Di-*directable*. Let us denote by $\mathrm{D}_i(\mathcal{A})$ the set of Di-directing words of $\mathcal{A}$, moreover, let $\mathbf{Dir}(i)$ and $\mathbf{CDir}(i)$ denote the classes of

Di-directable n.d. automata and c.n.d. automata, respectively. Now, we can define the following functions. For any $i = 1,2,3$ and $\mathcal{A} = (A, X)$ $\in$**Dir**(i), let

$$\mathrm{d}_i(\mathcal{A}) = \min\{|w| : w \in \mathrm{D}_i(\mathcal{A})\},$$

$$\mathrm{d}_i(n) = \max\{\mathrm{d}_i(\mathcal{A}) : \mathcal{A} \in \mathbf{Dir}(i) \ \& \ |A| = n\},$$

$$\mathrm{cd}_i(n) = \max\{\mathrm{d}_i(\mathcal{A}) : \mathcal{A} \in \mathbf{CDir}(i) \ \& \ |A| = n\}.$$

The functions $\mathrm{d}_i(n), \mathrm{cd}_i(n)$, $i = 1,2,3$, are studied in [8], where lower and upper bounds depending on n are presented for them. Similar functions can be defined for any class of n.d. automata. For a class $\mathbf{K}$ of n.d. automata, let

$$\mathrm{d}_i^{\mathbf{K}}(n) = \max\{\mathrm{d}_i(\mathcal{A}) : \mathcal{A} \in \mathbf{Dir}(i) \cap \mathbf{K} \ \& \ |A| = n\},$$

$$\mathrm{cd}_i^{\mathbf{K}}(n) = \max\{\mathrm{d}_i(\mathcal{A}) : \mathcal{A} \in \mathbf{CDir}(i) \cap \mathbf{K} \ \& \ |A| = n\}.$$

Obviously, $\mathrm{cd}_i^{\mathbf{K}}(n) \leq \mathrm{d}_i^{\mathbf{K}}(n)$, for $i = 1,2,3$.

In what follows, we study the case when the considered class is $\mathbf{T}$, the class of trapped n.d. automata. It is worth noting that if a trapped n.d. automaton is Di-directable, then it has only one trap.

3 Directable trapped n.d. automata

First we deal with the D3-directability. We consider D3-directable trapped c.n.d. automata, and using certain deterministic automata, introduced by Rystsov [12], we present an exact bound for this class. Then we study D3-directable trapped n.d. automata and present lower and upper bounds for the lengths of their shortest D3-directing words. For trapped c.n.d. automata the following statement is valid.

Theorem 1. *For any $n \geq 1$, $\mathrm{cd}_3^{\mathbf{T}}(n) = (n-1)n/2$.*

Proof. First we prove that $(n-1)n/2 \leq \mathrm{cd}_3^{\mathbf{T}}(n)$. This inequality follows from Theorem 6.1 in [12]. Since the proof is short, we recall it for the sake of completeness.

For every $n > 1$, let us define the c.n.d. automaton $\mathcal{B}_n = (\{0,1,\ldots,n-1\},\{x_1,\ldots,x_{n-1}\})$ as follows. Let $0x_1 = 1x_1 = \{0\}$, and $jx_1 = \{j\}, j = 2,\ldots,n-1$. Moreover, for all $2 \leq k \leq n-1$ and $j \in \{0,1,\ldots,n-1\}$, let

$$jx_k = \begin{cases} \{j-1\} & \text{if } j = k, \\ \{j+1\} & \text{if } j = k-1, \\ \{j\} & \text{otherwise.} \end{cases}$$

Obviously, $\mathcal{B}_n$ is a D3-directable trapped c.n.d. automaton with the trap 0. Let us observe that for any $j \in \{0,1,\ldots,n-1\}$, jp is a singleton set whenever $p \in X^*$, moreover, $jw = \{0\}$ for any D3-directing word of $\mathcal{B}_n$, because 0 is a trap state. Therefore, $\{0,1,\ldots,n-1\}w = \{0\}$, for any $w \in \mathrm{D}_3(\mathcal{B}_n)$. Now, let us assign to every nonempty subset J of states a weight, denoted by $g(J)$, which is the sum of the numbers contained in J, *i.e.*,

$$g(J) = \sum_{j \in J} j.$$

Then $g(\{0,1,\ldots,n-1\}) = (n-1)n/2$ and for any nonempty subset J of $\{0,1,\ldots,n-1\}$ and input sign x_k, $k \in [n-1]$,

$$|g(J) - g(Jx_k)| \le 1.$$

From these facts it follows that the length of any D3-directing word of $\mathcal{B}_n$ is not less than $(n-1)n/2$, because this word brings the state set of weight $(n-1)n/2$ into the set $\{0\}$ with weight 0. Hence, $(n-1)n/2 \le \mathrm{d}_3(\mathcal{B}_n)$. On the other hand, it is easy to check that the word

$$w = x_1x_2x_1x_3x_2x_1 \ldots x_{n-1}x_{n-2} \ldots x_2x_1$$

is a D3-directing word of $\mathcal{B}_n$ and $|w| = (n-1)n/2$. Consequently,

$$\mathrm{d}_3(\mathcal{B}_n) = (n-1)n/2.$$

Since $\mathcal{B}_n$ is a D3-directable trapped c.n.d. automaton of n states, the equality above implies $(n-1)n/2 \le \mathrm{cd}_3^{\mathrm{T}}(n)$.

In order to prove that this bound is sharp, we prove that for any D3-directable trapped c.n.d. automaton $\mathcal{A} = (A, X)$ of $n(> 1)$ states, there exists a D3-directing word whose length is not greater than $(n-1)n/2$. To simplify the notation, we assume that $A = \{0,1,\ldots,n-1\}$ and 0 is the trap of $\mathcal{A}$. Since $\mathcal{A}$ is a D3-directable c.n.d. automaton and $0x = \{0\}$, for all $x \in X$, there exists for any state $j \in A$ a word $x_1 \ldots x_m$ of minimum-length such that $0 \in jx_1 \ldots x_m$. Moreover, there are states $j_1, \ldots, j_{m-1} \in A$ such that $j_t \in jx_1 \ldots x_t$ and $0 \in j_tx_{t+1} \ldots x_m$, for all $t = 1, \ldots, m-1$. Since $x_1 \ldots x_m$ is a minimum-length word satisfying $0 \in jx_1 \ldots x_m$, the states $j, j_1, \ldots, j_{m-1}, 0$ must be pairwise different. Therefore, by $|A| = n$, we obtain $m \le n-1$. Observe that for any $2 \le t \le m$, $x_t \ldots x_m$ is a minimum-length word satisfying $0 \in j_{t-1}x_t \ldots x_m$. Based on these observations, by renaming the states, we may suppose that for any state $j \in A$, there exists a word p_j such that $0 \in jp_j$ and $|p_j| \le j$. By using the pairs j, p_j, $j = 0, \ldots, n-1$, we present a procedure for finding a D3-directing word with length, not greater than $(n-1)n/2$.

Initialization. Let $t = 0$, $B_0 = \{0\}$, $p_{i_0} = \lambda$, and $R_0 = \{1, 2, \ldots n-1\}$.

Iteration.

- *Step 1.* Terminate if $A = B_t$. Otherwise proceed to Step 2.
- *Step 2.* For each $j \in R_t$, let $k_j^{(t)}$ denote the smallest number in the set jp_{i_t}. Select the least element in $\{k_j^{(t)} : j \in R_t\}$ and denote it by i_{t+1}. Let

$$B_{t+1} = \{j : j \in A \;\&\; 0 \in jp_{i_0} \ldots p_{i_{t+1}}\}.$$

and

$$R_{t+1} = \{k_j^{(t)} : j \in A \setminus B_{t+1}\}.$$

Increase the value of t by 1 and proceed to the next iteration.

To verify the correctness of the above procedure, we note the following facts.

(i) For any i_{t+1}, there exists a $j \in A \setminus B_t$ such that i_{t+1} is an element of the set $jp_{i_0} \dots p_{i_t}$. Then $B_t \cup \{j\} \subseteq B_{t+1}$, and hence, $B_t \subset B_{t+1}$.

(ii) If $j \in A \setminus B_t$, then $0 \notin jp_{i_0} \dots p_{i_t}$ yielding $k_j^{(t)} > 0$. Therefore, R_t is a set of positive integers.

(iii) If $A \neq B_t$, then there is a $j \in A \setminus B_t$ with $jp_{i_0} \dots p_{i_t} \neq \emptyset$ since $\mathcal{A}$ is a c.n.d. automaton, and thus, $R_t \neq \emptyset$. Consequently, $A \neq B_t$ implies $R_t \neq \emptyset$.

From these facts it follows that there exists a positive integer $s \leq n-1$ such that $A = B_s$. Now, by the definition of B_s, we obtain that

$$w = p_{i_0} \dots p_{i_s}$$

is a D3-directing word of $\mathcal{A}$. Let $r_t = |R_t|$, $t = 0, \dots, s-1$. From the definition of R_t it follows that

$$n - 1 \geq r_0 > r_1 > \dots > r_{s-1} > 0.$$

On the other hand, since $|R_t| = r_t$, the least number i_{t+1} of $\{k_j^{(t)} : j \in R_t\}$ is not greater than $n - r_t$. This yields that $|p_{i_{t+1}}| \leq n - r_t$, $t = 0, \dots, s-1$. Since $|p_{i_0}| = 0$, we obtain that

$$|w| \leq \sum_{t=0}^{s-1} (n - r_t).$$

Let us observe that the numbers $n - r_t$, $t = 0, \dots, s-1$ are pairwise different and each of them is contained in the set $[n-1]$. Therefore, the upper bound of $|w|$ is the sum of some distinct elements of $[n-1]$. But this sum is not greater than the sum $(n-1)n/2$ of all the elements of $[n-1]$. Consequently, $|w| \leq (n-1)n/2$. If $n = 1$, then the statement is obviously also valid. This completes the proof of Theorem 1. □

For D3-directable trapped n.d. automata, we have the following bounds.

Theorem 2. *For any* $n \geq 2$, $\max\{\lfloor n^{\frac{1}{3}} - 1\rfloor !, (n-2)^2 + 1\} \leq \mathrm{d}_3^{\mathrm{T}}(n) \leq 2^{n-1} - 1$.

Proof. The first member in the lower bound comes from the general case (*cf.* [8]), where the automata, providing this bound, are trapped automata. The second member in the lower bound can be derived from Černý's well-known examples (*cf.* [3]) as follows. One can equip Černý's automaton of $n-1$ states with a trap state and a new input symbol, denoted by $\diamond$ and z, respectively. Let $\diamond z = \{\diamond\}$, $0z = \{\diamond\}$, and $jz = \emptyset$, for all $j = 1, \ldots, n-1$. Now, redefine the remaining transitions as follows. If $ax = b$, then let $ax = \{b\}$ be the new transition. Then we obtain an n.d. automaton of n states whose shortest D3-directing words are of length $(n-2)^2 + 1$.

To obtain the upper bound, let us consider an arbitrary D3-directable trapped n.d. automaton $\mathcal{A} = (A, X)$ of $n(> 1)$ states. Let $A = \{a_1, \ldots, a_n\}$ and a_n be the trap of $\mathcal{A}$. Let $w = x_1 \ldots x_m$ be a minimum-length D3-directing word of $\mathcal{A}$. Then $a_n w = \{a_n\}$ and by the D3-directability of $\mathcal{A}$, $a_n \in a_j x_1 \ldots x_m$, $j = 1, \ldots, n$. For all $j \in [n-1]$ and $k \in [m]$, let us select an element a_{jk} from $a_j x_1 \ldots x_k$ such that $a_n \in a_{jk} x_{k+1} \ldots x_m$. Such elements exist, because for every $j \in [n-1]$, $a_n \in a_j x_1 \ldots x_m$. Now, let $S_k = \{a_n\} \cup \{a_{1k}, \ldots, a_{n-1,k}\}$, for all $k \in [m]$, and $S_0 = \{a_1, \ldots, a_n\}$. Let us observe that $a_j x_1 \ldots x_k \cap S_k \neq \emptyset$, for every $k \in [m]$, and if $a_t \in S_k$ for some $t \in [n]$ and $k \in [m]$, then $a_n \in a_t x_{k+1} \ldots x_m$. By using these observations, it is easy to see that if $S_j = S_l$ for some $0 \le j < l \le m$, then $x_1 \ldots x_j x_{l+1} \ldots x_m$ is a D3-directing word of $\mathcal{A}$ which is a contradiction. Consequently, the sets $S_0, S_1, \ldots, S_m$ must be pairwise different. Since $a_n \in S_k$, $k = 0, \ldots, m$, the number of these sets can not exceed 2^{n-1}. Therefore, $|w| \le 2^{n-1} - 1$. This ends the proof of Theorem 2. □

Remark 1. It is worth noting that the proof above with a small changing can be applied for the general case, and one obtains the upper bound $2^n - 1$ for $\mathrm{d}_3(n)$ which is a significant improvement of the upper bound, given in [8].

Now, we study D1-directable trapped c.n.d. automata. By a slight modification of the automata, introduced by Burkhard [1], we prove the following sharp bound.

Theorem 3. *For any* $n \ge 1$, $\mathrm{cd}_1^{\mathrm{T}}(n) = 2^{n-1} - 1$.

Proof. First we prove that $2^{n-1} - 1$ is a lower bound for $\mathrm{cd}_1^{\mathrm{T}}$. To do so, for every integer $n > 1$, we present a D1-directable trapped c.n.d. automaton, having a minimum-length D1-directing word w with $|w| = 2^{n-1} - 1$.

Let us define the c.n.d. automaton $\mathcal{A}_n = ([n], X^{(n)})$ as follows. For every integer $2 \le k \le n-2$, let us consider all of the k-element subsets of the set $A' = \{2, \ldots, n\}$. Let us order these sets in a chain such that the first set is $\{n-k, \ldots, n-1\}$ and the last one is $\{n-k+1, \ldots, n\}$. We denote this sequence by $A_1^{(k)}, \ldots, A_{\binom{n-1}{k}}^{(k)}$. Now, let $X_k = \{x_r^{(k)} : r = 1, \ldots, \binom{n-1}{k} - 1\}$, $V = \{v_1, \ldots, v_{n-1}\}$, $Y = \{y_1, \ldots, y_{n-2}\}$, and

$$X^{(n)} = V \cup Y \cup \left(\bigcup \{X_k : 2 \le k \le n-2\}\right).$$

The transitions of $\mathcal{A}_n$ are defined as follows. For any $x \in X^{(n)}$, let $1x = \{1\}$. Moreover, for any $x_r^{(k)} \in X_k$, $v_t \in V$, $y_s \in Y$, and state $j \in A'$, let

$$jv_t = \begin{cases} \{j-1\} & \text{if } t = j-1, \\ A' & \text{otherwise,} \end{cases}$$

$$jx_r^{(k)} = \begin{cases} A_{r+1}^{(k)} & \text{if } j \in A_r^{(k)}, \\ A' & \text{otherwise,} \end{cases}$$

$$jy_s = \begin{cases} A_1^{(s)} & \text{if } 2 \leq s \leq n-2 \ \& \ n-s \leq j \leq n\}, \\ \{n\} & \text{if } s = 1 \ \& \ j \in \{n-1, n\}, \\ A' & \text{otherwise.} \end{cases}$$

Obviously, $\mathcal{A}_n$ is a trapped c.n.d. automaton, its trap is the state 1. Let us consider the word $w \in X^{(n)*}$, given by

$$w = y_{n-2}x_1^{(n-2)} \dots x_{\binom{n-1}{n-2}-1}^{(n-2)} y_{n-3} \dots y_2 x_{1,2}^{(2)} \dots x_{\binom{n-1}{2}-1}^{(2)} y_1 v_{n-1} \dots v_1 .$$

It is easy to check that w is a D1-directing word of $\mathcal{A}_n$, namely $[n]w = \{1\}$. Moreover, w is the unique minimum-length D1-directing word of $\mathcal{A}_n$. This fact is based on the following observation.

If px is a prefix of w, then for any $x' \in X^{(n)}$, different from x, there exists a prefix q of p such that $[n]px' = [n]q$.

Since w is a minimum-length D1-directing word of $\mathcal{A}_n$ and its length is equal to $2^{n-1} - 1$, we obtain $2^{n-1} - 1 \leq \mathrm{cd}_1^{\mathrm{T}}(n)$.

Regarding the upper bound, let us observe that if $w = x_1 \dots x_m$ is a minimum-length D1-directing word of a trapped c.n.d. automaton $\mathcal{A} = (A, X)$ of $n(> 1)$ states with a trap $a_n \in A$, then $Aw = \{a_n\}$. Moreover, the sequence $A, Ax_1, \dots, Ax_1 \dots x_m$ consists of pairwise different nonempty subsets of A and each of them contains a_n. The number of these subsets is at most 2^{n-1}, and so, the length of w is not greater than $2^{n-1} - 1$. Hence, we obtain that $\mathrm{cd}_1^{\mathrm{T}}(n) \leq 2^{n-1} - 1$. The statement is obviously also valid for $n = 1$. This ends the proof of Theorem 3. □

In what follows, we shall use the following observation.

Lemma. *For every $n \geq 1$, $\mathrm{cd}_1^{\mathrm{T}}(n) = \mathrm{cd}_2^{\mathrm{T}}(n)$ and $\mathrm{d}_1^{\mathrm{T}}(n) = \mathrm{d}_2^{\mathrm{T}}(n)$.*

Proof. Let us observe that for any trapped n.d. automaton $\mathcal{A} = (A, X)$ of n states, $\mathrm{D}_1(\mathcal{A}) = \mathrm{D}_2(\mathcal{A})$. Indeed, $\mathrm{D}_1(\mathcal{A}) \subseteq \mathrm{D}_2(\mathcal{A})$ follows from the definition. Now, let $w \in \mathrm{D}_2(\mathcal{A})$. Then $aw = bw$ for every pair of states. This yields that $\{a_n\} = a_n w = aw$ is valid for any state $a \in A$, where a_n denotes the trap state of $\mathcal{A}$. This means that $w \in \mathrm{D}_1(\mathcal{A})$, implying $\mathrm{D}_2(\mathcal{A}) \subseteq \mathrm{D}_1(\mathcal{A})$. Therefore, $\mathrm{D}_1(\mathcal{A}) = \mathrm{D}_2(\mathcal{A})$. From this equality it follows that $\mathrm{cd}_1^{\mathrm{T}}(n) = \mathrm{cd}_2^{\mathrm{T}}(n)$ and $\mathrm{d}_1^{\mathrm{T}}(n) = \mathrm{d}_2^{\mathrm{T}}(n)$. □

Now, we can conclude the following statement from Theorem 3 by our Lemma.

Theorem 4. *For any $n \geq 1$, $\mathrm{cd}_2^{\mathrm{T}}(n) = 2^{n-1} - 1$.*

For D1- and D2-directable trapped n.d. automata, we have the following bounds.

Theorem 5. *For any $n \geq 1$, $2^{n-1} - 1 \leq \mathrm{d}_1^{\mathrm{T}}(n) = \mathrm{d}_2^{\mathrm{T}}(n) \leq 2(2^{n-1} - 1)$.*

Proof. $\mathrm{d}_1^{\mathrm{T}}(n) = \mathrm{d}_2^{\mathrm{T}}(n)$ is provided by our Lemma. By Theorem 3, we have that $2^{n-1} - 1 \leq \mathrm{cd}_1^{\mathrm{T}}(n)$. On the other hand, $\mathrm{cd}_1^{\mathrm{T}}(n) \leq \mathrm{d}_1^{\mathrm{T}}(n)$, and therefore, $2^{n-1} - 1 \leq \mathrm{d}_1^{\mathrm{T}}(n)$.

Regarding the upper bound, let us consider an arbitrary D1-directable trapped n.d. automaton $\mathcal{A} = (\{a_1, \ldots, a_n\}, X)$ with $n \geq 2$. Without loss of generality, we may suppose that a_n is the trap of $\mathcal{A}$. First, let us observe that $\mathcal{A}$ is a D3-directable n.d. automaton, as well. Let w_1 be a minimum-length D3-directing word of $\mathcal{A}$. By Theorem 2, $|w_1| \leq 2^{n-1} - 1$. Since $\mathcal{A}$ is a trapped n.d. automaton, $a_n \in a_j w_1$, for all $j \in [n]$. Then for every $j \in [n]$ and $p \in X^*$, $a_j w_1 p \neq \emptyset$. Now, let $w_2 = x_1 \ldots x_m$ be a minimum-length word such that $Aw_2 = \{a_n\}$. Such a word there exists since $\mathcal{A}$ is D1-directable. Let us consider the sequence $A, Ax_1, \ldots, Ax_1 \ldots x_m$. We show that these sets are pairwise different. If it is not so, then there are integers $0 \leq r < s \leq m$ such that $Ax_1 \ldots x_r = Ax_1 \ldots x_s$. Then $Ax_1 \ldots x_r x_{s+1} \ldots x_m = \{a_n\}$ which is a contradiction. Since $a_n \in Ap$ for every prefix p of w_2, we obtain that $m \leq 2^{n-1} - 1$. Now, we prove that $w_1 w_2$ is a D1-directing word of $\mathcal{A}$. Let $j \in [n]$ be arbitrary. Then $a_n \in a_j w_1$ and $a_j w_1 \subseteq A$. Moreover, $a_j w_1 w_2 \neq \emptyset$ and $a_j w_1 w_2 \subseteq Aw_2 = \{a_n\}$, and hence, $a_j w_1 w_2 = \{a_n\}$. On the other hand, $|w_1 w_2| = 2^{n-1} - 1 + 2^{n-1} - 1 = 2(2^{n-1} - 1)$. Consequently, $\mathrm{d}_1^{\mathrm{T}}(n) \leq 2(2^{n-1} - 1)$ if $n \geq 2$. On the other hand, $\mathrm{d}_1^{\mathrm{T}}(n) \leq 0$ is obvious. This completes the proof of Theorem 5. □

Remark 2. Since $\mathrm{cd}_2^{\mathrm{T}}(n) \leq \mathrm{cd}_2(n) \leq \mathrm{d}_2(n)$, we obtain that $2^{n-1} - 1$ is a lower bound for both $\mathrm{cd}_2(n)$ and $\mathrm{d}_2(n)$. On the other hand, the known lower bound, given for $\mathrm{cd}_2(n)$ and $\mathrm{d}_2(n)$ in [8], is $\lfloor n^{\frac{1}{3}} - 1 \rfloor!$ which is less than $2^{n-1} - 1$. Therefore, $2^{n-1} - 1$ is an improvement of the lower bounds of both $\mathrm{cd}_2(n)$ and $\mathrm{d}_2(n)$.

Remark 3. The upper bound, presented in Theorem 5, is worse than the upper bound $2^n - n - 1$, given for $\mathrm{d}_1(n)$ in [8]. The verification of the inequality $\mathrm{d}_1(n) \leq 2^n - n - 1$ is based on the observation that if $w = x_1 \ldots x_m$ is a minimum-length D1-directing word of an n.d. automaton $\mathcal{A} = (A, X)$, then the sets $A, Ax_1, \ldots, Ax_1 \ldots x_m$ must be pairwise different. The following example shows that this observation is not valid, moreover, $2^n - n - 1$ is not necessarily upper bound for $\mathrm{d}_1(n)$ in general. Let $\mathcal{A} = (\{0, 1)\}, \{x, y\})$, where $0x = \{0, 1\}$, $1x = \{1\}$, $0y = \emptyset$, and $1y = 1$. Then xy is a minimum-length D1-directing word of $\mathcal{A}$, but $\{0, 1\}x = \{0, 1\}$. Moreover, $2 = |xy| \not\leq 2^2 - 2 - 1$.

References

[1] H. D. Burkhard, Zum Längenproblem homogener Experimente an determinierten und nicht-deterministischen Automaten, *Elektronische Informationsverarbeitung und Kybernetik, EIK* **12** (1976), 301-306.

[2] A. Carpi, On synchronizing unambigous automata, *Theoretical Computer Science* **60** (1988), 285-296.

[3] J. Černý, Poznámka *k* homogénnym experimentom s konečnými automatmi, *Matematicko-fysikalny Časopis SAV* **14** (1964), 208–215.

[4] J. Černý, A. Piricka, B. Rosenauerova, On directable automata, *Kybernetika (Praha)*, **7** (1971), 289-297.

[5] D. Eppstein, Reset sequences for monotonic automata, *SIAM Journal of Computing* **19** (1990), 500-510.

[6] P. Goralčik, Z. Hedrlin, V. Koubek, J. Ryšlinková, A game of composing binary relations, *R.A.I.O. Informatique théorique/Theoretical Informatics* **16** (1982), 365-369.

[7] B. Imreh, M. Steinby, Some remarks on directable automata, *Acta Cybernetica* **12** (1995), 23–35.

[8] B. Imreh, M. Steinby, Directable nondeterministic automata, *Acta Cybernetica* **14** (1999), 105-115.

[9] T. Petković, M. Ćirić, S. Bogdanović, Decompositions of automata and transition semigroups, *Acta Cybernetica* **13** (1998), 385–403.

[10] J.-E. Pin, Sur un cas particulier de la conjecture de Cerny, *Automata, languages and programming*, ICALP'79 (Proc. Coll., Udine 1979), LNCS 62, Springer-Verlag, Berlin, 1979, 345–352.

[11] Z. Popović, S. Bogdanović, T. Petković, M. Ćirić, Trapped automata, *Publicationes Mathematicae*, **60** (2002), 661–677.

[12] I. Rystsov, Reset words for commutative and solvable automata, *Theoretical Computer Science* **172** (1997), 273-279.

Received November, 2002

On variable sized vector packing

Leah Epstein[*]

Abstract

One of the open problems in on-line packing is the gap between the lower bound $\Omega(1)$ and the upper bound $O(d)$ for vector packing of d-dimensional items into d-dimensional bins. We address a more general packing problem with variable sized bins. In this problem, the set of allowed bins contains the traditional "all-1" vector, but also a finite number of other d-dimensional vectors. The study of this problem can be seen as a first step towards solving the classical problem. It is not hard to see that a simple greedy algorithm achieves competitive ratio $O(d)$ for every set of bins. We show that for all small $\varepsilon > 0$ there exists a set of bins for which the competitive ratio is $1+\varepsilon$. On the other hand we show that there exists a set of bins for which every deterministic or randomized algorithm has competitive ratio $\Omega(d)$. We also study one special case for $d = 2$.

1 Introduction

The problem. We consider the following problem. We are given a finite set B of d-dimensional vectors in $[0,1]^d$. This is the set of bin sizes. The "all-1" vector $(1,1,\dots,1)$ belongs to B. Items of sizes also in $[0,1]^d$ arrive on-line, to be assigned to bins of sizes in B. The packing needs to be valid, i.e. the vector sum of all items assigned to one bin cannot exceed the capacity of the bin (in any component). The "all-1" bin needs to be in B so that every item can fit into some bin. Each item, has to be assigned to a bin upon arrival, and cannot be moved after that. It can be assigned either to an open bin, or to a new bin of some size in B. The cost of a bin b is the sum of its components and the weight of an item is the sum of its components. The goal is to minimize the total cost of the bins that the algorithm uses.

Applications. The problem can be seen as a scheduling problem with limited resources. There are a few types of machines (the bins) with known and limited capacities of several resources as memory, running time, access to other computers etc. The items is this case are jobs that need to be run, each job requires a certain amount of each resource. Another application arises from viewing the problem as

[*]School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. Email: `lea@idc.ac.il`

Research supported in part by the Israel Science Foundation, (grant No. 250/01-1).

a storage allocation problem. Each bin has several qualities as volume, weight etc. Each item requires a certain amount of each quality.

In both applications it is likely for the items to arrive one by one, forcing the algorithm to make decisions without any knowledge of the future.

The quality measure. The competitive ratio of an on-line algorithm for this minimization problem is the worst case ratio, over all possible input sequences, of the cost of bins used by the on-line algorithm to the cost of bins used by an optimal off-line algorithm (which is familiar with the complete sequence in advance). Often an additive constant is allowed, yielding the following definition of the competitive ratio.

Definition 1.1. *For an on-line algorithm and a sequence I of items, let $C_{ONL}(I)$ be the cost of the bins used by the on-line algorithm and let $C_{OPT}(I)$ be the cost of the bins used by an optimal off-line algorithm. ($C_{ONL}(I)$ can be abbreviated by C_{ONL} and $C_{OPT}(I)$ can be abbreviated by C_{OPT}.) Let $R \geq 1$.*

An on-line algorithm is R-competitive *if there exists a constant b such that*
$$C_{ONL}(I) \leq R \cdot C_{OPT}(I) + b, \text{ for any sequence } I \text{ of items.}$$
The competitive ratio of an on-line algorithm is
$$r = \inf\{R \mid \text{the on-line algorithm is } R\text{-competitive}\}.$$

If the additive constant b is zero or negative, the algorithm is called *strictly R-competitive.* The negative results given in this paper are valid for the strict competitive ratio as well as for the competitive ratio in general. The positive results are valid for the general competitive ratio, as it is common for bin packing type problems.

For randomized algorithms, the competitive ratio is defined similarly, but $C_{ONL}(I)$ is replaced by $E(C_{ONL}(I))$.

A simple algorithm. The following algorithm achieves competitive ratio at most $2d$ for every set B. Hence the best competitive ratio for any set is $O(d)$. The algorithm uses only "all-1" bins, and packs the items in a "next-fit" fashion. It keeps one open active bin where all arriving items are packed, whenever an arriving item does not fit, this bin is closed and a new active bin is opened. To show the competitive ratio, partition all bins used by the algorithm into pairs, according to the order they were opened. (If the number of bins is odd, the last one is ignored). Now combine the contents of each pair. The items in the two bins could not fit into one bin, since the second is opened when an item does not fit into the first. Hence, at least one component of the combined contents is at least 1. Let W be the total weight of items. Let X be the number of bins used by the algorithm. Then $W \geq (X-1)/2$. The optimal off-line also need to pack the items hence $C_{OPT} \geq W$. Since $C_{ONL} = Xd$, we conclude that the algorithm is $2d$-competitive.

Previous work. To the best of our knowledge, no results exist on variable-sized vector packing. We mention the results for the classical on-line vector packing problem, where B consists of a single, all-1 bin. There is only a small number of results on on-line vector packing, and the problem seems to be difficult. Kou and Markowsky [10] considered a class of algorithms for which there never exists a pair of bins whose contents can be combined legally into a single bin. They showed that any algorithm in this class is $d+1$ competitive. Later [7] improved the analysis for a d-dimensional version of first-fit to $d+0.7$. The lower bounds [6, 1] are all below 2, tending to 2 as d goes to infinity. The best lower bound for $d=2$ is 1.6712 given by Blitz, Van Vliet, Woeginger in [1]. This large gap between the negative and positive results (for large d as well as for $d=2$) encouraged the current study of the model with variable sized bins. The main questions were as follows: Are there any examples for B where the competitive ratio is linear? Are there any examples where the competitive ratio is constant?

The results. We answer both questions positively. Specifically, we show a set B for which we design a constant competitive algorithm, moreover, for any small $\varepsilon > 0$, we give an algorithm of competitive ratio $1+\varepsilon$ (section 2). We further design a set for which we show a lower bound of $\Omega(d)$ on the competitive ratio of deterministic and randomized algorithms (section 3). In section 4, we design an algorithm for a special case $d=2$ and $|B|=3$. This algorithm demonstrates the simplicity in which it is possible to reduce the competitive ratio just by adding a small number of bins to B.

Other related work. A survey on on-line bin packing problems is given in [3]. The one dimensional variable-sized bin packing problem was studied in several papers [5, 9, 13, 2, 11, 4]. Those papers present and analyze various algorithms. Csirik [2] showed that there exists a choice of a set of bins (containing two bins of sizes 1 and 0.7) so that the competitive ratio (1.4) is much lower than the best known lower bound (1.5401 [12]) for the basic bin packing problem (a single type of bin which has size 1). In [4] improved upper bounds and new lower bounds for sets of two bins are given. The overall upper bound on the competitive ratio presented in that paper is $373/228 \approx 1.63596$.

2 A set with $1+\varepsilon$ approximation

In this section we introduce a bin set B for which an asymptotic competitive ratio, arbitrarily close to 1 is achieved by deterministic algorithms. Even though the algorithm is on-line, the methods are somewhat similar to those used in design of polynomial approximation schemes for off-line scheduling problems (see e.g. [8]).

Let $0 < \varepsilon < 1/3$ be a small positive constant. Let $\Delta = \lceil \frac{d}{\varepsilon} \rceil$. Let $\delta = 1/\Delta$. We define the set B_ε of allowed bins, as the set of the vectors $(a_1\delta^2, a_2\delta^2, \ldots, a_d\delta^2)$ such that all a_i are integer, and $0 \leq a_i \leq \Delta^2$ for $1 \leq i \leq d$. The number of different bins is at most $(\Delta^2+1)^d = \Theta((\frac{d}{\varepsilon})^{2d})$. Note that taking $a_i = \Delta^2$ for all $1 \leq i \leq d$ gives the "all-1" bin.

We define an algorithm which has asymptotic competitive ratio $1+\varepsilon$ for the set B_ε of bins. An item is called *senior*, if it has at least one component of size at least δ, and *junior* otherwise. Senior items and junior items are packed by different methods. In both cases there is very little room left in the bins (apart from a constant number of bins) and the competitive ratio is proved by area considerations.

Senior items: Each senior item is packed into a separate bin. Given an item $x = (x_1, \ldots, x_d)$, x is packed into a bin b such that if $b = (b_1, \ldots, b_d)$ then $b_i \geq x_i$ but $b_i - \delta^2 < x_i$. In other words, $b_i = \lceil \frac{x_i}{\delta^2} \rceil \delta^2$.

Junior items: Those items are packed only into a subset of B_ε. Let $B_\varepsilon(i)$ be the subset of B_ε containing all vectors whose component i equals 1. Throughout the algorithm, for each $1 \leq i \leq d$ there is one open bin of each size in $B_\varepsilon(i)$, which is used for junior items whose largest component has index i. Given a junior item $x = (x_1, \ldots, x_d)$, let $m = \max_{1 \leq i \leq d} x_i$, let j be the minimum index of a component which achieves the maximum (i.e. $x_j = m$ and $x_k < m$ for $k < j$). Let $y = x/m$. The item is assigned into an open bin of size $b' = (b'_1, \ldots, b'_d)$ in $B_\varepsilon(j)$ such that $b'_i = \lceil \frac{y_i}{\delta^2} \rceil \delta^2$. Note that component j of b' is 1. If the item does not fit into the open bin, this bin is closed, a new bin of size b' associated with $B_\varepsilon(j)$ is opened and the item is assigned there. We are going to show that for every bin used by our algorithm (apart from the last bin of every size in $B_\varepsilon(i)$ for every $1 \leq i \leq d$, that is used for junior items), the cost of a bin is at most $1 + \varepsilon$ times the weight of items assigned to this bin. This would give

$$C_{ONL} \leq (1 + \varepsilon)W + d|B_\varepsilon| \qquad (1)$$

where W is the total weight of items in the sequence. Since $C_{OPT} \geq W$ and $|B_\varepsilon|$ is constant (which depends on d and ε), this gives an asymptotic competitive ratio $1 + \varepsilon$ for a constant ε. We analyze senior and junior items separately.

Bins with a senior item: According to the definition of the algorithm, each bin contains a single item x. Let w_b the weight of the bin b where x is assigned and w_x be the weight of x. By the algorithm $w_b \leq w_x + d\delta^2$. Since x is a senior item, $w_x \geq \delta$, hence $w_b \leq w_x(1 + d\delta) \leq w_x(1 + \varepsilon)$.

Bins with junior items: Consider a bin $\beta = (\beta_1, \ldots, \beta_d)$ that was used for junior items with maximal component of index k, and was closed during the algorithm. Let $y' = (y'_1, \ldots, y'_d)$ be the sum vector of items assigned to this bin..

We prove the following two claims.

Claim 2.1. *For all* $1 \leq i \leq d$, $\frac{y'_i}{\beta_i} \leq y'_k$, *and* $y'_k \geq 1 - \delta$.

Claim 2.2. *For all* $1 \leq i \leq d$, $y'_i \geq y'_k \beta_i - \delta^2$.

Before we prove the claims, we show this is sufficient to achieve the required competitive ratio. We need to compare w_β which is the cost of the bin β, to the weight of y', $w_{y'}$, which is the weight of the items in the bin. By Claim 2.2.

$$w_{y'} = \sum_{i=1}^{d} y'_i \geq y'_k \sum_{i=1}^{d} \beta_i - d\delta^2 \ .$$

Using this and the second part of Claim 2.1, we get $w_{y'} \geq w_\beta(1 - \delta - d\delta^2)$ (since $w_\beta \geq 1$). It is left to show that $1/(1 - \delta - d\delta^2) \leq 1 + \varepsilon$. Since $\delta \leq \varepsilon/d$ it is enough to show $(\varepsilon + 1)^2 \leq d$ which is true for $\varepsilon \leq \sqrt{2} - 1$ (since $d \geq 2$).

To complete the proof, we need to prove the claims. We start by proving Claim 2.1. Consider an item $\alpha = (\alpha_1, \ldots, \alpha_d)$ assigned to a bin β. Then let k be the maximal component of α which has the minimum index among the maximal components. Then α was assigned according to component k. Recall that β is calculated in the following way:

$$\beta_i = \lceil \frac{\alpha_i/\alpha_k}{\delta^2} \rceil \delta^2$$

and hence $\beta_i \geq \alpha_i/\alpha_k$. Since $\alpha_i/\beta_i \leq \alpha_k$ is true for all items in the bin, then also $y'_i/\beta_i \leq y'_k$. Now let γ be the item that caused this bin to be closed. γ did not fit into the bin, but since it was inserted to a bin of the same size also for $1 \leq i \leq d$, $\gamma_i/\gamma_k \leq \beta_i$. Since γ did not fit, for some component j, $y'_j + \gamma_j > \beta_j$. Hence $y'_k + \gamma_k \geq y'_j/\beta_j + \gamma_j/\beta_j > 1$. Since $\gamma_k < \delta$ (junior item) we get $y'_k > 1 - \gamma_k > 1 - \delta$. This proves Claim 2.1.

To prove Claim 2.2 recall that $\beta_i \leq \alpha_i/\alpha_k + \delta^2$. Hence $\alpha_i \geq \alpha_k\beta_i - \alpha_k\delta^2$. Let I be the set of all items assigned to β.

$$\begin{aligned}
y'_i = \sum_{\alpha\in I} \alpha_i &\geq \sum_{\alpha\in I} \alpha_k\beta_i - \sum_{\alpha\in I} \alpha_k\delta^2 = \\
(\beta_i - \delta^2)\sum_{\alpha\in I} \alpha_k &= y'_k(\beta_i - \delta^2) = \\
y'_k\beta_i - y'_k\delta^2 &\geq y'_k\beta_i - \delta^2.
\end{aligned}$$

The last inequality holds since $y'_k \leq 1$. This completes the proof of Claim 2.2.

Theorem 2.1. *The above algorithm has asymptotic competitive ratio of at most* $1 + \varepsilon$.

Proof. Follows from (1). □

3 A set with only $\Omega(d)$ approximations

In the introduction we showed that for every set, there exists an algorithm with competitive ratio $O(d)$. However, in the previous section we showed a set where it is possible to get an $1 + \varepsilon$ approximation. In this section we show a set for which we give a lower bound of $\Omega(d)$ on the competitive ratio.

We give a deterministic lower bound, and later show how to extend it to a randomized lower bound.

We start by a description of B. The set B contains apart from the vector $(1, \ldots, 1)$ also $2^{d/2}$ vectors which are called small bins. (We assume that d is even, for odd values of d it is possible to use the construction for the even dimension $d-1$, setting the last component to zero in all items, and in all the bins apart from the "all-1" bin.)

For $k = 1, \dots, d/2$, the $(2k-1)^{th}$ and the $(2k)^{th}$ components of the bins are either $\frac{1}{d^{2k}}$ and $\frac{1}{d^{2k-1}}$ or $\frac{1}{d^{2k-1}}$ and $\frac{1}{d^{2k}}$ (respectively). Since every pair has two options, there are $2^{d/2}$ such possible bins. Throughout the sequence, the optimal off-line cost is going to be $\Theta(n/d)$. There are $d/2$ phases of items, with n items in each. (We pick n to be a large constant so that the lower bound is valid also for general competitive ratio and not only strict competitive ratio.) All items have weight $\Theta(1/d^2)$. Note that all bin costs are $\Theta(1/d)$ (apart from the "all-1" bin whose cost is d).

We now define the items of phase i. In phase i, for all items and for all $j > 2i$, the j^{th} component is zero. The components $2i-1$ and $2i$ are both $\frac{1}{d^{2i}}$. For all $j < i$, the components $2j-1$ and $2j$ are either 0 and $\frac{2}{d^{2j}}$ or $\frac{2}{d^{2j}}$ and 0 (respectively). Note that there can fit only at most one item of each phase in a small bin. The choice of the $(2j-1)^{th}$ and the $(2j)^{th}$ coordinates is done according to the behavior of the algorithm until the completion of phase j.

We say that the algorithm "may use" a bin in phase j if the bin is opened in phase j or if it was opened during phases $1, ..., j-1$ and can still accommodate an item of phase j.

For $1 \le j \le \frac{d}{2}$, let N_j be the number of small bins, where the $(2j)^{th}$ component is $\frac{1}{d^{2j}}$, that the algorithm may use for items in phase j. (Not including bins opened after the arrival of the items of phase $j+1$.) If $j > 1$, in the beginning of phase j, some old bins cannot accommodate any more items due to a wrong structure of the $(2j-3)^{th}$ and the $(2j-2)^{th}$ components, those bins will never be used again. For $1 \le j \le \frac{d}{2}$, let M_j be the number of other small bins that may be used in phase j, that have the opposite structure of components $2j$ and $2j-1$ than bins counted in N_j. Note that the numbers M_j and N_j include new small bins opened during phase j, and previously opened bins that can still be used (the latter is true only for $j > 1$). If $N_j \le M_j$ then all future items have a zero in the $(2j)^{th}$ component and $\frac{2}{d^{2j}}$ in the $(2j-1)^{th}$ component. Otherwise, the structure is opposite. Hence all the M_j bins will never be used in the first case, and all N_j bins will never be used again in the second case. We use the following notations for $0 \le j \le d/2 - 1$. For $0 \le j \le \frac{d}{2} - 1$, let L_j be the number of small bins opened in phase $j+1$ and let S_j be the number of items assigned to an "all-1" bin in phase $j+1$. For $0 \le j \le \frac{d}{2} - 1$, let K_j be the total number of small bins that the algorithm may use in phase $j+1$. According to the above definitions, for $0 < j \le \frac{d}{2}$, $K_{j-1} = M_j + N_j$ and

$$K_j = L_j + \min(N_j, M_j) \le L_j + K_{j-1}/2 \ . \tag{2}$$

Since all items have either $\frac{1}{d^2}$ in both the first and the second components, or $\frac{2}{d^2}$ in one of the first two components, the algorithm can pack only at most d^2 items in one large bin. Hence $C_{ONL} \ge (\sum_{j=0}^{\frac{d}{2}-1} S_j)/d + 1/d \sum_{j=0}^{d/2-1} L_j$. We need to get a bound on those two sums. Note that in order to pack all items, for $j > 0$, $K_j + S_j \ge n$. Using (2) we get that $L_j + S_j \ge n - K_{j-1}/2$. $L_j + S_j$ represents the number of items that need to be either assigned to "all-1" bins, or have new bins opened for them.

On the other hand (2) gives the relations between the number of valid bins in phase $j+1$ to valid bins in phase j $(0 < j < d/2 - 1)$. Summing up the two equations for all $1 \le j \le \frac{d}{2} - 1$, and setting $L = \sum_{j=0}^{\frac{d}{2}-1} L_j$, $K = \sum_{j=0}^{\frac{d}{2}-1} K_j$, and $S = \sum_{j=0}^{\frac{d}{2}-1} S_j$, we get:

$$L - L_0 + S - S_0 \ge (\frac{d}{2} - 1)n - 1/2(K - K_{\frac{d}{2}-1}) .$$

and

$$K - K_0 \le L - L_0 + 1/2(K - K_{\frac{d}{2}-1}) .$$

Hence $L+S+\frac{1}{2}K \ge L_0+S_0+\frac{d}{2}n-n+1/2K_{\frac{d}{2}-1}$, and $L-K/2 \ge -K_0+L_0+1/2K_{\frac{d}{2}-1}$. Since $L_0 + S_0 \ge n$ (need to assign all items of phase 1), and all variables are non-negative, $L+S+K/2 \ge \frac{d}{2}n$. Since $K_0 = L_0$ then $L-K/2 \ge 0$. Hence $2L+S \ge \frac{d}{2}n$. We are interested in $(S + L)/d$. Easy substitution gives $C_{ONL} \ge (S + L)/d \ge (L + S/2)/d \ge n/4$.

On the other hand, the optimal off-line algorithm picks n small bins according to into which bin, an item of the last phase fits. Since $\frac{d}{2}(\frac{2}{d^{2j}}) = \frac{1}{d^{2j-1}}$, it is possible to place one item from each phase in a bin, the bin cost is $\Theta(1/d)$ and hence $C_{OPT} = \Theta(n/d)$. The competitive ratio follows.

To extend the proof for randomized algorithms, each one of the variables should be replaced by the expectation of this variable. By linearity of expectation, we get the same lower bound.

This proves the following Theorem.

Theorem 3.1. *There exists a finite set of bins, for which every deterministic or randomized algorithm for bin packing has competitive ratio $\Omega(d)$.*

4 A special case for $d = 2$

In this section we demonstrate by example, that letting the algorithm choose the set B, even if its size is very limited, allows the algorithm to improve the competitive ratio it achieves. In particular we consider $d = 2$ and $|B| = 3$. In section 1 it was shown that if $|B|$ is large enough (but finite), it is possible to achieve a very small competitive ratio. Here we focus on an example where $|B|$ is small, but a simple algorithm already improves on the best known algorithm for the classical case $B' = \{(1,1)\}$ given in [7] (whose competitive ratio is 2.7).

Let $B = \{(1,1),(1,\mu),(\mu,1)\}$. The constant $0 < \mu < 1/2$ is fixed later. We partition items into two classes:

- Items (β,γ) where $\beta \le \gamma$.
- All other items (i.e. items (β',γ') where $\beta' > \gamma'$).

Each one of the two classes is packed separately, independently from the other class. We explain how to pack the first class, the algorithm for the second class is symmetric (i.e. bins of size $(1,\mu)$ are used instead of bins of size $(\mu,1)$ and so on). The class is partitioned into six sub-classes. The algorithm also packs each one of the six sub-classes separately, independently from the other sub-classes. Let α be a constant $0 < \alpha \leq 1/3$, whose exact value is fixed later. The sub-class of an item (β, γ) is determined as follows:

- Sub-Class 1: $\gamma > 1/2$ and $\beta > \mu$.
- Sub-Class 2: $\gamma > 1/2$ and $\beta \leq \mu$.
- Sub-Class 3: $\alpha < \gamma \leq 1/2$ and $\beta > \mu/2$.
- Sub-Class 4: $\alpha < \gamma \leq 1/2$ and $\beta \leq \mu/2$.
- Sub-Class 5: $\gamma \leq \alpha$ and $\beta > \mu\gamma$.
- Sub-Class 6: $\gamma \leq \alpha$ and $\beta \leq \mu\gamma$.

Each item of sub-classes 1 and 2, is packed as an only item in a bin. For sub-class 1 the bin is of size $(1,1)$ and for sub-class 2 the bin is of size $(\mu, 1)$. Items of sub-classes 3 and 4 are packed in pairs (from the same sub-class). Pairs of sub-class 3 use bins of size $(1,1)$ whereas pairs of sub-class 4 use bins of size $(\mu, 1)$. By the definition of this class, (i.e. the conditions on γ and β), any two items of each of those sub-classes can fit into a bin together. The algorithm always has at most one bin for sub-class 3 with a single item. The same is true for sub-class 4 as well. The items of sub-class 5 are packed by a next-fit manner into bins of size $(1,1)$. The items of sub-class 6 are similarly packed into bins of size $(\mu, 1)$. In each of those two sub-classes, there is always one active bin. When an item does not fit, the bin is closed, and a new active bin is opened for the sub-class. Consider the sub-class 6. Given a set of items A, let $a = (a_1, a_2)$ be their sum vector. Then if $a_2 \leq 1$, $a_1 \leq \mu a_2 \leq \mu$. Hence $a_2 \leq 1$ is a satisfactory condition for all items in A to fit into one bin of size $(\mu, 1)$. For sub-classes 3 and 5, it is easy to see that the second component determines whether an item fits into a non-empty bin. This is true since for all items $\beta \leq \gamma$, but all the bins are of size $(1,1)$.

Next, we calculate the amount of occupied space in all closed bins. Those are all bins for sub-classes 1 and 2, and all bins but the very last ones opened for sub-classes 3, 4, 5 and 6. Those four last bins add an additive constant which is calculated later.

Sub-Class 1: The minimum weight of an item is $1/2 + \mu$ and the cost of a bin is 2.

Sub-Class 2: The minimum item weight is $1/2$ and the cost of a bin is $1 + \mu$.

Sub-Class 3: The weight of a pair of items is at least $2(\alpha + \mu/2) = 2\alpha + \mu$. The cost of the bin is 2.

Sub-Class 4: The weight of a pair of items is at least 2α, the cost of a bin is $1 + \mu$.

Before we proceed to the other two sub-classes, we discuss the way next-fit runs in those two cases. Consider a case where a new bin is opened, when an items does not fit into the previous active bin. By the above arguments, it means that the second component of an item can determine whether it fits. Since the second component is bounded by α, all closed bins are occupied by at least $1 - \alpha$ in that component. Bins of sub-phase 5 are also occupied by at least $(1 - \alpha)\mu$ in the first component.

Sub-Class 5: The weight of items in a closed bin is at least $(1 - \alpha)(1 + \mu)$, and

the cost of a bin is 2.
Sub-Class 6: The weight of items in a closed bin is at least $1-\alpha$, and the cost of a bin is $1+\mu$.

Let c be the maximum ratio of cost to weight in sub-classes 3, 4, 5 and 6. Specifically

$$c = \max(\frac{2}{2\alpha+\mu}, \frac{1+\mu}{2\alpha}, \frac{2}{(1-\alpha)(1+\mu)}, \frac{1+\mu}{1-\alpha}) .$$

Since $\alpha < 1/3$, we do not need to consider the fourth possibility. Hence

$$c = \max(\frac{2}{2\alpha+\mu}, \frac{1+\mu}{2\alpha}, \frac{2}{(1-\alpha)(1+\mu)}) .$$

Taking α to be a solution of $8x^3 - 8x^2 + 5x - 1 = 0$ and $\mu = (1-3\alpha)/\alpha$, all other values are the same (this gives $\alpha \approx 0.302$, $\mu \approx 0.315$, $c \approx 2.177$). Consider now also bins used for the symmetric case, i.e. the class of items (β', γ') where $\beta' > \gamma'$. There are at most 4 bins of cost $1+\mu$ and 4 bins of size 2 that might be open but not occupied by enough weight, and are ignored in previous calculations. We add those into the calculations to get the value of the additive constant.

Let N_L be the number of bins used for sub-class 1 (in both classes) and N_S the number of bins used for sub-class 2 (in both classes). Let W be the total weight of all items. Clearly, $C_{OPT} \geq W$. Also $C_{OPT} \geq N_L + N_S$. The last inequality is true since those items can be packed either alone (possibly together with items of other sub-classes) or in pairs, in bins of size $(1,1)$. Hence the cost of the optimal off-line algorithm for each such item is at least 1 (if all items of sub-classes 3, 4, 5 and 6 are ignored).

The weight of items packed into closed bins of sub-classes 3, 4, 5 and 6 is at most $W - 1/2N_S - (1/2+\mu)N_L$.

Hence

$$\begin{aligned} C_{ONL} &\leq c(W - 1/2N_S - (1/2+\mu)N_L) + (1+\mu)N_S + 2N_L + 12 + 4\mu \\ &\leq cC_{OPT} + N_S(1+\mu-c/2) + N_L(2-c/2-c\mu) + 12 + 4\mu \end{aligned}$$

For the above choices of α and μ,

$$1+\mu-c/2 = 2-c/2-c\mu \approx 0.226 .$$

Hence $C_{ONL} \leq 2.403C_{OPT} + 13.26$.

This proves the following Theorem:

Theorem 4.1. *The competitive ratio of the above algorithm is 2.403.*

5 Conclusions

We have seen that there is a large difference between possible competitive ratios for different sets, and that the competitive ratio can actually vary between 1 and $\Theta(d)$. The classical case ($B = \{(1,1\ldots,1)\}$) seems to be easier than the most difficult cases, but harder than the easiest cases. We conjecture that the competitive ratio for that problem should be non constant, but sub-linear.

References

[1] D. Blitz, A. Van Vliet, and G. J. Woeginger. Lower bounds on the asymptotic worst-case ratio of online bin packing algorithms. Unpublished manuscript, 1996.

[2] J. Csirik. An online algorithm for variable-sized bin packing. *Acta Informatica*, 26:697–709, 1989.

[3] J. Csirik and G. Woeginger. On-Line Packing and Covering Problems. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms*, volume 1442 of *LNCS*, chapter 7, pages 147–177. Springer-Verlag, 1998.

[4] L. Epstein, S. S. Seiden, and R. van Stee. New bounds for variable-sized and resource augmented online bin packing. submitted, 2001.

[5] D. K. Friesen and M. A. Langston. A storage-size selection problem. *Inform. Process. Lett.*, 18:295–296, 1984.

[6] G. Galambos, H. Kellerer, and G. J. Woeginger. A lower bound for online vector packing algorithms. *Acta Cybernetica*, 11:23–34, 1994.

[7] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. C. Yao. Resource constrained scheduling as generalized bin packing. *J. Comb. Th. Ser. A.*, 21:257–298, 1976.

[8] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. of the ACM*, 34(1):144–162, 1987.

[9] N. G. Kinnersley and M. A. Langston. Online variable-sized bin packing. *Discr. Appl. Math.*, 22:143–148, 1988.

[10] L. T. Kou and G. Markowsky. Multidimensional bin packing algorithms. *IBM J. Research and Development*, 21:443–448, 1977.

[11] S. S. Seiden. An optimal online algorithm for bounded space variable-sized bin packing. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 283–295, 2000.

[12] A. Van Vliet. An improved lower bound for online bin packing algorithms. *Inform. Process. Lett.*, 43:277–284, 1992.

[13] G. Zhang. Worst-case analysis of the FFH algorithm for online variable-sized bin packing. *Computing*, 56:165–172, 1996.

Received February, 2002

On-Line Maximizing the Number of Items Packed in Variable-Sized Bins

Leah Epstein[*] and Lene M. Favrholdt[†]

Abstract

We study an on-line bin packing problem. A fixed number n of bins, possibly of different sizes, are given. The items arrive on-line, and the goal is to pack as many items as possible. It is known that there exists a legal packing of the whole sequence in the n bins. We consider fair algorithms that reject an item, only if it does not fit in the empty space of any bin. We show that the competitive ratio of any fair, deterministic algorithm lies between $\frac{1}{2}$ and $\frac{2}{3}$, and that a class of algorithms including Best-Fit has a competitive ratio of exactly $\frac{n}{2n-1}$.

Keywords: On-Line, Bin Packing.

1 Introduction

The Problem. We consider the following bin packing problem. The input consists of n bins, possibly of different sizes, and a sequence of positively sized items. The bins as well as the sizes of the bins are denoted by $B_1, B_2, \ldots, B_n$. The items arrive on-line, i.e., each item must be packed before the next item is seen, and packed items cannot be moved between bins. The goal is to pack as many items as possible into the n bins. A bin is legally packed if the total size of the items assigned to it is at most the size of the bin. This problem of maximizing the number of items packed in a fixed number of bins is sometimes called *dual bin packing*, to distinguish it from the classical bin packing problem which is to pack *all* items in as *few* bins as possible. In [8] the problem is reported to have been named dual bin packing in [18]. Note that this name is also sometimes used for bin covering [2, 14, 15]. For a survey on classical bin packing in identical bins, see [16, 11].

---

[*]School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. Email: `lea@idc.ac.il`.
Research supported in part by the Israel Science Foundation, (grant No. 250/01-1)

[†]Department of Mathematics and Computer Science, University of Southern Denmark. Email: `lenem@imada.sdu.dk`.
Supported in part by the Danish Natural Science Research Council (SNF) and in part by the Future and Emerging Technologies program of the EU under contract number IST-1999-14186 (ALCOM-FT).

Throughout the paper, we restrict the input sequences to be *accommodating* [6, 7], i.e., sequences that an optimal off-line algorithm, which knows all items in advance, can pack completely. The reason for this restriction is that, for general sequences, no on-line algorithm can pack a constant fraction of the number of items that can be packed by an optimal off-line algorithm.

The problem can also be seen as a scheduling problem with n uniformly related machines. In the basic scheduling problem, each job is to be assigned to one of the machines so as to minimize the makespan. This problem was first studied for the case of identical machines by Graham [17], and for uniformly related machines by [1, 10, 4]. For a survey on on-line scheduling problems, see [20]. Consider a scheduling problem with a deadline and assume that the aim is to schedule as many jobs as possible before this deadline. If an optimal off-line algorithm can schedule all jobs of any input sequence before the deadline, this problem is equivalent to our problem. Our problem can also be seen as a special case of the multiple knapsack problem (see [19, 9]), where all items have unit profit. (This problem was mainly studied in the off-line environment.)

The Algorithms. In this paper we study fair algorithms [3]. A *fair* algorithm rejects an item, only if the item does not fit in the empty space of any bin.

Some of the algorithms that are classical for the classical bin packing problem (where the whole sequence of items is to be packed in as few bins as possible) can be adapted to our problem. Such an adaptation for identical bins was already done in [7]: the n bins are all considered open from the beginning, and no new bin can be opened. We also use this adaptation. Since there is no unique way to define First-Fit for variable sized bins, we discuss this in Section 3.

The Quality Measure. The competitive ratio of an on-line algorithm $\mathbb{A}$ for the dual bin packing problem is the worst case ratio, over all possible input sequences, of the number of items packed by $\mathbb{A}$ to the number of items packed by an optimal off-line algorithm. Often an additive constant is allowed, yielding the following definition of the competitive ratio.

Definition 1.1. *For any algorithm $\mathbb{A}$ and any sequence I of items, let $\mathbb{A}(I)$ be the number of items packed by $\mathbb{A}$ and let $OPT(I)$ be the number of items packed by an optimal off-line algorithm. Furthermore, let $0 \le c \le 1$. An on-line algorithm $\mathbb{A}$ is* c-competitive *if there exists a constant b such that*

$$\mathbb{A}(I) \ge c \cdot OPT(I) - b, \text{ for any sequence } I \text{ of items.}$$

The competitive ratio *of $\mathbb{A}$ is*

$$C_{\mathbb{A}} = \sup\{c \mid \mathbb{A} \text{ is } c\text{-competitive}\}.$$

Note that since dual bin packing is a maximization problem, the competitive ratio lies between 0 and 1.

If the additive constant b is zero or negative, the algorithm is called *strictly c-competitive.* The bounds given in this paper are valid for the strict competitive ratio as well as for the competitive ratio in general.

For randomized algorithms, the competitive ratio is defined similarly, but $\mathbb{A}(I)$ is replaced by the expected value of $\mathbb{A}(I)$, $E(\mathbb{A}(I))$.

The Results. We show the following results for fair algorithms on accommodating sequences.

- Any fair algorithm has a competitive ratio of at least $\frac{1}{2}$, and the competitive ratio of Worst-Fit is exactly $\frac{1}{2}$.
- A class of algorithms that give preference to smaller bins has a competitive ratio of exactly $\frac{n}{2n-1}$. This class contains Best-Fit as well as the variant of First-Fit that sorts the bins in order of non-decreasing sizes.
- Any fair, deterministic algorithm has a competitive ratio of at most $\frac{2}{3}$, and any fair, randomized algorithm has a competitive ratio of at most $\frac{4}{5}$.

Previous Work. Dual bin packing in identical bins has been studied both in the off-line version [13, 12] and in the on-line version for accommodating sequences [6, 7, 3]. Even for identical bins, a restriction on the input sequences is needed in order to be able to achieve a constant competitive ratio [7]. In [7], fair algorithms are considered and it is shown that First-Fit has a competitive ratio of at least $\frac{5}{8}$ on accommodating sequences. An upper bound of $\frac{6}{7}$ for any fair or unfair randomized algorithm is also given. In [3], a $(\frac{2}{3} - \frac{2}{4n+1})$-competitive unfair algorithm is given, the negative result for fair deterministic algorithms is improved to 0.809, and the bound of $\frac{5}{8}$ for First-Fit is shown to be asymptotically tight (the upper bound approaches $\frac{5}{8}$ as n approaches infinity).

2 General Results on Fair Algorithms

In this section we show that, on accommodating sequences, the competitive ratio of any fair, deterministic algorithm lies between $\frac{1}{2}$ and $\frac{2}{3}$, and the competitive ratio of any randomized algorithm is at most $\frac{4}{5}$.

2.1 Positive Results

The main result of this section is that any fair algorithm is $\frac{1}{2}$-competitive on accommodating sequences. We need the following lemma which is adapted from a similar lemma for identical bins in [7].

Lemma 2.1. *For any fair algorithm, the number of rejected items is no larger than the number of accepted items, if the input sequence is accommodating.*

Proof. Given an instance of the dual bin packing problem with an accommodating sequence I, we define a sequence I' as follows. Each accepted item of size x is replaced by $\lfloor \frac{x}{s} \rfloor$ items of size s, where s is the minimum size of any rejected item. Each rejected item is decreased to have size s. Clearly, a packing of all items of I defines a legal packing of all items of I', hence I' is also an accommodating sequence.

Let P be the on-line packing of I and let P' be the packing of I' induced by P. Note that all items of I' have the same size. Thus, to calculate an upper bound on the number of items rejected we just need to find an upper bound on the number of items of size s that fit in the bins after doing the packing P'.

For each bin B_i, let k_i denote the number of items in bin B_i in the packing P. The empty space in B_i in the packing P' consists of the empty space in B_i in the packing P and the space freed by the rounding down of the items packed in B_i. The empty space in B_i in P is less than s, since the algorithm is fair, and the total size of each original item was decreased by less than s. Thus, the empty space in B_i in P' is strictly less than $s(k_i + 1)$. We conclude that the number of rejected items is at most $\sum_{i=1}^{n} k_i$ which is the number of accepted items. □

Corollary 2.1. *Any fair algorithm has a competitive ratio on accommodating sequences of at least $\frac{1}{2}$.*

We close this section with an easy lemma that will be needed in Section 2.2 and Section 3. Let C be the set of non-empty bins in the optimal off-line packing. Let $N = |C|$.

Lemma 2.2. *Given an accommodating input sequence, any fair algorithm rejects at most $N - 1$ items.*

Proof. If the on-line algorithm does not reject any items, its packing is optimal. Assume now, that at least one item is rejected. Let s be the minimum size of any rejected item. Since the algorithm is fair, the empty space in each bin is less than s. Another trivial upper bound on the empty space in any bin B is the size B of the bin. Thus, the total empty space in the on-line packing is strictly less than $Ns + \sum_{B \notin C} B$. The total empty space of OPT is at least $\sum_{B \notin C} B$. Hence, since OPT accepts all items, the total size of all rejected items is strictly less than Ns. Since all rejected items are of size at least s, there are at most $N - 1$ rejected items. □

2.2 Negative Results

In this section we show an upper bound of $\frac{2}{3}$ for deterministic, fair algorithms and an upper bound of $\frac{4}{5}$ for randomized, fair algorithms.

We first prove the upper bound of $\frac{2}{3}$ for the strict competitive ratio. This is relatively easy for any $n \geq 2$. Consider for example the following instance with $n - 2$ bins of size ε, $0 < \varepsilon < 1$, one bin of size 2, and one bin of size 3. The input sequence consists of two or three items that are all too large for the bins of size ε. The first item has size 1. If this first item is assigned to the bin of size 3, an item of size 3 arrives next. Otherwise, two items of size 2 will arrive. In the first case, only the first item is packed, since the second does not fit, and in the second case only two items are accepted, the third does not fit. It is easy to see that both sequences are accommodating. This gives an upper bound of $\frac{2}{3}$ on the strict competitive ratio, for $n \geq 2$. Applying Yao's inequality [21] as described in [5] on these two sequences gives an upper bound of $\frac{4}{5}$ on the strict competitive ratio for randomized algorithms. This can be seen in the following way. Consider the sequence where the first item of size 1 is followed by one item of size 3 with probability $p_1 = \frac{2}{5}$ and by two items of size 2 with probability $p_2 = \frac{3}{5}$. An algorithm that packs the first item in the bin of size 3 will have an expected performance ratio of at most $p_1 \cdot \frac{1}{2} + p_2 \cdot 1 = \frac{4}{5}$. Similarly, an algorithm that packs the first item in the bin of size 2 will have an expected performance ratio of at most $p_1 \cdot 1 + p_2 \cdot \frac{2}{3} = \frac{4}{5}$. Thus, no deterministic algorithm can have an expected performance ratio larger than $\frac{4}{5}$ on this sequence.

However, we are interested in negative results that hold for the competitive ratio in general, and not only for the strict competitive ratio. By Lemma 2.2, the number of rejected items is at most $n-1$. As long as there is only a constant number of bins, we can view the number of rejected items as just an additive constant, and hence any fair algorithm has competitive ratio 1. Thus, to prove the following theorem, we need to find arbitrarily long accommodating sequences with the property that only $\frac{2}{3}$ of the items are accepted.

Theorem 2.1. *Any fair, deterministic on-line algorithm for the dual bin packing problem has a competitive ratio of at most $\frac{2}{3}$ on accommodating sequences.*

Proof. For $\ell = 1, \ldots, \lfloor \frac{n}{2} \rfloor$, we give the pair of bins

$$B_{2\ell-1} = 2\ell + 4^{\ell}\varepsilon \text{ and } B_{2\ell} = 2\ell + 2 \cdot 4^{\ell}\varepsilon,$$

where $\varepsilon < \frac{1}{4^n}$ is a positive constant. Thus, $4^{\ell}\varepsilon < 1$, $1 \leq \ell \leq \lfloor \frac{n}{2} \rfloor$. If n is odd, the last bin is of size $\frac{\varepsilon}{2}$ (so that no items are packed in that bin for the sequence we define). The sequence contains $3 \cdot \lfloor \frac{n}{2} \rfloor$ items and is constructed so that exactly $2 \cdot \lfloor \frac{n}{2} \rfloor$ of them are accepted.

The sequence is defined inductively in steps $\lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor - 1, \ldots, 1$. In step k, two large items are given and one small item is defined. The small items are given after all large items and are defined such that they will be rejected by the on-line algorithm. The sizes of the two large items are defined such that

- the on-line algorithm will pack them in B_{2k} and B_{2k-1}, one in each bin, and
- after packing the two items, the empty space in the two bins have the same size denoted E_k.

For convenience we define $E_{\lfloor \frac{n}{2} \rfloor + 1} = 0$. As will be seen later, $E_{k+1} < E_k$, $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$. Furthermore, we will prove that $E_1 < 1$.

The first large item given in step k has size $2k - E_{k+1}$. Thus, the very first item has size $2 \cdot \lfloor \frac{n}{2} \rfloor$, and the size of the first large item of each of the later steps depends on the empty space created in the previous step. Since $2k - E_{k+1} > 2k - 1$ and all previous bins $B_n, \ldots, B_{2k+1}$ have less than one unit of empty space, this item fits only in B_{2k} and B_{2k-1}. What happens next depends on which of these two bins the algorithm chooses.

Case 1: The first large item is packed in B_{2k-1}. In this case, the next large item has size $2k - E_{k+1} + 4^k\varepsilon$. This item will be packed in B_{2k}. Now, the empty space in each of the bins B_{2k} and B_{2k-1} is $E_k = E_{k+1} + 4^k\varepsilon$. The small item defined in this step has size $S_k = E_k + 4^k\varepsilon$. Note that this item does not fit in B_{2k} or B_{2k-1}, but the off-line algorithm can pack the first large item in B_{2k} together with the small item and put the second large item in B_{2k-1}.

Case 2: The first large item is packed in B_{2k}. In this case, the next large item has size $2k - E_{k+1} - 4^k\varepsilon$. For $k \geq 2$, this item does not fit in B_{2k-2}, since $2k - E_{k+1} - 4^k\varepsilon > 2k - 1 - 4^k\varepsilon \geq 2k - 2 + 3 \cdot 4^k\varepsilon$, for $n \geq 2$, and $B_{2k-2} = 2k - 2 + 2 \cdot 4^{k-2}\epsilon$. Hence, this item must be packed in B_{2k-1}. Now, the empty space in each of the bins B_{2k} and B_{2k-1} is $E_k = E_{k+1} + 2 \cdot 4^k\varepsilon$. The small item defined in this step has size $S_k = E_k + 4^k\varepsilon$. This item does not fit in B_{2k} or B_{2k-1}, but the off-line algorithm can pack the first large item in B_{2k-1} and put the second large item in B_{2k-1} together with the small item.

Note that $E_{k+1} + 4^k\varepsilon \leq E_k \leq E_{k+1} + 2 \cdot 4^k\varepsilon$, $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$. The first inequality tells us that, to prove that none of the small items will be accepted, it suffices to prove that $S_k > E_1$, $2 \leq k \leq \lfloor \frac{n}{2} \rfloor$. This is easily done using the second inequality. For $2 \leq k \leq \lfloor \frac{n}{2} \rfloor$,

$$E_1 \leq E_k + 2 \cdot \sum_{i=1}^{k-1} 4^i\varepsilon < E_k + 4^k\varepsilon = S_k.$$

Finally,

$$E_1 \leq E_{\lfloor \frac{n}{2} \rfloor + 1} + 2 \cdot \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} 4^i\varepsilon < 4^{\lfloor \frac{n}{2} \rfloor + 1}\varepsilon < 4^{\lfloor \frac{n}{2} \rfloor + 1 - n} \leq 1.$$

□

We move on to randomized algorithms. Since the previous sequence was built step by step, we need to give a simpler sequence in order to prove the following theorem.

Theorem 2.2. *Any fair randomized algorithm has a competitive ratio on accommodating sequences of at most $\frac{4}{5}$.*

Proof. We use $\lfloor \frac{n}{2} \rfloor$ bins of size $1 + \varepsilon$ and $\lfloor \frac{n}{2} \rfloor$ bins of size $2 - \varepsilon$, where $0 < \varepsilon < \frac{1}{2}$. If n is odd, the last bin is of size ε. The sequence starts with $\lfloor \frac{n}{2} \rfloor$ items of size 1. We describe a proof for deterministic algorithms first. Since the algorithm is fair, all $\lfloor \frac{n}{2} \rfloor$ items are accepted. Let x be the number of bins of size $1 + \varepsilon$ that received an item (no bin can receive more than one item). Then, exactly x bins of size $2 - \varepsilon$ are empty. What happens next depends on the size of x.

Case $x \leq \frac{3}{5} \cdot \lfloor \frac{n}{2} \rfloor$. In this case, the sequence continues with $\lfloor \frac{n}{2} \rfloor$ items of size $2 - \varepsilon$, and the on-line algorithm accepts $\lfloor \frac{n}{2} \rfloor + x$ items in total out of the $2\lfloor \frac{n}{2} \rfloor$. This gives a fraction of

$$\frac{\lfloor \frac{n}{2} \rfloor + x}{2\lfloor \frac{n}{2} \rfloor} \leq \frac{1 + \frac{3}{5}}{2} = \frac{4}{5}.$$

Case $x > \frac{3}{5} \cdot \lfloor \frac{n}{2} \rfloor$. In this case, the sequence continues with $\lfloor \frac{n}{2} \rfloor$ items of size $1+\varepsilon$ followed by $\lfloor \frac{n}{2} \rfloor$ items of size $1-\varepsilon$. After the arrival of items of size 1, there are $\lfloor \frac{n}{2} \rfloor$ empty bins. Thus, all items of size $1+\varepsilon$ are accepted and now each bin has exactly one item. Items of size $1-\varepsilon$ can only be assigned to bins of size $2-\varepsilon$ that contain an item of size 1, hence $\lfloor \frac{n}{2} \rfloor - x$ of them are accepted. Thus, the fraction

$$\frac{3\lfloor \frac{n}{2} \rfloor - x}{3\lfloor \frac{n}{2} \rfloor} < \frac{3-\frac{3}{5}}{3} = \frac{4}{5}$$

of the items is accepted.

To get a randomized result, let x be the expectation of the number of bins of size $1+\varepsilon$ that got an item. The bound follows by linearity of expectation. □

3 Results on Specific Fair Algorithms

We now analyze specific algorithms. Some natural fair algorithms are First-Fit, Best-Fit, and Worst-Fit. The algorithm First-Fit is not a single algorithm, but a class of algorithms that give an order to the bins, and use the algorithm according to this order, i.e., assign an item to the first bin (in the ordered set of bins) that the item fits in. Among the various versions of First-Fit, two are most natural; *Smallest-Fit* assigns an item to the smallest bin it fits into, and *Largest-Fit* assigns an item to the largest bin it fits into. The other algorithms are used in their classical version, i.e., Best-Fit packs each item in a bin where it will leave the smallest possible empty space, and Worst-Fit packs it in the bin where it leaves the largest empty space. We refer to these four algorithms as SF, LF, BF, and WF.

We start the analysis by showing that $\frac{1}{2}$ is indeed the exact competitive ratio of WF and LF.

Theorem 3.1. *The competitive ratio of Worst-Fit and Largest-Fit on accommodating sequences is* $\frac{1}{2}$.

Proof. Let $\varepsilon > 0$ be a constant such that $\varepsilon \leq \frac{1}{n}$. Consider the following set of bins. One large bin of size n and $n-1$ small bins of size 1. The sequence consists of $n-1$ items of size 1 followed by $n-1$ items of size $1+\varepsilon$. Both algorithms LF and WF assign all items of size 1 to the large bin. As a result, all bins have a free space of size 1, hence none of the items of size $1+\varepsilon$ can be accepted. The optimal algorithm assigns each small item to a small bin, and all other items to the large bin; they all fit since

$$(1+\varepsilon)(n-1) \leq \frac{(n+1)(n-1)}{n} < n\ .$$

This example in combination with Corollary 2.1 proves the theorem. □

We further analyze a class of fair algorithms called *Smallest-Bins-First* to which SF and BF belong. This is the class of fair algorithms that whenever an item is assigned to an empty bin, this is the smallest bin in which the item fits. There are no additional rules, and the algorithm may use an empty bin even if the item fits in a non-empty bin, as long as it uses the smallest empty bin for that. SF belongs to this class according to its definition. BF belongs to this class since, among the empty bins that an item fits into, it fits better into the smaller bins than the larger bins. We give a tight analysis of this class as a function of n. Specifically we prove the following.

Theorem 3.2. *The competitive ratio of any Smallest-Bins-First algorithm on accommodating sequences is* $\frac{n}{2n-1}$.

Proof. If, after running the algorithm, all bins of the on-line algorithm are non-empty, then there are at least n accepted items and at most $n-1$ rejected items (by Lemma 2.2). Thus, in this case, the competitive ratio is at least $\frac{n}{2n-1}$.

Otherwise, consider the largest (last) bin b that remained empty after running the on-line algorithm. We consider items of size smaller than or equal to b, and items larger than b separately. Since a bin of size b is empty and no bin larger than b is empty, according to the definition of the class of algorithms, each bin of size more than b contains at least one item larger than b, namely the first item packed in the bin. Moreover, all items of size at most b are accepted. Let x_s be the number of items in bins of size at most b and let n_ℓ be the number of bins larger than b. Let N_s be the number of non-empty bins of OPT of size at most b and N_ℓ its number of non-empty bins larger than b. Clearly, $x_s \geq N_s$ (all those bins are of size at most b and contain at least one item). We get that the number of accepted items is at least $x_s + n_\ell \geq N_s + N_\ell = N$. Thus, by Lemma 2.2, the competitive ratio is at least $\frac{N}{2N-1} \geq \frac{n}{2n-1}$.

To show that the result is tight for this class of algorithms, let $\varepsilon < \frac{1}{n}$ be a positive constant. Consider the set of bins $B_i = 1 + \varepsilon i$, $i = 1, \ldots, n$. The sequence consists of n items, one of size $1 + \varepsilon(i-1)$ for each $i = 1, \ldots, n$, followed by $n-1$ items of size $\frac{n\varepsilon}{n-1}$. All algorithms in the class assign the item of size $1 + \varepsilon(i-1)$ to B_i. All other items are rejected. The optimal off-line algorithm assigns each large item except the first one to a bin of its size. The first item and the $n-1$ small items are assigned to B_n. □

Note that when $n = 2$, the lower bound of $\frac{n}{2n-1}$ matches the general upper bound of $\frac{2}{3}$.

4 Conclusion

We have proven an upper bound of $\frac{2}{3}$ for all fair algorithms. We have also shown that any fair algorithm accepts at least half of the items, and that some algorithms do significantly better for very small n. It is left as an open problem to design a fair algorithm with a competitive ratio significantly larger than $\frac{1}{2}$ for any n, or prove that this is not possible. It is also unknown how much unfair algorithms can be better; the best negative result for those is $\frac{6}{7}$, which holds even for identical bins [7].

Acknowledgment: We would like to thank Joan Boyar for reading and commenting on the paper. We also thank the editor Gerhard Woeginger for suggesting the title.

References

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-Line Routing of Virtual Circuits with Applications to Load Balancing and Machine Scheduling. *Journal of the ACM*, 44(3):486–504, 1997. Also in *Proc. 25th ACM STOC*, 1993, pp. 623-631.

[2] S. F. Assmann, D. S. Johnson, D. J. Kleitman, and J. Y. Leung. On a Dual Version of the One-Dimensional Bin Packing Problem. *Journal of Algorithms*, 5:502–525, 1984.

[3] Y. Azar, J. Boyar, L. Epstein, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen. Fair versus Unrestricted Bin Packing. *Algorithmica*, 34(2):181–196, 2002. Preliminary version at SWAT 2000, volume 1851 of *LNCS*: 200-213, Springer-Verlag, 2000.

[4] P. Berman, M. Charikar, and M. Karpinski. On-Line Load Balancing for Related Machines. *Journal of Algorithms*, 35:108–121, 2000.

[5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[6] J. Boyar and K. S. Larsen. The Seat Reservation Problem. *Algorithmica*, 25:403–417, 1999.

[7] J. Boyar, K. S. Larsen, and M. N. Nielsen. The Accommodating Function: A Generalization of the Competitive Ratio. *SIAM Journal on Computing*, 31(1):233–258, 2001.

[8] J. L. Bruno and P. J. Downey. Probabilistic Bounds for Dual Bin-Packing. *Acta Informatica*, 22:333–345, 1985.

[9] C. Chekuri and S. Khanna. A PTAS for the Multiple Knapsack Problem. In *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 213–222, 2000.

[10] Y. Cho and S. Sahni. Bounds for List Schedules on Uniform Processors. *SIAM Journal on Computing*, 9:91–103, 1988.

[11] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation Algorithms for Bin Packing: A Survey. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 2, pages 46–93. PWS Publishing Company, 1997.

[12] E. G. Coffman Jr. and J. Y. Leung. Combinatorial Analysis of an Efficient Algorithm for Processor and Storage Allocation. *SIAM Journal on Computing*, 8(2):202–217, 1979.

[13] E. G. Coffman Jr. J. Y. Leung, and D. W. Ting. Bin Packing: Maximizing the Number of Pieces Packed. *Acta Informatica*, 9:263–271, 1978.

[14] J. Csirik and J. B. G. Frenk. A Dual Version of Bin Packing. *Algorithms Review*, 1:87–95, 1990.

[15] J. Csirik and V. Totik. On-Line Algorithms for a Dual Version of Bin Packing. *Discr. Appl. Math.*, 21:163–167, 1988.

[16] J. Csirik and G. Woeginger. On-Line Packing and Covering Problems. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms*, volume 1442 of *LNCS*, chapter 7, pages 147–177. Springer-Verlag, 1998.

[17] R. L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

[18] J. Y. Leung. *Fast Algorithms for Packing Problems.* PhD thesis, Pennsylvania State University, 1977.

[19] S. Martello and P. Toth. *Knapsack Problems.* John Wiley and Sons, Chichester, 1990.

[20] J. Sgall. On-Line Scheduling. In *A. Fiat and G. J. Woeginger, editors,* Online Algorithms: The State of the Art, volume 1442 of *LNCS*, pages 196–231. Springer-Verlag, 1998.

[21] A. C. Yao. Towards a Unified Measure of Complexity. *Proc. 12th ACM Symposium on Theory of Computing*, pages 222–227, 1980.

Received February, 2003

SW-type puzzles and their graphs

Benedek Nagy*

Abstract

In this paper, we present the SW-type of truth-tellers and liars puzzles. We examine the SW-type puzzles where each person can utter a sentence about the person's type and in which he uses only the "and" connective. We make the graphs of these puzzles. The graph of a puzzle has all information about the puzzle if we have no other information to solve the puzzle than the statements given (clear puzzles). We analyze the graphs of the possible puzzles. We give some transformations of graphs based on local information, for instance arrow-adding steps. These local steps are very helpful to solve these puzzles. We show an example that we can solve using these local steps. After this, we examine into the global properties of the graphs. We show a special example when the local steps do not help, but the puzzle is solvable by using global information. Finally we show a graph-algorithm which is a combination of local and global information, and show that it can solve the SW-type puzzles.

Keywords: puzzles, truth-tellers and liars, graphs, graph-algorithm

1 Introduction

Games are as old as humanity. Nowadays most people connect them to computers. Game playing is also good time-spending activities. The problems needing more or less time to solve represent useful ways of spending one's spare time. A part of games are puzzles. Logical puzzles can be solved by a rational way of thinking. From children to very wise people everybody can find puzzles which develop their skills. It can be a good hobby as well. Therefore, logical puzzles are very useful to explore the ability of logical thought. There are many kind of puzzles. In this article, we consider a simple type called "truth-tellers and liars". In these puzzles, there are some people each of the following two types: either truth-teller, who can say only true statements; or liar, who can say only false statements. All participants have full information about the type of the others. Some of them claim about the type of the others. The puzzle is to figure out the types of each person. These problems are very popular. Smullyan examined such puzzles in scientifical and logical way ([7], [8], [9], [10]), where the participants was distinguished as

*Institute of Mathematics and Informatics, University of Debrecen, H-4010 Debrecen, Pf. 12, Hungary. Email: nbenedek@math.klte.hu

knights and knaves. In [6], satellites send messages informing mechanics whether the neighbouring satellites work properly, or not. In [1], [2] and in [3], Aszalós solves many puzzles by using tableaux method and Prolog language, as well.

In this paper, we investigate a special type of truth-teller–liar puzzles. We show a characteristic example.

Example 1.1. Consider the following problem. There are four people: Alice, Bob, Charlie and David. Each of them is either a truth-teller or a liar. They say:

Alice says that Bob is a truth-teller and David is a liar. Bob says that Charlie is a liar. Charlie says that Alice is a truth-teller and David is a liar.

Using Smullyan's method [7] we write the example in the next logical form:

$$A \equiv B \wedge \neg D, \quad B \equiv \neg C, \quad C \equiv A \wedge \neg D$$

One can check easily that valuation $A = C = false$, $B = D = true$ gives valid formulae. Therefore a solution of the example that Alice and Charlie are truth-tellers; Bob and David are liars. Moreover it is easy to check that the variables A, B, C, D have not other values such that all given formulae are true. So our solution is unique.

In the next section we describe more precisely the SW-type truth-teller and liar puzzles. In [4] we presented a program in language C which can generate special SW-type puzzles. Now we will associate graphs to the puzzles, which are very useful to examine the structure of the puzzles, and we can solve a puzzle by using the associated graph.

In section 3 we analyse the SW-type puzzles and we show some useful steps to solve them by using local information, in this section we will solve Example 1.1.

In the next sections, we show an example such that the local information is not enough to solve it. We show how we can use the global information of the graph to get the solution. We present a general algorithm mixed the local and global information.

2 SW-type puzzles

We need a few concepts to the mathematical discussion and so; we give some basic definitions and notations.

The sentences which are not dividable to smaller sentences are called atomic (or simple) statements. In this paper, we use atomic statements only about a person's type as we seen in Example 1.1.

In Example 1.1 there are 5 atomic sentences. Alice and Charlie tell two-two atomic statements and Bob tells one.

In this puzzles, if a person is a truth-teller then the conjunction of his atomic statements must be true. If a person is a liar then the conjunction of his atomic statements is false. In [5] we used these definitions to define S(trong) Truth-tellers and W(eak) Liars.

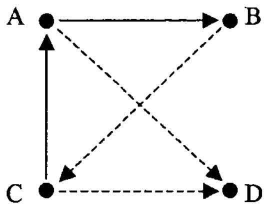

Figure 1: The graph of Example 1.1

If a person remains silent then he may be Truth-teller or he may be Liar. We may know it only from the atomic sentences about his type.

We call our puzzles SW-type because each person is an S truth-teller, or a W liar.

In [4] we investigated clear and non-clear puzzles. A puzzle is clear, if we have no other information to solve the puzzle than the statements given.

An example of a non-clear puzzle is someone's type or the number of truth-tellers being known independently of the statements. In this paper, we examine only clear puzzles. Our example is a clear SW-puzzle.

We use the value 1 to represent the truth-tellers, and the value 0 for the liars.

Definition 2.1. *The* solution *of a puzzle is a function which assigns either a 1 or a 0 to each person, who is in the puzzle depending upon the truth-values of the statements he or she makes. Two solutions are different, if there is a person, whose type is not the same in these two functions.*

We say that a puzzle is good *if it has a unique solution.*

In this paper we use three type of participants in dynamic way. The initial type is the unknown. The other two types are the known 1 and known 0. We will sign the known values at the participants who have it. Usually we will use the sets T and L as sets of truth-tellers and sets of liars, respectively.

This paper will investigate clear puzzles only with solutions and the most of our results are about good puzzles.

Definition 2.2. *Puzzles are represented by directed* graphs *with a node of the graph for person in the puzzle. There are two types of arrows: if A said that B is a truth-teller then we use a solid arrow from A to B; if A said that B is a liar then we use a dashed arrow from A to B. We will use $\boldsymbol{N}$ as the set of nodes and $\boldsymbol{t}, \boldsymbol{l}$ as the sets of solid and dashed edges, respectively.*

We will use the names of persons as names of nodes, and sometimes as logical statements which can have either 0 or 1 values. We use the following notation: $P(\mathbf{N},\mathbf{t},\mathbf{l})$ (where the nodes are $B_i \in \mathbf{N}$, the solid and dashed edges $t_j \in \mathbf{t}$ and $l_k \in \mathbf{l}$ respectively, and they are sorted pairs of nodes) as the associated graph of a puzzle.

In Fig. 1 we show the graph of Example 1.1.

We say that the puzzles P and Q are equivalent if the solution(s) of P are the same as the solution(s) of Q.

Now we continue the describing the SW-type puzzles, which are widely used. One can find them in almost every book on puzzles.

According to our concepts, in these puzzles we have only S truth-tellers, and W liars, hence we can use conjunction in logical form of the sentences. In an SW-type puzzle, every person can make at most one complex assertion which has a truth-value corresponding to the type of the person. The possible atomic sentences are: the man names a person, and he states that this person is a truth-teller (or he states, that this person is a liar).

So in an SW-type puzzle with n persons, each person can claim at most one sentence, and it must be the following type: he names m persons $(0 \leq m \leq n)$, and he states that all of them are truth-tellers, and he names k persons $(0 \leq k \leq n)$, and he states that all of them are liars. So, there are two – maybe one of them or both empty, and not necessarily disjoint – sets of the persons for each member in the puzzle, about whom that member claims something.

Definition 2.3. (Formal definition of SW-puzzles) *Suppose that there are n people $B_1, B_2, \ldots B_n$.*
From [7] and [4] we write the following logical form from the statements: if $\exists j (B_iB_j \in t$ or $B_iB_j \in l)$ then

$$(*) \qquad B_i \equiv \Big(\bigwedge_{B_iB_j \in t} B_j\Big) \wedge \Big(\bigwedge_{B_iB_k \in l} \neg B_k\Big).$$

If all the conditions of a puzzle can be written by this way then it is a clear SW-puzzle.

Now we consider that P is a graph of an SW-type puzzle. Let us examine the meaning of the arrows.

If A states the atomic statement, that B is a truth-teller then $A \supset B$ is valid (this is what the solid arrow means), or if A states, that B is a liar then the formula $A \supset \neg B$ is true (it is a dashed arrow). In $(*)$ we have equivalence, so we need one more concept: the relevant edges.

Definition 2.4. *In the graph P, we call an edge* relevant edge *if it is possible for it to stand for a liar's actual, false atomic statement.*

We will use this concept in dynamic way. First we assume, that all arrows are relevant. (We do not know yet if it was not possible for an edge to stand for a liar's false statement.) And while it turns out that an edge might not be relevant we assume that it is possible for it to stand for a liar's actual, false atomic statement. It is evident that there is at least 1 relevant arrow from the nodes which are type 0, assuming that person said something.

Notation 2.5. In the graph we cross the non-relevant edges. An edge is annotated with the sign '!' if there is only this relevant edge starting from that node.We will use these notations in figures. In form $P(\mathbf{N},\mathbf{t},\mathbf{l})$ we will use for a non-relevant edge AB ($A, B \in \mathbf{N}$) the sign $\overline{AB}$ (overlying). And for the unique relevant edge CD (where $C, D \in \mathbf{N}$) we use the $\underline{CD}$ (underlying) form in text instead of the sign '!'. From here $AB \in \mathbf{t}$ means that one of the AB, $\overline{AB}$ and $\underline{AB}$ is in the set $\mathbf{t}$, and similarly for the set $\mathbf{l}$.

The following assertion shows how we can use the relevant edges to make the $(*)$ form from the implications.

If there is only one relevant edge from a node B, then we use equivalence in the formula instead of implication: $\underline{BC} \in \mathbf{t}$ means that $B \equiv C$; $\underline{BC} \in \mathbf{l}$ means that $B \equiv \neg C$.

Hence we can see that the graph of a puzzle represents really the sets of logical formulas of the puzzle. The nodes (B_i) like atomic statements ("B_i is a truth-teller"), the arrows like logical connectives between them, and hence we can use them as logical statements also.

3 Local steps in the graph

Solving the puzzles requires some techniques which modify the graph of the puzzle. However our new graph is equivalent to the original. These local techniques are the following (we will give detailed description of them in this section):

Definition 3.1.

a) An arrow-adding step *is the following: add a new (non-relevant) arrow to the graph such that all solutions remain such that our new graph is equivalent to the original.*

b) A node-union *is when it turns out that two nodes must be same types, and we need use only one of them as common node.*

c) A subgraph is a basic scheme *if the type of one of the nodes of this part can be only one of* $\{0, 1\}$ *according to the arrows in this part.*

d) An arrow is a valuable arrow, *if we know the type of starting or the ending node, and we can infer the type of the other node (using only the information about the type of the first node, and the type of connection.)*

e) In arrow deletion *and* arrow change to irrelevant *steps we will delete the arrows, which are non useful (we cannot use them to get new information, for example we know the type of both end-nodes), or we cross the edges, which we cannot use as relevant edges, but we may will use as valuable arrows.*

The basic schemes and the valuable arrows change the types of the nodes to known value. The node-union step decrease the number of nodes and use the new node as endpoint of the edges, which had endpoint one of the joined nodes. In an arrow-adding step we increase the number of edges, while in an arrow deletion we decrease it.

Now we have some lemmas about these local steps: when and how we can use them.

First we show the arrow-adding steps. We add a new non-relevant edge to the graph in the following cases. (The new edge is non-relevant, because it is not a real statement in the original puzzle.)

Lemma 3.2. *The following steps are arrow-adding steps. The graphs before and after a step are equivalent.*

	edge(s) in the graph before the step	*the new irrelevant edge(s)*
a)	$AC \in \mathbf{l}$	$\overline{AC} \in \mathbf{l}$
b)	$AB, BC \in \mathbf{t}$	$\overline{AC} \in \mathbf{t}$
c)	$AB \in \mathbf{t}$ *and* $(BC \in \mathbf{l}$ *or* $CB \in \mathbf{l})^{\clubsuit}$	$\overline{AC}, \overline{CA} \in \mathbf{l}^{\diamondsuit}$
d)	$AB \in \mathbf{l}$ *and* $(\underline{BC}$ *or* $\underline{CB} \in \mathbf{l})^{\heartsuit}$	$\overline{AC} \in \mathbf{t}$.

If the new type of arrow with this direction has directly connected the nodes A and C, then we do not need the new edge (we already have edge which means this type of connection).

Proof. It is from the logical meanings of the edges. The case a) is from $A \supset \neg B$ is equivalent to $B \supset \neg A$, which is the meaning of the dashed arrow for opposite direction, case b) is from: $A \supset B$, $B \supset C$ are the original arrows, and their logical consequence is $A \supset C$, the new arrow. In case c) $\clubsuit$ means that we can use this step independently the direction of the dashed edge between B and C, because of point a), and $\diamondsuit$ signs that we can add arrows with both directions because we have $A \supset B$, $B \supset \neg C$, therefore $A \supset \neg C$ is valid, and than using point a) $C \supset \neg A$ also valid. In case d) the sign $\heartsuit$ notes that the dashed edge between B and C must be only one relevant at least from one direction and we have no restriction about the relevance of other edges in these steps. Then $A \supset \neg B$, $B \equiv \neg C$, which is implies that $C \equiv \neg B$, hence $A \supset C$. $\square$

The meanings of these new edges are about "that person could say these things also". In [3] Aszalós examines this modal operator in puzzles.

According to the point a) of the previous lemma, if the relevance is not important then we can use only dashed line instead arrows. (But sometime we need the directions of these arrows for using relevance.)

Let us see how we modify the graph of the example using these steps.

We note by sign ! the edge from B because it is unique. We can use arrow-adding steps b) and c) among A, B and C (Fig. 2).

Now, we can use arrow-adding step c) for $\overline{CB} \in \mathbf{t}$, $BC \in \mathbf{l}$ or for $CA \in \mathbf{t}$, $\overline{AC} \in \mathbf{l}$.

Now we go back to the theory. We have a basic scheme:

Lemma 3.3. *If the graph of a puzzle $P(N,\mathbf{t},\mathbf{l})$ contains dashed loop-edge $AA \in \mathbf{l}$, then the node A is type* 0.

Proof. If there is a dashed loop edge at node A, then $A \supset \neg A$, and it means that A must be a liar. $\square$

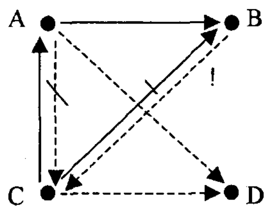

Figure 2: The graph of the Example 1.1 after some arrow-adding steps

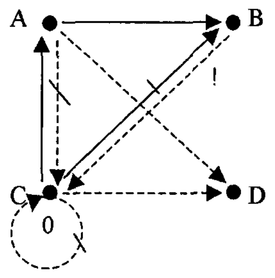

Figure 3: The graph after evaluating basic scheme at C

Let us see our example. We get basic scheme at C (Fig. 3). Using the scheme first we write the sign 0 to the node (it is a known type), and after this we examine the arrows, which remain relevant. It is very important part of this local method. In the next part we examine the valuable arrows, and the arrows, which become irrelevant. So if we know the type of the node of an end of an edge, what do we know about the other end?

Lemma 3.4. *Let $A, C \in N$ such a way that A is known type (let T and L be the known nodes with type 1 and type 0, respectively) as in the first column of the table and the noted edge between them is in P. The noted edge is a valuable arrow in the SW-puzzle if it is one of the following:*

	the case of valuable arrow	*after valuation we have these information*
a)	$A \in T$, $AC \in t$	*C is type* 1 *also*
b)	$A \in T$, (AC *or* $CA \in l$)	*C is type* 0
c)	$A \in L$, $\underline{AC} \in t$	*C is type* 0
d)	$A \in T$, $\underline{CA} \in t$	*C is type* 1
e)	$A \in L$, $CA \in t$	*C is type* 0
f)	$A \in L$, ($\underline{AC}$ *or* $\underline{CA} \in l$)♠	*C is type* 1

♠ *this arrow must be unique relevant at least from one direction to use this step*

Proof. It is evident from logical meaning of the arrows. □

At this point we detail the arrow-deletions and arrows changing to irrelevant steps. First of all, we note that the point d) and some special case of point c) in the previous lemma are this kind of steps also, as we will show in the next lemma.

Lemma 3.5.

a) Let A, B and C be three nodes. If $AB \in \boldsymbol{l}$ and $\underline{BC}$ or $\underline{CB} \in \boldsymbol{l}$ and the edge $AC \in \mathbf{t}$ is already in the graph such that both arrows from A (the dashed edge to B and the solid one to C) are relevant then we cross out one of them.

b) Our graph will be equivalent to the previous one in the following case also: let A, B and C be nodes such a way that $AB \in \mathbf{t}$, ($\underline{BC}$ or $\underline{CB} \in \boldsymbol{l}$) and $AC \in \boldsymbol{l}$ where both arrows from A are relevant. In this case we cross out one of them from A.

Proof. The connection between B and C means that B and C are different types. Hence if an arrow above is relevant in the solution then the other is relevant also. □

Now we will show the other cases when we cross out or delete an arrow.

Lemma 3.6. *Each arrow which starts from a type* 1 *node will be irrelevant.*

Proof. Trivially, it is from the definition of relevant edge. □

The following two lemmas are about that when we delete edges. We delete only edges, which have a known type end. If we cannot use an edge to get more information then we delete it.

Lemma 3.7. *Let T and L be the set of known type nodes (T is the nodes type 1, and L is the nodes type 0). Let A be a node, whose type is unknown at this time. If not only one relevant arrow started from the node A, but there is an arrow which goes to a known type node C, like*

a) $AC \in \mathbf{t}$, $C \in T$, and/or

b) $AC \in \boldsymbol{l}$, $C \in L$,

then this arrow will be deleted.

Proof. From the logical meaning of the arrows from A, we can use the $(*)$ formula. We have a conjunction in left hand side, these edges means values 1 in this conjunction. We can delete these values if it is not alone in this side. But there is other relevant edge from A, therefore we delete these arrows. □

Lemma 3.8. *If we know about the node A that it is type* 0 *($A \in L$), and there is a relevant arrow from A to a node C, like*

a) $AC \in \mathbf{t}$, $A, C \in L$, or

b) $AC \in \boldsymbol{l}$, $A \in L$ and $C \in T$

then we delete all other arrows starting from A.

Proof. We can use $(*)$. This formula must be valid independently the values of the other atomic sentences in the left hand side. □

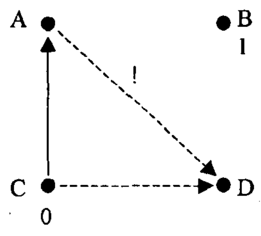

Figure 4: The graph after deleting edges

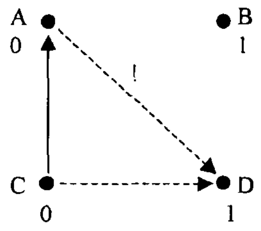

Figure 5: The solution of Example 1.1 on the graph

After using basic schemes we can use the point b) of lemma 3.7. We delete and cross the edges, which were relevant until this step.

Now we continue the solution of our example.

We can evaluate the value of B by step f) of lemma 3.4, hence B is type 1. We delete the edges AB, CB, BC, AC and CC (lemma 3.7). And we can sign the arrow AD by ! (Fig. 4).

We can use the step lemma 3.5. to cross out an edge from C. But after this we can ! sign the other arrow from C hence we can use valuable-arrow steps and get the value 0 for A and 1 for D. Therefore the result is: Alice and Charlie are liars, Bob and David are truth-tellers (Fig. 5).

In general case it is possible that we need the node-union step. We can use node union step in the following situation:

Lemma 3.9. *If there are two nodes A and B, such that $\underline{AB} \in t$ (the unique relevant arrow from A goes to B, and it is solid), then we unite these nodes. The united node has label "A, B" and we have all edges at this node which were into/from A and B but the $\underline{AB} \in t$ edge. And if there was relevant BA also, then after the node-union we have a relevant loop arrow at this node. The new graph is equivalent to the original one in the following sense. In the solution of the previous graph the nodes A and B have the same value as in the solution of the new graph the united node with label "A, B"; and all other nodes have the same value, respectively.*

Proof. From the logical meaning of the edge $\underline{AB} \in$ t we know, that $A \equiv B$. So all edge, which had endpoint A or B must be valid in the new graph. There were not more relevant edges from A, and we have all relevant edges from B. □

Sometimes we have information that two nodes are the same type but we cannot use node-union. Therefore we need the concept of parity of nodes, what we can use usually after the arrow-adding steps.

Definition 3.10. *Two nodes are in* parity, *if they are connected by solid arrows by both directions. We will use the notation $A \Leftrightarrow B$ to show that there are solid arrows between them in both way.*

(And we may use, that each node in parity with itself, because we can add solid loop arrow for each node, $A \supset A$ must be valid.)

Lemma 3.11. *The nodes in parity have same value.*

Proof. Assume, that A and B are in parity. Then from logical meanings of the solid arrows: $A \supset B$, $B \supset A$: $A \equiv B$. □

Lemma 3.12. *If from a node there are more same-type (solid or dashed) relevant edges going to the nodes which are in parity, then we can keep only one of them relevant, and we cross the others.*

Proof. The nodes in parity are same type, so all these relevant edges mean true atomic statements, or all of them mean false atomic statements. So it is equivalent to only one independent statement. Easy to show, that the new graph is equivalent to the previous one. □

Remark 3.13. *If from a node A there are more than 1 same-type edge going to the node B, then we leave only one of them. If there was relevant one among them, then we keep a relevant one, and delete the others.*

Remark 3.14. *If an edge is relevant in the solution, then it must be relevant also in the original graph of the puzzle.*

4 The global properties of the possible puzzle-graphs

Now, before we examine how we can use the global information of a puzzle-graph, we make some statements about the possible graphs.

Lemma 4.1. *There is no good and clear SW-type puzzle only with solid arrows.*

Proof. It has at least two different solutions: everybody is truth-teller; or each person is a liar. □

Lemma 4.2. *There is no good and clear SW-type puzzle, whose solution is that each person is a truth-teller.*

Proof. In the graph of this puzzle there are only solid arrows. So according to the previous lemma, our statement is true. □

Lemma 4.3. *If the graph of a clear and good puzzle has two or more components, then this puzzle falls apart: we have two or more less clear and good puzzle.*

Proof. In a clear puzzle in a component there is no information about the nodes in other components. □

According to the previous lemma, we assume that our graph has only one component, or we can solve the less one-component's puzzles.

The following lemma plays important rule when we use global information of a graph.

Lemma 4.4. *There is no dashed edge between two type* 1 *nodes in the solution.*

Proof. If a node is type 1, then all dashed arrows from it must go to type 0 nodes. □

Lemma 4.5. *We know from Lemma 4.1 that there is a dashed edge in the puzzle, but from the previous lemma we know that this edge cannot be between type* 1 *nodes. So there must be a type* 0 *node in an end of each dashed edge.*

Lemma 4.6. *Let T be the set of the truth-tellers in the solution. Then there is no solid arrow from this set which goes outside T.*

Proof. If a solid arrow starting from a type 1 node goes to a node A, then A must be type 1 also. □

Lemma 4.7. *If there is a directed circle built by solid arrows, then all nodes in this circle are in parity.*

Proof. Easy by using arrow adding steps b). □

Lemma 4.8. *Parity is an equivalence relation among nodes.*

Proof. Each node is in parity with itself, according to the note after the Definition 3.10. The symmetry come from the definition. It is transitive (if $A \Leftrightarrow B$, $B \Leftrightarrow C$ then $A \Leftrightarrow C$) because we can use step b) of Lemma 3.2. □

Lemma 4.9. *Let P be the graph of a good, clear SW-puzzle, and T be the set of type* 1 *nodes in the solution. If it is a node B, who is liar in the solution of P and he remained silent, then we can use arrow-adding steps for a new dashed edge from B to a truth-teller, or we can use node-union step to join B to an another liar.*

Proof. We assume that the graph is connected. If there is dashed edge from T to B, then we can use arrow-adding step a), and we get a new dashed edge from B to a truth-teller. If there is no edges between T and B originally, then must be an arrow from a liar to B. Let L is the set of nodes, which are not in T and differ from B. In this case originally there is not edge between T and B. (From B does not start any, and from T to B there is no solid arrow (Lemma 4.6.), and we assumed that there is nor dashed edge.) But the P was connected, so there must be edges from L to B. If there was a solid arrow from L to B, which is uniquely relevant from where it starts then we must use node-union step. And in final case there is no uniquely relevant arrow to B, which means that all non-silent liars have other arrows meaning his lie. But in this case it is also a solution, when $T \cup \{B\}$ is the sets of truth-tellers, and L is the set of liars. So in this case we get contradiction. □

Lemma 4.10. *Let P be the graph of a good, clear SW-puzzle, and T be the set of type 1 nodes in the solution. After all usual arrow-adding and node-union steps for P there is not possible only one node, which is not in T, and not connected with an element of T by dashed edge.*

Proof. We can assume, that P is connected, and we have no usual arrow-adding or node-union steps. From Lemma 4.2. we know, that there must be a node outside of T. Now we have two possibilities: B said something, or he remained silent. If he said something, and in the solution he is a liar, then must start a relevant arrow from B. If he said about a truth-teller C, that C is a liar, then this edge is dashed between B and T. If he said about a liar C that, C is truth-teller, then – because of all liar, but B are connected with T by dashed edge – the arrow-adding step c) (Lemma 3.2) is useful, and we get a dashed edge between B and T. In the case when B was silent, we can use the previous Lemma 4.9 for using node-union step. □

Lemma 4.11. *If in the good and clear puzzle's solution everybody is a liar, and the graph of the puzzle is connected, then after the possible arrow-adding and node-union steps we get a puzzle with only one node with two kind of loop edges.*

Proof. Easy to show, that for one node it is the unique puzzle. We will show that if we have more nodes then we can use node-union steps (and we get smaller and smaller puzzle with same solution).

If there was a node without starting relevant edges, then we can use node-union step by using Lemma 4.9. Now, we assume that we already used all possible arrow-adding steps. It is evident, that all relevant edges in the solution are solid, because each person is a liar. If from a node there is only one relevant arrow, then we must use node-union step. In other case from each node must start at least two relevant arrows. Let A be a node. Let T_A be the set of nodes, which we can reach from A by directed solid arrows. (By using arrow-adding steps it is evident, that we have a direct arrow from A to each element of T_A.) The set T_A is finite, let $Y_1, Y_2, \ldots Y_k$ the subsets of T_A, such that all nodes in a Y_i are in parity. Then we can use Lemma 3.12, so in each Y_i there at most only one relevant edge from each node is inside of Y_i. If there is a node, for which only one relevant edge remains, then we can use node-union. If such a node does not exist, then an other relevant arrow starting from all node in Y_i to outside of Y_i. So there is at least one set Y_j, which differs from Y_i, and there is relevant arrow from Y_i to Y_j. But there is the same situation with Y_j. So if we cannot use a node-union step inside of Y_j then a relevant edge must go to another Y_k. But we have only finite number of set Y_n. So we must have a circle by using directed solid arrows among the sets Y inside in T_A. But it means, that two or more sets are in parity. It is a contradiction.

So it is not possible that we cannot use node-union step, if we have at least two nodes. □

The following theorem is a summary of the previous lemmas. It shows the global information of the graph, what we can use in the next section.

Theorem 4.12. *Let P be the graph of a good, clear SW-puzzle, and T be the set of type 1 nodes in the solution. After all usual arrow-adding and node-union steps put L be the set of the nodes which are connected to T by a dashed edge. If P is connected then there is no node in P which is not in $T \cup L$.*

Proof. Let S be the set of the nodes, which are nor in T, neither in L. We will show that S will be empty set. From Lemma 4.10. we know, that it is impossible that S has only one element. Let us see how the set S connected to the other sets. According to Lemma 4.6. and the definition of set S from T there is no arrow to S. And there is no relevant edge from S to T (the solid arrows are not relevant, and there is no dashed edge between S and T). So from the nodes in S all relevant edges go to liars. If there is a relevant arrow from S to L, it must be solid, therefore we can use arrow adding step c), and we have a dashed edge between T and S, which contradicts the definition of S. So all relevant edges from S are in inside of S. But if there is a node A in set S from which there are not at least two relevant edges, then we can use node-union step (which is contradict to our assumption, that we already used these steps). So we are in the same situation as the proof of Lemma 4.11. As we state there, because these sets are finite, we have contradiction. So S must be the empty set. □

5 The general solving method

We know everything which we need to solve puzzles with the graph method.
Now we describe our method:

Algorithm 5.1.

0. Let B_i be the nodes of the graph. Draw the initial graph of the puzzle using only relevant edges.

Part I. (Graph-changing, by using local information) We try to use the following steps.

1. Use all possible node-union steps. (Lemma 3.9.)
2. Use all possible arrow-adding steps. (Lemma 3.2.)
3. Cross as many arrow possible. (Lemmas 3.5, 3.12, and 3.13)

If these steps cannot be repeated any more, then we continue by Part II.
Part II. (Choosing the set of truth-tellers, T, by using global information)

4. Make the matrix of the subgraph of dashed edges. (Use only the edges of l.)
5. Choose a maximal set of nodes T, which are not connected with dashed edges.

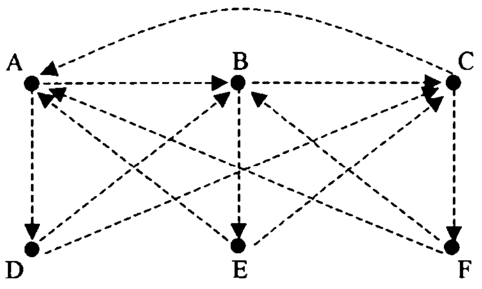

Figure 6: Graph of the Example 5.5

6. Check the following property: if there is a relevant arrow from each node which is not in T and is not silent originally, like dashed arrow to inside T, or solid arrow to outside T, then we have the solution.

If the property in step 6 is not true, then we choose another set T in step 5.
The solution is: all persons in T are truth-teller, and the others are liars.

Theorem 5.1. (Completeness and soundness of the algorithm) *Let P be the graph of a connected, good and clear SW-type puzzle. We can solve P by using Algorithm 5.1.*

Proof. It is clear, that the Part I. of the algorithm stops, because P is finite. It is evident, that we have only finite possibility to choose the set T. Let us assume, that we finished Part I. Let T' be the set of truth-tellers in the solution. T' is maximal because of Theorem 4.12, so we can choose T' as set T. We show that the property in point 6 must be true for this unique solution. Indirectly, assume that there is an – originally not silent – node B not in T for which we have nor solid relevant arrow to outside T neither dashed relevant edge to inside T. Then B did not lie originally, but he said something, hence he must be a truth-teller. But we have a dashed edge between the truth-tellers and B. It contradicts to Lemma 4.4. □

Remark 5.2. *In the case when each person is a liar our T set is empty. In this case according to Lemma 4.11 we have a puzzle with only one node after the steps of Part I.*

Remark 5.3. *Our algorithm detect if a puzzle has not any solution.*

And now we show an interesting example. In Lemma 4.1 we can see, that there is no good and clear SW-type puzzle only with solid arrows, now we show an example of a good and clear SW-type puzzle with only dashed edges. It is very nice symmetric example.

Example 5.4. A: B and D are liars. B: C and E are liars. C: A and F are liars. D: B and C are liars. E: A and C are liars. F: A and B are liars.

	A	*B*	*C*	*D*	*E*	*F*
A		x	x	x	x	x
B	x		x	x	x	x
C	x	x		x	x	x
D	x	x	x			
E	x	x	x			
F	x	x	x			

Table 1: The matrix of the graph

We can see that we have no local graph-step to use, so we cannot solve this puzzle without global information. There are two arrows starting from each nodes.

Now we solve this puzzle: As we can see, that we cannot use any steps of Part I of our algorithm. So we use Part II.

Let us make the matrix of the graph, which shows if two nodes are directly connected by a dashed edge. We use step 4.

The matrix of the graph in the table.

Our maximal T sets are the following: $\{A\}$, $\{B\}$, $\{C\}$, $\{D, E, F\}$. Easy to show in the original graph, that the condition of point 6 is not true for the first 3 sets. Our solution is D, E and F are truth-tellers, A, B and C are liars.

6 Summary

In this paper we defined and examined the SW-type of truth-tellers and liars puzzles. We represented these puzzles with graphs, which are very useful to examine and solve these puzzles. We examine what the edges of the graphs mean logically. The graph of a puzzle has all information about the puzzle in case of clear puzzle. We took some interesting statement about the possible structure of the puzzles. We used some local information steps in a graph as valuable arrows, arrow-adding, node-union steps and basic schemes. We showed that there is no clear and good SW-type puzzle with only solid arrows. Later on we presented a special example, when we have only dashed arrows. Finally we showed a graph-algorithm, which based on both local and global information of the graph, and it can solve the clear and good SW-type puzzles. The advantage of this method is to avoid case separations, which occurs for instance in tableaux method and requires great care for programmers. Using our method we need memory only size of n^2 for a puzzle with n persons to store our graph.

Using this approach from graph theory we can solve the puzzles in a new thinking way. Our theory connects the special type of satisfiability problems to graph theoretical problems.

References

[1] László Aszalós: *Smullyan's logical puzzles and their automatic solution* (in Hungarian: Smullyan logikai rejtvényei és automatikus megoldásuk), Technical Reports No. 2000/14, University of Debrecen, Institute of Mathematics and Informatics, 2000.

[2] László Aszalós: *The logic of Knights, Knaves, Normals and Mutes*, Acta Cybernetica, 14, 533-540, 2000.

[3] László Aszalós: *Examining the modal operator "can say" with mathematical logical techniques* (in Hungarian), PhD thesis, University of Debrecen, Institute of Mathematics and Informatics, 2001.

[4] Benedek Nagy, Márk Kósa: *Logical puzzles (Truth-tellers and liars)*, ICAI'01, 5th International Conference on Applied Informatics, Eger, 105-112, 2001.

[5] Benedek Nagy: *Truth-teller and liar puzzles and their graphs* (in Hungarian: Igazmondó-hazug fejtörők és gráfjaik), Technical Reports No. 2001/17, University of Debrecen, Institute of Mathematics and Informatics, 2001.

[6] Dennis Shasha, *The puzzling adventures of Dr Ecco* (Hungarian translation: Dr. Ecco talányos kalandjai), Typotex, Budapest, 1999.

[7] Raymond Smullyan, *Forever Undecided (A Puzzle Guide to Gödel)*, Alfred A. Knopf, New York, 1987.

[8] Raymond Smullyan, *The Lady or the Tiger? and other logical puzzles*, Alfred A. Knopf, New York, 1982, (Hungarian translation: A hölgy és a tigris) Typotex, Budapest, 1991.

[9] Raymond Smullyan, *What is the name of this book? (The riddle of Dracula and other logical puzzles)* , Prentince Hall, 1978, (Hungarian translation: Mi a címe ennek a könyvnek) Typotex, Budapest, 1996.

[10] Raymond Smullyan, *The riddle of Scheherazade, and other amazing puzzles, ancient & modern* (Hungarian translation: Seherezádé rejtélye) Typotex, Budapest, 1999.

Received October, 2002

Infinite limits and R-recursive functions

Jerzy Mycka*

Abstract

In this paper we use infinite limits to define R-recursive functions. We prove that the class of R-recursive functions is closed under this operation.

Keywords: Theory of computation, Real recursive function.

1 Introduction

The theory of recursion had been originally formulated for enumerable domains [3, 9]. Later the extensions on the continuous domains were proposed (for example see [4]). During a few past years many authors have studied problems of recursion theory for reals [1, 2].

The new approach was given by Moore in [5]. He used not only continuous functions, but also continuous operators on real recursive functions. The set of R-recursive functions defined by Moore is the subclass of real functions constructed as the smallest set containing $0, 1$ and closed under operations of composition, differential recursion and μ-recursion.

Infinite limits are the natural operation on real functions. and can be viewed as a method to define new functions. It is mentioned in [5] that limits can be expressed in terms of μ-operation, but without giving the way of this 'translation'. In this paper we give a proper way to define limits by μ-recursion.

This result can be useful for a few reasons. First, infinite limits are natural operations in calculus in contrast to the μ-operation. Furthermore with infinite limits we can define a limit hierarchy and relate it to the μ-hierarchy. This would be a continuous analog of Shoenfield's theorem [8]. Infinite limits can also be useful to compare the μ-hierarchy with the levels of Rubel's [7] Extended Analog Computer.

2 Preliminaries

This section summarizes some notions and results taken from [5], which are useful in this paper. Let us start with the precise definition of an R-recursive function.

*Institute of Mathematics, M. Curie-Sklodowska University, pl. M. Curie-Sklodowskiej 1,20-031 Lublin, Poland. E-mail: jmycka@golem.umcs.lublin.pl

Definition 2.1. *A function $h : R^m \to R^n$ is R-recursive if it can be generated from the constants 0 and 1 with the following operators:*

1. *composition: $h(\bar{x}) = f(g(\bar{x}))$;*
2. *differential recursion: $h(\bar{x}, 0) = f(\bar{x})$, $\partial_y h(\bar{x}, y) = g(\bar{x}, y, h(\bar{x}, y))$ (an equivalent formulation can be given by integrals: $h(\bar{x}, y) = f(\bar{x}) + \int_0^y g(\bar{x}, y', h(\bar{x}, y'))dy'$);*
3. *μ-recursion $h(\bar{x}) = \mu_y f(\bar{x}, y) = inf\{y : f(\bar{x}, y) = 0\}$, where infimum chooses the number y with the smallest absolute value and for two y with the same absolute value the negative one.*

Several comments are needed to the above definition. A solution of a differential equation need not to be unique or can diverge. Hence, we assume that if h is defined by differential recursion then h is defined only where a finite and unique solution exists. This is why the set of R-recursive functions includes also partial functions. For coherence with Moore's paper[5] we use the name of R-recursive functions in the article, however we should remember that in reality we have partiality here (partial R-recursive functions).

The second problem arises with the operation of infimum. Let us observe that if an infinite number of zeros accumulate just above some positive y or just below some negative y then the infimum operation returns that y even if y itself is not a zero.

The above defintion creates the class of R-recursive functions with some interesting features. Let us cite a few results from [5].

Lemma 2.2. *The functions $-x, x + y, xy, x/y, e^x, \ln x, x^y, \sin x, \cos x$ and the projection functions $I_n^i(x_1, \ldots, x_n) = x_i$ are R-recursive.*

The power of the system of R-recursive functions can be viewed from the following lemma, which is sufficient to solve the classical halting problem.

Lemma 2.3. *The function χ_S such that $\chi_S(x) = \begin{cases} 1 & x \in S, \\ 0 & x \notin S, \end{cases}$ is R-recursive for any partial N-recursive set S (S is partial N-recursive if $S = f_S(N)$ for $f_S : N \to N$, f_S is some N-recursive function).*

It is possible to define for every R-recursive function $f : R^n \times R \to R$ the characteristic function $\eta_y f$ for the set of $\bar{x}$ on which $\mu_y f$ is well-defined. Precisely this fact is stated by the below theorem:

Theorem 2.4. *If $f(\bar{x}, y)$ is R-recursive, then $\eta_y f(\bar{x}, y)$ is also R-recursive, where*

$$\eta_y f(\bar{x}, y) = \begin{cases} 1 & \exists y f(\bar{x}, y) = 0 \\ 0 & \forall y f(\bar{x}, y) \neq 0 \end{cases}$$

The operator μ is a key operator in generating the R-recursive functions. This fact suggests creating a μ-hierarchy, which is built with respect to the number of uses of μ in the definition of a given f.

Definition 2.5. *For a given R-recursive expression $s(\bar{x})$, let $M_{x_i}(s)$ (the μ-number with respect to x_i) be defined as follows:*

$$M_x(0) = M_x(1) = M_x(-1) = 0,$$

$$M_x(f(g_1, g_2, \ldots)) = \max_j (M_{x_j}(f) + M_x(g_j)),$$

$$M_x(h = f + \int_0^y g(\bar{x}, y', h)dy') = \max(M_x(f), M_x(g), M_h(g)),$$

$$M_y(h = f + \int_0^y g(\bar{x}, y', h)dy') = \max(M_{y'}(g), M_h(g)),$$

$$M_x(\mu_y f(\bar{x}, y)) = \max(M_x(f), M_y(f)) + 1,$$

where x can by any $x_1, \ldots, x_n$ for $\bar{x} = (x_1, \ldots, x_n)$.

For an R-recursive function f, let $M(f) = \max_i M_{x_i}(s)$ minimized over all expressions s that define f. Now we are ready to define μ-hierarchy.

Definition 2.6. *The μ-hierarchy is a family of $M_j = \{f : M(f) \leq j\}$.*

Let us add that if f is in M_j then $\eta_y f$ is in M_{j+2}.

As it was mentioned we focus our interest on functions defined by the infinite limits.

Definition 2.7. *Let $g : R^n \times R \to R$, then we can say the function $f : R^n \to R$ is defined by an infinite limit from g if:*

$$f(\bar{x}) = \begin{cases} \lim_{y\to\infty} g(\bar{x}, y) & \lim_{y\to\infty} g(\bar{x}, y) \textit{ exists,} \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

3 Auxiliary results

In this section we give a few results which will be useful in the proof of the main theorem. We start with a slight modification of the definition of R-recursive functions.

Lemma 3.1. *Let us consider the set of functions generated from $0, 1, -1$ by the operations 1,2 from the Definition 2.1 and by absolute μ-recursion $\mu_y^A f(\bar{x}, y) = inf\{|y| : f(\bar{x}, y) = 0\}$. Then this set of functions is equal to the set of all R-recursive functions.*

Proof. Because -1 can be simply defined in the set of R-recursive functions, it is sufficient to prove that in definitions operation μ can be replaced by μ^A and vice versa.

As the first case we consider the method of replacing μ^A by μ. Let $h(\bar{x}) = \mu_y^A f(\bar{x}, y)$. Let us set $f'(\bar{x}, y) = f(\bar{x}, y)f(\bar{x}, -y)$. Clearly $h(x) = -\mu_y f'(\bar{x}, y)$.

Because $-x$ and multiplication are R-recursive, so h defined as above is R-recursive too.

Now we must show that function $h(\bar{x}) = \mu_y f(\bar{x}, y)$ can be defined by μ^A. Let us point out the fact that $|x| = \mu_y^A(x - y)$. Then we will use $f^+(\bar{x}, y) = f(\bar{x}, |y|), f^-(\bar{x}, y) = f(\bar{x}, -|y|)$ to define

$$h^+(\bar{x}) = \mu_y^A f^+(\bar{x}, y), \quad h^-(\bar{x}) = \mu_y^A f^-(\bar{x}, y).$$

It is simple to observe that h^+ gives as the result the smallest nonnegative zero of f and h^- the absolute value of the greatest nonpositive (the smallest with respect to absolute value) zero of f.

We will choose as $h(\bar{x})$ the proper value from $h^+(\bar{x}), -h^-(\bar{x})$. We can define $K_=(w, y) = \delta(w - y)$, where $\delta(x) = 1 - \mu_y^A(x^2 + y^2)(y - 1)$. From definition $K_=(w, y)$ is equal to 1 if $w = y$, 0 if $w \neq y$. Then the function $\Theta(w) = 1$ for $x \geq 0$, 0 otherwise, can be defined as $K_=(w, |w|)$.

We can conclude the proof by the following observation

$$h(\bar{x}) = \begin{cases} -h^-(\bar{x}) & h^+(\bar{x}) \geq h^-(\bar{x}), \\ h^+(\bar{x}) & h^+(\bar{x}) < h^-(\bar{x}). \end{cases}$$

Hence $h(\bar{x}) = -h^-(\bar{x})\Theta(h^+(\bar{x}) - h^-(\bar{x})) + h^+(\bar{x})(1 - \Theta(h^+(\bar{x}) - h^-(\bar{x})))$ and since $+$ is defined without μ and all remaining functions in the last equation are defined only by μ^A, so $h(\bar{x})$ can be defined by μ^A instead of μ. □

It is interesting to define the μ^A-hierarchy of R-recursive functions as the analog to μ-hierarchy.

Definition 3.2. *For a given R-recursive expression $s(\bar{x})$, let $M_{x_i}^A(s)$ (the μ^A-number with respect to x_i) be defined as follows:*

$$M_x^A(0) = M_x^A(1) = M_x^A(-1) = 0,$$

$$M_x^A(f(g_1, g_2, \ldots)) = \max_j(M_{x_j}^A(f) + M_x^A(g_j)),$$

$$M_x^A(h = f + \int_0^y g(\bar{x}, y', h)dy') = \max(M_x^A(f), M_x^A(g), M_h^A(g)),$$

$$M_y^A(h = f + \int_0^y g(\bar{x}, y', h)dy') = \max(M_{y'}^A(g), M_h^A(g)),$$

$$M_x^A(\mu_y^A f(\bar{x}, y)) = \max(M_x^A(f), M_y^A(f)) + 1,$$

where x can by any $x_1, \ldots, x_n$ for $\bar{x} = (x_1, \ldots, x_n)$.

For an R-recursive function f, let $M^A(f) = \max_i M_{x_i}^A(s)$ minimized over all expressions s that define f and $M_j^A = \{f : M^A(f) \leq j\}$.

Now we can add a corollary from the previous lemma.

Corollary 3.3. *In the above lemma we use only one μ instead of μ^A, when we change the definition of function with μ^A by the definition with μ. Hence if some function f is from M_k^A then $f \in M_k$.*

The reverse relation is more complicated. In the definition of h given by $h(\bar{x}) = -h^-(\bar{x})\Theta(h^+(\bar{x}) - h^-(\bar{x})) + h^+(\bar{x})(1 - \Theta(h^+(\bar{x}) - h^-(\bar{x})))$ the operation μ^A is used 3 times. So each function from M_j belongs to M_{3j}^A.

Lemma 3.4. *Let $g : R^{n+1} \to R$ be an R-recursive function. Then there are R-recursive functions $G : R^{n+1} \to R$, $S : R \to R$ such that*

$$\inf_y g(\bar{x}, y) = S(\mu_w^A G(\bar{x}, w)).$$

Proof. We can distinguish three cases in the proof:

1. $(\forall \bar{x}, y) g(\bar{x}, y) \geq 0$. Then we can write

$$\inf_y g(\bar{x}, y) = \inf_w \{|w| : G'(\bar{x}, w) = 0\},$$

where

$$G'(\bar{x}, w) = \begin{cases} 0 & (\exists y) g(\bar{x}, y) - w = 0, \\ 1 & \text{otherwise.} \end{cases}$$

The condition $(\exists y) g(\bar{x}, y) - w = 0$ is equivalent to the fact that $\mu_y(g(\bar{x}, y) - w)$ is defined. But the last statement can easily be checked by the function η. Finally we have

$$G'(\bar{x}, y) = 1 - \eta_y(g(\bar{x}, y) - w) \text{ and } \inf_y g(\bar{x}, y) = \mu_w^A G'(\bar{x}, w),$$

so in this case $S = I_1^1, G = G'$.

2. $(\forall \bar{x}, y) g(\bar{x}, y) < 0$. In this case we can observe that $\inf_y g(\bar{x}, y)$ is a negative number and the infimum 'searches' in the direction of the smallest negative numbers, whereas μ^A gives us a positive result and its 'search' is oriented to zero. That is why we must use some transformation in a construction of the proper result. The simplest way to change 'the orientation' of the infimum is the expression $\frac{1}{g(\bar{x}, y)}$. Let our expression be equal to

$$G''(\bar{x}, w) = \begin{cases} 0 & (\exists y) \frac{1}{g(\bar{x}, y)} - w = 0, \\ 1 & \text{otherwise.} \end{cases}$$

then $G''(\bar{x}, w)$ is zero iff $\frac{1}{w}$ is a value of $g(\bar{x}, y)$ for some y. Hence an infimum on g is equal to:

$$\frac{-1}{\inf_w \{|w| : G''(\bar{x}, w) = 0\}}$$

and (like in the previous step) we eliminate the quantifier:

$$G''(\bar{x}, w) = 1 - \eta_y(\frac{1}{g(\bar{x}, y)} - w).$$

We can write now that

$$\inf_y g(\bar{x}, y) = S(\mu_w^A G(\bar{x}, w)), \quad S(z) = -1/z, G = G''.$$

3. Now we are prepared to consider the general case, where $g : R^{n+1} \to R$ is an arbitrary function. Let us observe, that if there is one point y_0^- such that $g(\bar{x}, y_0^-) < 0$ then $\inf_y g(\bar{x}, y)$ must be negative and if such point y_0^- does not exist then the first case of our proof solves the problem.
   To check an existance of the point y_0^- it is sufficient to use the condition $\eta_y K_<(g(\bar{x}, y), 0) = 1$ where $K_<(z, y) = 0 \iff z < y$. Then we can find y_0^-. We will use the following method (we must remember that μ_A gives us the absolute value of the proper solution)

$$y_0^- = \begin{cases} \mu_y^A K_<(g(\bar{x}, y), 0) & g(\bar{x}, \mu_y^A K_<(g(\bar{x}, y), 0)) \leq 0, \\ -\mu_y^A K_<(g(\bar{x}, y), 0) & \text{otherwise.} \end{cases}$$

   Then we can define the function:

$$g^-(\bar{x}, y) = \begin{cases} g(\bar{x}, y) & g(\bar{x}, y) < 0, \\ g(\bar{x}, y_0^-) & \text{otherwise.} \end{cases}$$

   This function for a given $\bar{x}$ has the same infimum as g, but its values are always negative.
   As a summary of the previous considerations we give the conditional definition of $q(\bar{x})$, where $q(\bar{x})$ denotes the expression $\inf_y g(\bar{x}, y)$:

$$q(\bar{x}) = \begin{cases} \inf_w\{|w| : 1 - \eta_y[g(\bar{x}, y) - w] = 0\} & \text{if } (\forall y) g(\bar{x}, y) > 0, \\ \frac{-1}{\inf_w\{|w| : 1 - \eta_y[\frac{1}{g(\bar{x}, y)} - \bar{w}] = 0\}} & \text{if } (\forall y) g(\bar{x}, y) < 0, \\ \frac{-1}{\inf_w\{|w| : 1 - \eta_y[\frac{1}{g^-(\bar{x}, y)} - w] = 0\}} & \text{otherwise.} \end{cases}$$

   Let us add that the condition $(\forall y) g(\bar{x}, y) > 0$ is equivalent to the statement y_0^- *does not exist*, but this last phrase can be expressed by $\eta_y K_<(g(\bar{x}, y), 0) = 0$. The similar translation of $(\forall y) g(\bar{x}, y) < 0$ is: $\eta_y K_<(0, g(\bar{x}, y)) = 0$.

It is obvious that such $q(x)$ is R-recursive ($\inf_w\{|w| : \ldots\}$ can be replaced by μ_A). The final forms of functions S and G can be obtained from the above definition of q. □

Remark 3.5. *Let us observe that in the general case (the third point of the above proof) we used for the construction of the definition of q the function η, infimum and y_0^-, which gives the number of used μ^A operations equal to 5. This is the maximal number of μ^A operations for all cases. Hence for $g \in M_j$ we have $q \in M_{j+5}$.*

We also need a similiar result, but with restricted infimum.

Lemma 3.6. *Let $g : R^{n+1} \to R$ be an R-recursive function. Then there are R-recursive functions $G : R^{n+2} \to R$, $S : R \to R$ such that for all $z \in R$*

$$\inf_{y\in(z,\infty)} g(\bar{x},y) = S(\mu^A_w G(\bar{x},w,z)).$$

Proof. Let us consider the set $S^{g,\bar{x}}_z$ such that

$$S^{g,\bar{x}}_z = \{w : (\exists y \geq z) g(\bar{x},y) = w\}.$$

We will use the characteristic function of this set: $\chi^{g,\bar{x}}_z(w) = 0 \iff w \in S^{g,\bar{x}}_z$ $\chi^{g,\bar{x}}_z(w) = 1 \iff w \notin S^{g,\bar{x}}_z$. From the Lemma 3.4 we have S_g, G_g for a given g and the problem of unrestricted infimum. It is clear that

$$\inf_{y\in(z,\infty)} g(\bar{x},y) = S_g(\mu^A_w(|G_g(\bar{x},w)| + |\chi^{g,\bar{x}}_z(w)|)).$$

Now we should prove only that $\chi^{g,\bar{x}}_z(w)$ is an R-recursive function. But $\chi^{g,\bar{x}}_z(w)$ can be written in the form

$$\chi^{g,\bar{x}}_z(w) = 0 \iff (\exists y)[(g(\bar{x},y) = w) \wedge (y \geq z)],$$

$$\chi^{g,\bar{x}}_z(w) = 1 \iff (\forall y)[(g(\bar{x},y) \neq w) \vee (y < z)].$$

These last equations define $\chi^{g,\bar{x}}_z(w)$ as $1 - \eta_y(|g(\bar{x},y)| + K_{\geq}(y,z))$, which ends this proof. □

Remark 3.7. *Because $\chi^{g,\bar{x}}_z(w)$ is defined by means of η and $K_{\geq}$, so it uses 3 μ^A operations but unrestricted infimum uses μ_A five times. Hence if $g \in M_j$ then the* $\inf_{y\in(z,\infty)} g(\bar{x},y)$ *belongs to M_{j+5}.*

4 Main theorem

In this section we prove that the class of R-recursive functions is closed under the operation of defininig functions by infinite limits.

Theorem 4.1. *Let $F : R^{n+1} \to R$ be an R-recursive function. Let us define $f : R^n \to R$ in the following way $f(\bar{x}) = \lim_{y\to\infty} F(\bar{x},y)$. Then there exist such R-recursive functions $G : R^{n+1} \to R$, $S : R \to R$ that*

$$f(\bar{x}) = S(\mu^A_w G(\bar{x},w)).$$

Proof. Let us consider the function $f(\bar{x})$ defined as above. The function $f(\bar{x})$ is defined in the point $\bar{x}$ if there exist limits

$$\liminf_{y\to\infty} F(\bar{x},y),\ \limsup_{y\to\infty} F(\bar{x},y)$$

and they are equal to each other in this point.

Let us recall the definitions:

$$\liminf_{y\to\infty} F(\bar{x},y) = \sup_z \inf_{y>z} F(\bar{x},y), \quad \limsup_{y\to\infty} F(\bar{x},y) = \inf_z \sup_{y>z} F(\bar{x},y).$$

To check that the function f is defined in $\bar{x}$ first we must check the conditions: $\sup_z \inf_{y>z} F(\bar{x},y)$, $\inf_z \sup_{y>z} F(\bar{x},y)$ are defined for $\bar{x}$. So it will be helpful if we prove that there exist functions K^i, K^s such that

$$K^i(\bar{x}) = \begin{cases} 1 & \sup_z \inf_{y>z} F(\bar{x},y) \text{ exists,} \\ 0 & \text{otherwise.} \end{cases}$$

and K^s is analogously defined for $\inf_z \sup_{y>z} F(\bar{x},y)$.

It is easy to see that we can replace the expression 'sup F' by the expression '$-\inf(-F)$' in the above equations. So we can apply Lemmas 3.4 and 3.6, which means, that there are R-recursive functions S^s, G^s, S^i, G^i such that

$$\sup_z \inf_{y>z} F(\bar{x},y) = S^i(\mu_w^A G^i(\bar{x},w)),$$

$$\inf_z \sup_{y>z} F(\bar{x},y) = S^s(\mu_w^A G^s(\bar{x},w)).$$

The left sides in the last two lines are defined iff there exist such w_i, w_s, that $G^i(\bar{x},w_i) = G^s(\bar{x},w_s) = 0$. This condition can be checked by the R-recursive functions $\eta_w G^i(\bar{x},w), \eta_w G^s(\bar{x},w)$. The above considerations imply that K^i, K^s exist and they are R-recursive.

Now to end the proof it is sufficient to define:

$$f(\bar{x}) =$$

$$= \begin{cases} S^i(\mu_w^A G^i(\bar{x},w)) & K^i(\bar{x})K^s(\bar{x})K_=(S^i(\mu_w^A G^i(\bar{x},w)), S^s(\mu_w^A G^s(\bar{x},w))) = 1 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

This definition of f by R-recursive functions is in the obvious way equivalent to the definition by the operation of infinite limit. □

Let us point out that in the above proof we use two operations of infimum for lim inf: the outer (unrestricted) infimum, which is obtained from the transformed supremum and the second - inner (restricted) infimum. We have the analogous construction for lim sup. Hence and from remarks below the Lemmas 3.4, 3.6 we can give the following result:

Theorem 4.2. *If F is a function from M_j then f defined as in the Theorem 4.1 is in M_{j+10}.*

References

[1] L. Blum, M. Shub, S. Smale, On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, *Bull. Amer. Math. Soc.* 21 (1989), 1-46.

[2] V. Brattka, Recursive characterization of computable real-valued functions and relations, *Th. Comp. Sc.* 162 (1996) 45-77.

[3] K. Gödel, Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I, *Monatsh. Math. Phys.* 38 (1931), 173-198.

[4] A. Grzegorczyk, On the definitions of computable real continuous functions, *Fund. Math.* 44 (1957), 61-71.

[5] C. Moore, Recursion theory on the reals and continuous-time computation, *Th. Comp. Sc.*, 162 (1996) 23-44

[6] P. Odifreddi, *Classical Recursion Theory*, North-Holland, Amsterdam, (1989).

[7] L. A. Rubel, The extended analog computer, *Adv. in Appl. Math.* 14 (1993) 39-50.

[8] J. R. Shoenfield, On degrees of unsolvability, *Ann. Math.* 69 (1959), 644-653.

[9] A. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc.* 42 (1936-37), 230-265.

Received May, 2002

Acta Cybernetica **16** (2003) 93–131.

Derivation of Incremental Equations for PNF Nested Relations

Jixue Liu* and Millist Vincent*

Abstract

Incremental view maintenance techniques are required for many new types of data models that are being increasingly used in industry. One of these models is the nested relational model that is used in the modelling complex objects in databases. In this paper we derive a group of expressions for incrementally evaluating query expressions in the nested relational model. We also present an algorithm to propagate base relation updates to a materialized view when the view is defined as a complex query.

Keywords: view maintenance, data warehousing, nested databases, partitioned normal form, incremental computation.

1 Introduction

Materialized views are stored data collections that are derived from source data. Materialized views have attracted a significant amount of attention in recent years because of their importance in data warehousing [5, 7, 20]. In using materialized views, an issue of fundamental significance is developing efficient methods for updating the materialized views in response to changes in the source data; a procedure referred to as *view maintenance*. To maintain a materialized view, one has in general a choice between recomputing the view from scratch or maintaining the views incrementally. The incremental method is generally considered to be less expensive [13, 4, 6] since the size of an update to the source data is generally small in relation to the size of the source data. To maintain a view incrementally, one computes the new view using the updates to the source data, the old view and possibly some source data. For example, let the view V be defined in the flat relational model (using set semantics) as $V = R_1 \bowtie R_2$. For an insertion δR_1 to R_1, the incremental technique calculates the change to V as $\delta V = \delta R_1 \bowtie R_2$ and computes the new view, V^{new}, by $V^{new} = V^{old} \cup \delta V$ (where V^{old} equals $R_1 \bowtie R_2$) [13, 6]. This expression is called an *incremental propagation expression* (or *incremental expression* (IE) for short) for the Join operator.

*School of Computer and Information Science, University of South Australia, Mawson Lakes, SA5095, Australia. Email: {j.liu, vincent}@cs.unisa.edu.au

Incremental expressions for updating materialized views depend on the data model and query operators. Up to now, incremental equations have been derived for the models of flat relations [13], bags [4], and temporal data models [21]. Incremental equations for the nested relational model, on the other hand, have not been studied. The nested relational model is important because of its usage in modelling complex objects, a feature that has been incorporated in several commercial database systems such as Oracle8 and Illustra [18]. The nested model has also been used in data warehouses to model complex semantics [3], where incremental view maintenance has critical impact on system performances [20]. Further, the nested relational model is an important subclass of the object- relational model; a model that has been predicted to become the industry standard within the next few years [18]. Motivated by these observations, in this paper we derive IEs and develop a view maintenance algorithm for the nested relational model.

Several variations of the nested relational model have been proposed in the literature, depending on whether null values are permitted [10], whether empty sets are permitted [2], whether atomic attributes form a key and what data manipulation operators are required [16, 15]. The model we use in this paper is the one proposed by [2] and called the *Verso model* which is based on *partitioned normal form (PNF) relations* [14]. The reason for adopting this model is because of its flexibility in supporting empty sets, the assumption that relations are in partitioned normal form (which has clearer semantics than general nested relations), and its ability to allow partial updates. Also, some commercial object-relational database systems such as Informix support the use of PNF relations.

The main contributions of this paper are as follows. Firstly, we derive incremental expressions for the data manipulation operators in the Verso model. Interestingly, these expressions differ significantly from those derived for the *flat* relational model [13]. Secondly, we propose an algorithm to propagate base relation updates to a materialized view when the view is defined as a complex nested relational algebra expression. Lastly, we implement our view maintenance algorithm and perform experiments to determine what we call the *maintenance limit* of our algorithm, which is defined to be the limit on the size of the update beyond which incremental maintenance is no longer cheaper than full view recomputation. This is an important issue and one that up to now has not been adequately investigated in the literature.

The rest of this paper is organized as follows. In Section 2, we introduce the Verso model and its operators. In section 3, we define containment and disjointedness properties for the PNF nested relations. These two properties will be used in Section 4 for deriving IEs. Section 4 contains IEs derived for PNF nested operators and the derivation proofs. In Section 5, we propose a view maintaining algorithm that maintains a view using IEs when the view is defined with multiple operators. Section 6 covers the implementation details of the IEs and performance analysis. In the last section of the paper, we give the conclusion.

2 Data Model and Operators

In this section, we review the Verso data model and algebra defined in [2].

2.1 Trees

A *tree* T is a finite, acyclic, directed graph in which there is a unique node, called the *root* and denoted by *root*(T), with in-degree (the number of edges coming into the node) 0 and every other node has in-degree 1.

A node n' is a child of a node n (or equivalently, n is the *parent* of n') if there is a directed edge from n to n'.

A node is a *leaf* if it has no children.

The *level* of a node n in a tree T is the number of nodes on a path from the root of T to n. Thus, the level of the root node is 1 and the root node is said on the *top level.*

The *height* of a tree is the maximum level of any node in the tree.

A tree T' is a *subtree* of a tree T if the nodes of T' are a subset of those of T and for every pair of nodes n' and n, n' is a child of n in T if and only if n' is a child of n in T'.

A subtree T' is a *child subtree* of T if the root node of T' is a child of T and the set of all nodes of T' and the set of all nodes of the child of T are equivalent.

2.2 Schema Trees and Nested Relation Schemas

Let U be a fixed countable finite set of atomic attribute names. Each attribute name $A \in U$ is associated with a countably infinite set of values denoted by *dom*(A).

A *schema tree* T is a tree having at least one node; *each* node of the tree is labeled by a set of names from U. The names on the labeled nodes form a partition of U.

A *nested relation schema* is the set of attribute names mapped from a schema tree T, denoted by *sch*(T), and defined recursively by:

(i) If T contains only one node (the root), then $sch(T) = \{A_1, ..., A_m\}$ where $A_1, .., A_m$ are attributes labeled on the root of T;

(ii) If $T_1, ..., T_n$ are child subtrees of T and $A_1, .., A_m$ are attributes labeled on the root of T, then $sch(T) = \{A_1, ..., A_m, sch(T_1), \ldots, sch(T_n)\}$.

In the schema definition, $A_1, ..., A_m$ are called the *atomic attributes* while $sch(T_1), ..., sch(T_n)$ are called the *structured attributes*. We denote each structured attribute $sch(T_i)$ $(i = 1, ..., k)$ by R_i^* and simplify $sch(T)$ by R. As a result, $sch(T) = R = \{A_1,\ ...,\ A_m,\ R_1^* : sch(T_1),\ ...,\ R_i^* : sch(T_i),\ ...,\ R_n^* : sch(T_n)\}$. Note that R_i^* is used only for referencing the schema of the child tree. If necessary, R_i^* can be labeled at the edge from $root(T)$ to $root(T_i)$.

Let $R' = sch(T')$ and $R = sch(T)$. R' is a *subschema* of R, denoted by $R' \Subset R$, if T' is a subtree of T. The *level of an attribute* in R is defined to be the level of the node in the tree where the attribute is labeled. When the level l of an attribute is specially concerned, l is attached to the attribute name as a superscript: A_i^l or R_j^{*l}. The *levels of the schema R* is defined to be the height of T. If a schema has l levels, the schema is called a *l-level nested* schema.

Because nested relation schemas are sets, set operations of union ($\cup$), difference ($-$), and intersection ($\cap$) can be applied to the top levels of schemas. Subset ($\subseteq$) can also be defined on the top levels of two schemas.

We define some short-hand notations for schemas. The set of atomic attributes on the top level of R is denoted by $\alpha(R)$ which is $\{A_1, ..., A_m\}$. The set of all structured attributes on the top level of R is denoted by $\beta(R)$ which is $R - \alpha(R)$. The function $\alpha_U(R)$ is defined to return all atomic attribute names labeled on all nodes of the schema tree of R.

The following is an example of a nested relation schema.

Example 2.1. We introduce a nested relation schema for a student database. A student with the name of *Name* has studied some subjects *Subjs**. The student has achieved a set of marks (denoted by *Marks**) for each subject; each mark is for a different test type of the subject. The student also has a set of telephone numbers stored in the database for the convenience of communication. The schema tree describing the student data is given in Figure 1. The schema of the schema tree is $Stud = \{Name,\ Subjs^*{:}\{sjName, Year,\ Marks^*{:}\{testName,\ Mark\}\},\ Tel^*{:}\{Tel\}\}$.

The schema is a three-level nested relation schema. On the first level, there is one atomic attribute *Name* and two structured attributes (structured attributes) *subjs** and *Tels**. That is, $\alpha(Stud) = \{Name\}$ and $\beta(Stud) = \{subjs^*, Tels^*\}$. The set of all atomic attributes of the schema is $\alpha_U(Stud) = \{Name, sjName, Year, Tel, testName, Mark\}$.

A subschema of *Stud* is $sjTest = \{sjName, tests^* : \{testName\}\}$ or $studTel = \{Name, Tels^* : \{Tel\}\}$.

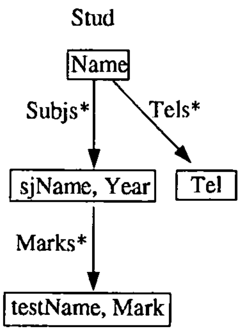

Figure 1: Schema tree *Stud*

Now we define the notion of prime subschema.

Definition 2.1 (Prime Subschema). *Let* $R = \alpha(R)\{R_1^*, ..., R_n^*\}$ *and* $S = \alpha(S)\{S_1^*, ..., S_{ns}^*\}$. *S is a* prime subschema *of R, denoted by* $S \Subset^p R$, *if*

(1) $\alpha(R) = \alpha(S)$ *and* $\beta(R) = \beta(R) = \phi$;

(2) $\alpha(R) = \alpha(S)$ *and for each* S_k^* $(k \in [1, ..., ns])$, *there exists a* R_j^* $(j \in [1, ..., nr])$ *such that* S_k^* *is the prime subschema of* R_j^*.

Note that if S is a prime subschema of R, then $\alpha(S) = \alpha(R)$ and the definition is recursive, which means that on each level of the two schemas, two corresponding structured attributes share the same atomic attribute set. The next example shows a prime subschema.

Example 2.2. Let *StudTel* $= \{Name, Tels^* : \{Tel\}\}$. Then *StudTel* is a prime subschema of *Stud* defined in Example 2.1 because the two schemas have the same set of atomic attributes $\{Name\}$ on the top level and because $Tels^*$ in *StudTel* is the same as $Tels^*$ in *Stud.*

2.3 Nested Relations

We now recursively define the domain of a schema R, denoted by $dom(R)$, by:

(i) If R is of one level, $dom(R) = dom(A_1) \times \ldots \times dom(A_m)$;

(ii) If R is of more than one level, then $dom(R) = dom(A_1) \times \ldots \times dom(A_m) \times P(dom(R_1^*)) \times \ldots \times P(dom(R_n^*))$ where $P(D)$ denotes the set of all nonempty, finite subsets of a set D.

A *nested relation* over a nested relation schema $R = \{A_1, ..., A_m, R_1^*, ..., R_n^*\}$, denoted by $r(R)$, or often simply by r when R is understood, is defined to be a finite set of elements from $dom(R)$. An element t in a relation is called a *tuple* and has the form of $t =< a_1, ..., a_m, r_1, ..., r_n >$ where $a_i \in dom(A_i)$ and r_j, called a *subrelation,* is a relation over the definition of structured attribute R_j^*. Each item, a_i or r_j, is called a *value* or a *component.* Two tuples are equivalent if their corresponding components are equivalent.

The *restriction* of tuple t to attributes A_i and to R_j^*, denoted by $t[A_i]$ and $t[R_j^*]$ respectively, is defined to be $t[A_i] = a_i$ and $t[R_j^*] = r_j$. If $Y = \{A_{1y}...A_{my}R_{1y}^*...R_{ny}^*\}$ is a subset of R, the **restriction of t to the subset Y**, denoted by $t[Y]$, is defined to be a tuple $< t[A_{1y}], ...t[A_{my}], t[R_{1y}^*], ...t[R_{ny}^*] >$. The **restriction** of relation r to Y, denoted by $r[Y]$, is defined to be the nested relation $\{t[Y] | t \in \mathbf{r}\}$.

We now give an example of a nested relation.

Example 2.3. Let *Stud* be the nested relation schema defined in Example 2.1. A nested relation r over the schema *Stud* is given in Table 1. There are three tuples in the relation: two tuples are for student *Jack* and one for *John.* Subrelations are labeled by pairs of curly brackets.

A nested relation is in *Partitioned Normal Form*(PNF) if all atomic attributes on the top level of the relation comprise the key and all subrelations are in partitioned normal form [14]. The nested relation in Table 1 is a PNF nested relation.

Table 1: A nested relation *stud* on schema *Stud*

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	*DB*	1998	*test1*	81	04143
			test2	90	1435
			exam	80	2302
	Java	1997	*ass1*	60	
			exam	80	
John	DB	1998	*test1*	60	2354
			test2	80	

2.4 Verso Operators

In this section we review the definitions of the Verso operators proposed in [2] and reviewed in [8].

Definition 2.2 (Expansion Operator). *Let S be a prime subschema of R. Let s be a relation defined over S. The* expansion *of s to schema R is a relation over R, denoted by $\eta_R(s)$, is defined recursively by:*

$$\begin{aligned}\eta_R(s) = \{x \mid \exists\, v \in s \ \wedge\ x[\alpha(R)] = v[\alpha(R)] \ \wedge\ \forall\, i \in [1,...,n] \\ (\textit{if}\ \exists\, S_j^* \ \wedge\ S_j^* \Subset R_i^* (x[R_i^*] = \eta_{R_i}(v[S_j^*])) \\ \textit{else}\ x[R_i^*] = \phi \qquad\qquad)\qquad \}\end{aligned}$$

The expansion operator recursively packs each tuple in s with empty sets to make it match the schema of R. The next example shows the use of the operator.

Example 2.4. Let $s = \{< \mathit{Tony}, \{51234, 51535\} >\}$ be a relation on schema $S = \{\mathit{Name}, \mathit{Tels}^* : \{\mathit{Tel}\}\}$. Let *Stud* be the schema described in Example 2.3. Then, $\eta_{Stud}(s) = \{< \mathit{Tony}, \phi, \{51234, 51535\} >\}$ where the empty set ϕ is the value packed for structured attribute Subjs^*.

Definition 2.3 (Projection Operator). *Let S be a prime subschema of schema R. Let r be a relation defined over R. The* projection *of r onto S is a relation over S, denoted by $\mathring{\pi}_S(r)$, defined recursively by:*

(i) $\mathring{\pi}_S(r) = \{x \mid x \in r\}$, *if R is flat;*

(ii) $\mathring{\pi}_S(r) = \{x \mid \forall\, u \in r\ (\ x[\alpha(R)] = u[\alpha(R)] \ \wedge\ \forall\, i \in [1,...,ns]$
$(x[S_i^*] = \mathring{\pi}_{S_i^*}(u[R_j^*])$ *where* $S_i^* \Subset^p R_j^*\ )\ \}$

The projection operator preserves key values of r on every level and recursively projects subrelations of r. Following is an example showing the use of the projection operator.

Example 2.5. Let *pstud* be the relation defined as in Table 1. Let *StudTel* = $\{\mathit{Name}, \mathit{Tel}^* : \{\mathit{Tel}\}\}$. The projection of *pstud* to *StudTel*, i.e. $\mathring{\pi}_{StudTel}(pstud)$ is given in Table 2.

Table 2: Projection of *pstud* to *StudTel*

Name	*Tels**
	Tel
Jack	$\left\{ \begin{array}{c} 04143 \\ 1435 \\ 2302 \end{array} \right\}$
John	{ 2354 }

We now define the selection condition for the Verso selection operator.

Definition 2.4 (Atomic condition). *An* atomic condition c_a *over a schema* $R = \{A_1, ..., A_m, R_1^*, ..., R_n^*\}$ *is defined by* $c_a = A_i \theta a_i$ *where* $A_i \in \{A_1, ..., A_m\}$, $a_i \in dom(A_i)$, *and* $\theta \in \{<, \leq, >, \geq, =, \neq\}$.

An atomic condition is set to an atomic attribute on the top level of a schema.

Definition 2.5 (Basic condition). *A* basic condition c_b *over a schema* $R = \{A_1, ..., A_m, R_1^*, ..., R_n^*\}$ *is defined by connecting a set of atomic conditions with* $\vee$ *(or),* $\wedge$ *(and),* $\neg$ *(not), and brackets.*

Definition 2.6 (Selection condition). *A* selection condition c *over a schema* $R = \{A_1, ..., A_m, R_1^*, ..., R_n^*\}$ *is defined recursively by*

(i) $c = (c_b)$ *if* R *is flat;*

(ii) $c = (c_b \ \wedge \ c_{r1} : R_1^*.c_1\theta' r_1 \ \wedge \ ... \ \wedge \ c_{rn} : R_n^*.c_n\theta' r_n)$.
In the condition c, c_{rj} $(j = 1, ..., n)$ *is a reference name to the expression* $R_j^*.c_j\theta' r_j$ *and ':' means 'defined by'. In* $R_j^*.c_j\theta' r_j$, $R_j^*.c_j$ *denotes the returned set selected from the subrelation over* R_j^* *by recursively applying selection condition* c_j. *The returned set then participates in the evaluation of* $\theta' r_j$ *where* r_j *is either the empty set* ϕ *or the any set* ω [1] *over* R_j^*. *When* r_j *is* ϕ, θ' *is one of* $\{=, \neq\}$ *while when* r_j *is* ω, θ' *is* $=$.
We call c_{rj} *the* existence condition *on subrelation of* R_j^*.

We now give an example of a selection condition.

Example 2.6. For schema $Stud = \{Name,\ Subjs^*{:}\{sjName, Year,\ Marks^*{:}\ \{testName,\ Mark\}\},\ Tel^*{:}$
$\{Tel\}\}$ defined in Example 2.1, a select condition over the schema is $c = (Name =' Jack' \wedge Subjs^* : (Marks^* : (Mark \geq 90) \neq \phi) \neq \phi\ )$. This selection condition selects a student named '*Jack*' who has obtained at least a good mark (≥ 90) for some subjects.

We use $c_b(x[\alpha(R)]) = true$ to denote the case where the key value of a tuple x makes an existence condition true. Accordingly, we use $c_{rj}(x[R_j^*]) = true$ to mean the case where a subrelation on R_j^* makes an existence condition is true.

[1] 'Any set' means that the number of elements in the set does not matter.

Definition 2.7 (Selection Operator). *Let $R = \alpha(R)\{R_1^*, ..., R_n^*\}$ be a schema and r be a relation over R. Let c be a selection condition defined with Definition 2.6. The* selection *of r based on c is a relation over R, denoted by $\mathring{\sigma}_c(r)$, recursively defined by:*

$$\begin{array}{rl} (i) & \mathring{\sigma}_c(r) = \{x | x \in r \text{ and } c_b(x[\alpha(R)]) = true\}, \text{ if } R \text{ is flat;} \\ (ii) & \mathring{\sigma}_c(r) = \{x | \exists\, u \in r \ \wedge \ c_b(x[\alpha(R)]) = true \ \wedge \ x[\alpha(R)] = u[\alpha(R)] \ \wedge \\ & \quad \forall j \in (1, ..., n)(c_{rj}(x[R_j^*] = \mathring{\sigma}_{c_j}(u[R_j^*])) = true \ ) \qquad \} \end{array}$$

Example 2.7. We apply the selection to relation *stud* in Table 1 with the selection condition c defined in Example 2.6. The returned relation from the selection is given in Table 3. We take the second tuple, denoted by t_2 in *stud* as an example to explain the operation. $t_2[Name]$ is 'Jack', which makes the basic condition $Name =' Jack'$ true. In the recursive part and on the inner-most level, the evaluation of $(Mark \geq 90)$ against $Marks^*$ is ϕ since the mark in tuple of $Marks^*$ is less than 90. Therefore $Marks^*.(Mark \geq 90) \neq \phi$ is evaluated 'False'. Since $t_2[Subj^*]$ has only one tuple and its subrelation is evaluated to 'False', so no tuple in $t_2[Subj^*]$ can be selected. This makes $Subjs^*.(Marks^*.(Mark \geq 90) \neq \phi) \neq \phi$ 'False'. As a result, the evaluation of the selection condition against this tuple is 'False' and not in Table 3.

Table 3: $\bar{\sigma}_c$ (*stud*)

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	{ *DB*	1998	{ *test2*	90 } }	{ 04143 1435 }

Definition 2.8 (Union Operator). *Let r and s be two relations over R. The* union *of r and s is a relation over R, denoted by $r \oplus s$, and recursively defined by:*

$$\begin{array}{rl} (i) & r \oplus s = \{x | x \in r \text{ or } x \in s\}, \text{ if } R \text{ is flat;} \\ (ii) & r \oplus s = \{x | \exists\, u \in r \wedge \ \exists\, v \in s \ \wedge \ x[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] \ \wedge \\ & \qquad \forall\, i \in [1, ..., n](\ x[R_i^*] = u[R_i^*] \oplus v[R_i^*] \ ) \text{ or} \\ & \quad \exists\, u \in r \ \wedge \ x[\alpha(R)] = u[\alpha(R)] \notin s[\alpha(R)] \ \wedge \ x = u \ ) \text{ or} \\ & \quad \exists\, v \in s \ \wedge \ x[\alpha(R)] = v[\alpha(R)] \notin r[\alpha(R)] \ \wedge \ x = v \ ) \} \end{array}$$

The union operator recursively combines two tuples, one from each operand relation, if their key values match on each level. The operation guarantees that the output of the union is in PNF, i.e., there are no duplicate values for atomic attributes on each level of the relation. The next example introduces the use of the union operator.

Example 2.8. Table 5 shows the union of relation *pstud* in Table 1 and relation *δpstud* in Table 4. *pstud* and *δpstud* each has a tuple with key value of *Jack*. As a result, the subrelations of the two *Jack* tuples are combined. This rule is applied recursively until two *ass*1 tuples on the most internal level of *java* 1997 merges into one tuple in the union and *ass*2 of *java* 1997 is added to the union. The same combination applies to *Tels**.

Tuple *John* in *pstud* and tuple *Andrew* in *δpstud* do not match any tuples in the other relation, they appear the same as they were before the union.

Table 4: *δpstud*

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	*Java*	1997	*ass*1	60	54111
			*ass*2	70	
Andrew	*DB*	1999	*test*1	60	ϕ

Table 5: Union of *pstud* and *δpstud*

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	*DB*	1998	*test*1	81	54111
			*test*2	90	04143
			exam	80	1435
	Java	1997	*ass*1	60	2302
			*ass*2	70	
			exam	80	
John	DB	1999	*test*1	60	2354
			*test*2	80	
Andrew	*DB*	1999	*test*1	60	ϕ

Definition 2.9 (Difference Operator). *Let r and s be two relations over R. The* difference *of r and s is a relation over R, denoted by $r \ominus s$, and recursively defined by:*

(i) $r \ominus s = \{x | x \in r \text{ and } x \notin s\}$, *if R is flat;*

(ii) $r \ominus s = \{x | \exists u \in r \wedge \exists v \in s \ \wedge \ x[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] \ \wedge$
$\forall i \in [1, ..., nt](\ x[R_i^*] = u[R_i^*] \ominus v[R_i^*] \neq \phi\ )$ *or*
$\exists u \in r \ \wedge \ x[\alpha(R)] = u[\alpha(R)] \notin s[\alpha(R)] \ \wedge \ x = u\ \}$

The difference operator is like the union operator in that it recursively differences subrelations if the key values of two tuples, one from each operand relation, match. The output of the difference operator is a relation in PNF. The following example shows the use of the difference operator.

Example 2.9. Table 6 gives the difference of *pstud* in Table 1 and the relation *δpstud* in Table 4. The difference is applied to the two tuples having the name of *Jack* in the two relations. This procedure recursively applies until it reaches the most inner level. As a result, in the most inner level the tuple *ass*1 of *Java* 1997 does not appear in the result. The tuple *John* in relation *pstud* does not match any tuples in the relation *δpstud* and it appears the same in output. In contrast, the tuple *Andrew* in relation *δpstud* does not affect any tuple in *pstud* because of no match of key values and is excluded in the result.

Table 6: $pstud \ominus \delta pstud$

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	{ *DB*	1998	{ *test*1	81	{ 04143
			*test*2	90	1435
			exam	80 }	2302 }
	Java	1997	{ *exam*	80 } }	
John	{ DB	1999	{ *test*1	60	{ 2354 }
			*test*2	80 } }	

Definition 2.10 (Intersection Operator). *Let r and s be two relations over R. The* intersection *of r and s is a relation over R, denoted by $r \odot s$, and recursively defined by:*

(i) $r \odot s = \{x | x \in r \text{ and } x \in s\}$, *if R is flat;*
(ii) $r \odot s = \{x | \exists u \in r \wedge \exists v \in s \ (\ x[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] \wedge \forall i \in (1, ..., n) \ (\ x[R_i^*] = u[R_i^*] \odot v[R_i^*] \ ) \ \}$

The use of the intersection operator is shown in the next example.

Example 2.10. Table 8 shows the intersection of $pstud_1$ in Table 7 and *pstud* in Table 1.

Table 7: A nested relation $pstud_1$

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	{ *DB*	1998	{ *exam*	80 } }	83304143

Table 8: $pstud \odot pstud_1$

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	{ *DB*	1998	{ *exam*	80 } }	ϕ

Definition 2.11 (Joinable schemas). *Let S and R be two schemas satisfying $\alpha(R) = \alpha(S)$. Then R and S are* joinable schemas *if there exists a schema T such that (1) $\alpha_U(T) = \alpha_U(R) \cup \alpha_U(S)$; (2) both R and S are prime subschemas of T. We call T the* ***joined schema.***

Example 2.11. Let *Stud* be a schema defined in Example 2.3. Let $S = \{Name, Addrs^* : \{Addr\}\}$ be another schema which describes the student addresses. *Stud* and S are joinable because $\alpha(Stud) = \alpha(S)$ and there exists a schema $T = \{Name, Subjs^* : \{Subj, Year, Marks^* : \{TestName, Mark\}\}, Tels^* : \{Tel\}, Addrs^* : \{Addr\}\}$ such that (1) $\alpha_U(T) = \alpha_U(Stud) \cup \alpha_U(S)$; (2) *Stud* and S are prime subschemas of T. So T is the joined schema.

Definition 2.12 (Join). *Let S and R be joinable schemas and T be the joined schema. Let r and s be relations on R and S respectively. The* join *of r and s is a relation over T, denoted by $r \overset{\circ}{\bowtie} s$, defined recursively by:*

$$\begin{array}{ll} (i) & r \overset{\circ}{\bowtie} s = \{x | x \in r \ \wedge \ x \in s\}, \textit{ if } R = S = T \textit{ are flat;} \\ (ii) & r \overset{\circ}{\bowtie} s = \{x | \exists\, u \in r \ \wedge \ \exists\, v \in s \ \wedge \ x[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] \ \wedge \ \forall\, i \in [1, ..., nt] \\ & \qquad (\textit{if } \exists\, R_j^* \subseteq^p T_i^* \ \wedge \ \exists\, S_k^* \subseteq^p T_i^* \ (\, x[T_i^*] = u[R_j^*] \overset{\circ}{\bowtie} v[S_k^*] \,) \textit{ or} \\ & \qquad \textit{if } \exists\, R_j^* \subseteq^p T_i^* \ \wedge \ \nexists\, S_k^* \subseteq^p T_i^* \ (\, x[T_i^*] = u[R_j^*]) \textit{ or} \\ & \qquad \textit{if } \exists\, S_k^* \subseteq^p T_i^* \ \wedge \ \nexists\, R_j^* \subseteq^p T_i^* \ (\, x[T_i^*] = v[S_k^*]) \qquad) \} \end{array}$$

The join operator joins two relations based on the equivalence of the values of the atomic attributes starting from the top level. The next example shows the use of the join operator.

Example 2.12. Let *pstud* be defined in Table 1. Let *studAddr* be a relation defined in Table 9. The results of join of *pstud* and *studAddr* is shown in Table 10.

Table 9: *studAddr*

Name	*Addrs**
	Addr
John	{12 *Newton st*, 5 *Darling av* }

The Verso operators presented in this section have the property of preserving key attributes on all levels. In other words, all operators do not shrink or expand keys of relations. For example, the projection operation only projects structured attributes but not atomic attributes. This property guarantees the results of operations are in partitioned normal form.

Table 10: The join of *pstud* and *stAddr*

Name	*Subjs**				*Tels**	*Addrs**
	sjName	*Year*	*Marks**		*Tel*	*Addr*
			testName	*Mark*		
John	{ DB	1999	{ *test1*	60	{ 2354 }	{ 12 *Newton st*
			test2 } }	80		5 *Darling av* }

3 Containment and Disjointedness in Nested Relations

In this section, we review the definitions and results from [11] concerning the properties of containment and disjointedness in PNF relations. These results will be used in deriving IEs in the next section. At the same time, we compare the properties of containment and disjointedness for nested relations with the corresponding properties in flat relations.

In flat relations [13, 4], disjointedness means that when an insertion is made to a relation, the tuples to be inserted should not be included in the relation; whereas containment means that when tuples are deleted from a relation, the deleted tuples should be contained in the relation. It is also desirable in many applications, such as those involving triggers or real-time databases, that the changes to the view computed using IEs also satisfy the containment and disjointedness properties.

The issue of how to extend the definitions of containment and disjointedness from flat relations to PNF relations is not as straightforward as might first appear. This is discussed in more detail in [11] but we briefly summarise our approach here for the sake of completeness. In [11] we adopted the approach of [10, 15]. In this approach we require that the definitions for containment for and disjointedness must be *faithful* and *precise*. By faithful, we mean that the definitions for containment and disjointedness for PNF relations should coincide with the definitions for containment and disjointedness for flat relations when the PNF relations are in fact flat. By preciseness we mean that the properties should coincide with the corresponding properties for flat relations when applied to the total unnests of the PNF relations.

For containment, we proposed the following definition in [11] and showed it to be faithful and precise.

Definition 3.1 (Containment). *Let r and δr be two instances over schema R. Then δr is defined to be contained in r, denoted by $\delta r \,\textcircled{$\subseteq$}\, r$, if:*

(i) when R is flat, $\forall v \in \delta r \ \wedge\ v \in r$;

(ii) when R is not flat, $\forall v \in \delta r \ \wedge\ \exists u \in r \ \wedge\ v[\alpha(R)] = u[\alpha(R)] \ \wedge\ \forall i \in [1, ..., nr](v[R_i^*] \,\textcircled{$\subseteq$}\, u[R_i^*])$.

For example in Table 11, $\delta r \,\textcircled{$\subseteq$}\, r$. However, we note that in this table that δr is not a subset of r.

Table 11: Relations showing the containment

A	B^*
	B
a_1	$\{b_1\}$

δr

A	B^*
	B
a_1	$\{b_1, b_2\}$

r

Also, in [11] we show that nested containment has the following properties. These properties will be used in the next section.

Theorem 3.1. *Let r and δr be two instances over schema R. Then the following are equivalent:*

(i) $\delta r \subseteq_n r$;

(ii) $r \odot \delta r = \delta r$;

(iii) $r \oplus \delta r = r$.

As for disjointedness, the following definition was proposed in [11] and shown to be faithful and precise.

Definition 3.2 (Disjointedness). *Let $\delta r \neq \phi$ and r be two relations over schema R. δr is defined to be disjoint from r, denoted by $\delta r \mid\!\cap_n r$, if*

(i) r *is* ϕ;

(ii) when R is flat, $\forall\, v \in \delta r \ \wedge \ v \notin r$;

(iii) when R is not flat, $\forall\, v \in \delta r$,

(a) $v[\alpha(R)] \notin r[\alpha(R)]$ *or*

(b) $\exists\, u \in r$, $v[\alpha(R)] = u[\alpha(R)]$ *and*
$\exists\, i(v[R_i^*] \neq \phi \ \wedge \ v[R_i^*] \mid\!\cap_n u[R_i^*])$.

For example, the two relations shown in Table 12 are disjoint.

Table 12: Two cases of disjointedness

A	B^*	C^*
	B	C
a	$\{b_1\}$	$\{c_1\}$

δr

A	B^*	C^*
	B	C
a	$\{b_2\}$	$\{c_2\}$

r

We now introduce another type of disjointedness which, when it holds, we will show in the next section to considerably simplify incremental equations.

Definition 3.3. *Let r_1 and r_2 be two nested relations defined over schema R and let $A \subseteq R$. Then a tuple $x \in r_1$ is A-*disjoint *from r_2 if $x[A]$ is not in $r_2[A]$ (otherwise x is said A-*overlapping *with r_2). The two relations r_1 and r_2 are defined to be A-*disjoint *if every $x \in r_1$ is A-disjoint from r_2 (note that the definition is symmetric).*

We now illustrate the definition by Example 3.1.

Example 3.1. There are three relations r_0, r_1, and r_2 defined over a schema $R = \{A, B, C^* : \{C\}, D^* : \{D\}\}$ in Table 13. Let $Y = \{B, C^*\}$. Then r_0 and r_1 are $R - Y$ disjoint, but r_0 and r_2 are not. This is because $R - Y = \{A, D^*\}$ while $r_0[\{A, D^*\}] \cap r_1[\{A, D^*\}] = \phi$ and $r_0[\{A, D^*\}] \cap r_2[\{A, D^*\}] = \{< a_1, \{d_1, d_2\} > \} \neq \phi$.

The first tuple in r_0 is a $R - Y$ overlapping tuple with r_2 while the second tuple in r_0 is a $R - Y$ disjoint tuple with r_2.

Table 13: An example for $R - Y$ disjointedness

A	B	C^*	D^*
		C	D
a_1	b_1	$\{c_1, c_2\}$	$\{d_1, d_2\}$
a_2	b_1	$\{c_1, c_2\}$	$\{d_1, d_2\}$

r_0

A	B	C^*	D^*
		C	D
a_1	b_2	$\{c_1, c_2\}$	$\{d_1\}$

r_1

A	B	C^*	D^*
		C	D
a_1	b_2	$\{c_1, c_2\}$	$\{d_1, d_2\}$

r_2

4 Incremental Equations for Nested Operators

In this section, we derive incremental expressions for the nested operators defined in Section 2. We assume that the update to a relation is a full tuple update, i.e., the updating tuples and the relation have the same schema. Otherwise, if the schema of the update is a prime subschema of the updated relation, we assume that the expansion operator has been applied to expand the updating tuples into full tuples.

We firstly give a general overview of what we are aiming to derive in this section of the paper. We are aiming to derive equations of the form $op_u(r \bigoplus \delta r) = f(op_u(r), r, \delta r)$ in the case of a unary query operator op_u, and $op_b(r \bigoplus \delta r, s) = f(op_b(r, s), r, s, \delta r)$ in the case of a binary operator op_b. In this notation $\bigoplus$ means either the PNF union operator $\oplus$ or the PNF difference operator $\ominus$; r and s are called ***base relations***; δr is called the ***update*** to the base relation and f is a function. We call $op_u(r)$ and $op_b(r, s)$ the ***old views***, $op_u(r \bigoplus \delta r)$ and $op_b(r \bigoplus \delta r, s)$ the ***recomputation***, $f(op_u(r), r, \delta r)$ and $f(op_b(r \bigoplus s), r, s, \delta r)$ the ***incremental computation***. For each equation, we use the abbreviation of LHS for left hand side and RHS for right hand side.

It is particularly desirable if the RHS of the IE for an operator take the simple form of $op_u(r) \bigoplus op_u(\delta r)$ $(op_b(r, s) \bigoplus op_b(\delta r, s))$. We call this form of IE the ***standard form***. The advantage of this form is that is does not involve extra operators. When the size of the increment is small, in general it is much more efficient to compute the new view incrementally than by recomputation. Standard

IEs may not exist for some operators, but we can in some cases still derive IEs in the ***limited standard form*** which means a standard form attached with some conditions. The advantage of the limited standard form of an IE is that it reveals the reason why the IE can not be standard. However, the test of conditions in the limited standard form can be costly because recursive traversal down to subrelations is needed and there is no index possible for the internal subrelations. To avoid testing the expensive conditions, we define the ***implementation form*** for IEs. In the implementation form, the concept of the attribute disjointedness defined in the last section is used and testing the expensive conditions is replaced by top level selection.

We note that we use induction to prove the IEs in the section because all the operators involved in IEs of the section are recursive. In the proofs of induction, we will firstly prove that an equation is correct for a flat relation and then prove the equation is correct for a n-level nested relation if it holds for $(n-1)$-level nested subrelations.

4.1 Incremental Equations for the Expansion Operator

Theorem 4.1. *Let S be a prime subschema of a schema R and let r and δr be two instances over S. Then the following two expressions for the expansion operator are true.*

$$\eta_R(r \oplus \delta r) = \eta_R(r) \oplus \eta_R(\delta r) \tag{1}$$
$$\eta_R(r \ominus \delta r) = \eta_R(r) \ominus \eta_R(\delta r) \tag{2}$$

Proof.
Proof of Equation 1:

(1) Base Case: when $R = S$ are flat, the equation holds. The proof is obvious. In this case, by the definition of expansion, on LHS: $\eta_R(r) = r$, $\eta_R(\delta r) = \delta r$, $\eta_R(r) \oplus \eta_R(\delta r) = r \oplus \delta r$. on RHS: $\eta_R(r \oplus \delta r) = r \oplus \delta r$. Base case is proved.

(2) Induction: suppose $\eta_{R_i^*}(u[S_i^*] \oplus v[S_i^*]) = \eta_{R_i^*}(u[S_i^*]) \oplus \eta_{R_i^*}(v[S_i^*])$ where $u \in r$ and $v \in \delta r$. We prove the equation is correct over r and δr.

   (a) $\eta_R(r \oplus \delta r) \subseteq \eta_R(r) \oplus \eta_R(\delta r)$
   For a tuple $x \in \eta_R(r \oplus \delta r)$, by the definition of union, x is expanded from a tuple u of r, a tuple v of δr, or a tuple unioned from u and v.

      (i) x is expanded from u (i.e. $u[\alpha(R)] \notin \delta r[\alpha(R)]$):
      $x = u[\alpha(R)](\eta_{R_1^*} u[S_1^*])...(\eta_{R_m^*} u[S_m^*]) \underbrace{\phi \ldots \phi}_{m+1\ldots nr}$.
      On RHS : since $u \in r$, the expansion of u, which is the same as x, is contained in $\eta_R(r)$. Because expansion does not change key values of tuples, $u[\alpha(R)] \notin \delta r[\alpha(R)] \Longrightarrow u[\alpha(R)] \notin \eta_R(s)[\alpha(R)]$. Further, the union in RHS does not change values of the tuple expanded from u. Hence, $x \in (\eta_R(r) \oplus \eta_R(\delta r))$.

      (ii) x is expanded from v: this case is symmetric to the last case.

(iii) x is expanded from the union of u and v ($u[\alpha(R)] = v[\alpha(R)]$):
The union of u and v is $u[\alpha(R)](u[S_1^*] \oplus v[S_1^*])...(u[S_m^*] \oplus v[S_m^*])$.
Then x is expanded from this union,
$x = u[\alpha(R)](\eta_{R_1^*}(u[S_1^*] \oplus v[S_1^*]))...(\eta_{R_m^*}(u[S_m^*] \oplus v[S_m^*])) \underbrace{\phi \,...\, \phi}_{m+1...nr}$. By the induction assumption that the equation holds on level $(n-1)$, we have union and expansion are exchangeable on second level. Then
$x = u[\alpha(R)](\eta_{R_1^*}u[S_1^*] \oplus \eta_{R_1^*}v[S_1^*])...(\eta_{R_m^*}u[S_m^*] \oplus \eta_{R_m^*}v[S_m^*]) \underbrace{\phi \,...\, \phi}_{m+1...nr}$.
By rewriting,
$x = u[\alpha(R)](\eta_{R_1^*}u[S_1^*] \oplus \eta_{R_1^*}v[S_1^*])...(\eta_{R_m^*}u[S_m^*] \oplus \eta_{R_m}v[S_m^*]) \underbrace{(\phi \oplus \phi)...(\phi \oplus \phi)}_{from\ m+1\ to\ nr}$
By definition of union and expansion, x is the union of the expansion of u and the expansion of v. So the tuple x is contained in RHS.

Item (a) is proved.

(b) $\eta_R(r) \oplus \eta_R(\delta r) \subseteq \eta_R(r \oplus \delta r)$ The proof is similar to Item (a) and omitted.

The equation is proved.

Intuitively, the expansion operator packs the structured attributes in R but not in S with ϕ. It changes neither values for the structured attributes nor the key values of r and δr. As a result, expansion does not affect the union property of r and δr, and the equation is correct. In other words, the expansion operator is faithful [10] with respect to the union operator.

Proof of Equation 2:

(1) Base case: when $R = S$ are flat, the equation holds. In this case, bythe definition of expansion, on LHS, $\eta_R(r) = r$, $\eta_R(\delta r) = \delta r$, $\eta_R(r) \ominus \eta_R(\delta r) = r \ominus \delta r$. On RHS, $\eta_R(r \ominus \delta r) = r \ominus \delta r$. Base case is proved.

(2) Induction: suppose $\eta_{R_i^*}(u[S_i^*] \ominus v[S_i^*]) = \eta_{R_i^*}(u[S_i^*]) \ominus \eta_{R_i^*}(v[S_i^*])$ where $u \in r$ and $v \in \delta r$. We prove the equation is correct over r and δr.

(a) $\eta_R(u[S] \ominus v[S]) \subseteq \eta_R(u[S]) \ominus \eta_R(v[S])$
For a tuple $x \in (\eta_R(u[S] \ominus v[S]))$, there must exit a tuple $x' \in (r \ominus \delta r)$ such that x is expanded from x'. By the definition of difference, x' must be produced from r and δr in two disjoint cases.

(i) x' is from r: $\exists\, u \in r$, $u[\alpha(R)] \notin \delta r[\alpha(R)]$, $x' = u$.
In this case, x is expanded from u as $x = u[\alpha(R)]\eta_{R_1}(u[S_1^*])...\eta_{R_m}(u[S_m^*]) \underbrace{\phi \,...\, \phi}_{m+1...nr}$. On RHS : $u \in r \Longrightarrow$ the expansion of u, which is the same as x, is contained in $\eta_{R_i^*}(u[S_i^*])$. Because expansion does not change key value of tuples, $u[\alpha(R)] \notin \delta r[\alpha(R)] \Longrightarrow u[\alpha(R)] \notin \eta_R(\delta r)[\alpha(R)]$. By the definition

of difference, the expansion of u is not changed by difference. So $x \in RHS$.

(ii) x' is the difference of tuples from r and δr: $\exists u \in r, \exists v \in \delta r$, $u[\alpha(R)] = v[\alpha(R)]$
$x' = u[\alpha(R)](u[S_1^*] \ominus v[S_1^*])...(u[S_m^*] \ominus v[S_m^*])$ and $\exists i \in [1,...,m], (u[S_i^*] \ominus v[S_i^*]) \neq \phi$
x is expanded from x':
$x = u[\alpha(R)](\eta_{R_1}(u[S_1^*] \ominus v[S_1^*]))...(\eta_{R_m}(u[S_m^*] \ominus v[S_m^*])) \underbrace{\phi \ ... \ \phi}_{m+1...n}$, and $\exists i \in [1,...,m]$, $(u[S_i^*] \ominus v[S_i^*]) \neq \phi$.
Note that expansion adds empty sets to the structured attributes. It does not change the values of existing attributes. Therefore, $(u[S_i^*] \ominus v[S_i^*]) \neq \phi \Longrightarrow (\eta_{R_1} u[S_i^*] \ominus \eta_{R_1} v[S_i^*]) \neq \phi$. By rewriting x, we have
$x = u[\alpha(R)](\eta_{R_1} u[S_1^*] \ominus \eta_{R_1} v[S_1^*])...(\eta_{R_m} u[S_m^*] \ominus \eta_{R_m} v[S_m^*]) \underbrace{(\phi \ominus \phi)...(\phi \ominus \phi)}_{from\ m+1\ to\ n}$ and $\exists i \in [1,...,m]$, $(\eta_{R_1} u[S_i^*] \ominus \eta_{R_1} v[S_i^*]) \neq \phi$.
This just equals to the difference of the expansion of u and the expansion of v in RHS . So we proved that $x \in RHS$.

Item (a) is proved.

(b) $\eta_{R_i}(u[S_i]) \ominus \eta_{R_i}(v[S_i]) \subseteq \eta_{R_i}(u[S_i] \ominus v[S_i])$
This item is proved in a similar way as in Item (a).

Equation 2 is proved. □

4.2 Incremental Equations for the Projection Operator

Lemma 4.1. *Let S be a prime subschema of R. Let r and δr be instances over R. Then, the following two equations hold.*

$$\mathring{\pi}_S(r \oplus \delta r) = \mathring{\pi}_S(r) \oplus \mathring{\pi}_S(\delta r) \tag{3}$$
$$\mathring{\pi}_S(r \ominus \delta r) = \mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r) \oplus \mathring{\pi}_S(r \ominus \delta r) \tag{4}$$

Proof.
Proof of Equation 3:

(1) Base case: when $R = S$ are flat, the projection does nothing to tuples in r and s. So the equation holds.

(2) Induction: suppose that for $u \in r$ and $v \in s$, $\mathring{\pi}_{S_j}(u[R_j^*] \oplus v[R_j^*]) = (\mathring{\pi}_{S_j}(u[R_j^*]) \oplus \mathring{\pi}_{S_j}(v[R_j^*]))$, we prove $\mathring{\pi}_S(r \oplus s) = (\mathring{\pi}_S(r) \oplus \mathring{\pi}_S(s))$.

(a) $\mathring{\pi}_S(r \oplus s) \subseteq (\mathring{\pi}_S(r) \oplus \mathring{\pi}_S(s))$
For a tuple $x \in \mathring{\pi}_S(r \oplus s)$, there must exist a tuple $x' \in (r \oplus \delta r)$ such that x is the projection of x'. x' is generated by the union in 3 cases:

(i) $\exists\, u \in r \ \wedge\ u[\alpha(R)] \notin \delta r[\alpha(R)]$ and x' is produced by u:
In this case, x is the expansion of u. On RHS, $u \in r \Longrightarrow x \in\in \mathring{\pi}_S(r)$; the projection does not change the key value of a tuple. Therefore no tuple in $\mathring{\pi}_S(\delta r)$ will affect $x \in \mathring{\pi}_S(r)$ when the union is conducted. $x \in (\mathring{\pi}_S(r) \oplus \mathring{\pi}_S(s))$.

(ii) $\exists\, v \in \delta r \ \wedge\ v[\alpha(R)] \notin r[\alpha(R)]$ and x' is produced by v: Symmetric to case (i).

(iii) $\exists\, u \in r \ \wedge\ \exists\, u \in \delta r \ \wedge\ u[\alpha(R)] = v[\alpha(R)]$ and x' is the union u and v: $x' = u[\alpha(R)](u[R_1^*] \oplus v[R_1^*])...(u[R_{nr}^*] \oplus v[R_{nr}^*])$,
x is the projection of x':
$x = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*] \ominus v[R_1^*])...(\mathring{\pi}_{S_{ns}}(u[R_{ns}^*] \ominus v[R_{ns}^*])$
By the induction assumption,
$x \ = \ u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*]) \ominus \mathring{\pi}_{S_1^*}(v[R_1^*]))...(\mathring{\pi}_{S_{ns}}(u[R_{ns}^*]) \ominus \mathring{\pi}_{S_{ns}}(v[R_{ns}^*]))$

On RHS: let the projection of u be denoted by x^u and the projection of v be denoted by x^v. Then $x^u \in \mathring{\pi}_S(r)$ and $x^v \in \mathring{\pi}_S(\delta r)$:
$x^u = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*]))...(\mathring{\pi}_{S_{ns}^*}(u[(R_{ns}^*)]))$
$x^v = v[\alpha(R)](\mathring{\pi}_{S_1^*}(v[R_1^*]))...(\mathring{\pi}_{S_{ns}^*}(v[(R_{ns}^*)]))$
The union of x^u and x^v produces: $u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*]) \ominus \mathring{\pi}_{S_1^*}(v[R_1^*]))...(\mathring{\pi}_{S_{ns}}(u[R_{ns}^*]) \ominus \mathring{\pi}_{S_{ns}}(v[R_{ns}^*])) \Longrightarrow x$
Consequently, $x \in (\mathring{\pi}_S(r) \oplus \mathring{\pi}_S(s))$.

Item (a) is proved.

(b) $(\mathring{\pi}_S(r) \oplus \mathring{\pi}_S(s)) \subseteq \mathring{\pi}_S(r \oplus s)$: This proof is similar to Item (a) is omitted.

The equation is proved.

Proof of Equation 4: We only need to prove that $\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r) \ \underline{\Subset}\ \mathring{\pi}_S(r \ominus \delta r)$ because of Theorem 3.1.

(1) Base case: when $R = S$ are flat, the projection does nothing to tuples in r and s. $\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r) = \mathring{\pi}_S(r \ominus \delta r)$.

(2) Induction: suppose that for $u \in r$ and $v \in s$, $(\mathring{\pi}_{S_j}(u[R_j^*]) \ominus \mathring{\pi}_{S_j}(v[R_j^*])) \underline{\Subset}\ \mathring{\pi}_{S_j}(u[R_j^*] \ominus v[R_j^*])$, we prove $(\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(s)) \underline{\Subset}\ \mathring{\pi}_S(r \ominus s)$.
For a tuple $x \in (\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(s))$, x is produced in two cases:

(i) x is the difference of x^u and x_v where $x^u \in \mathring{\pi}_S(r)$ and $x^v \in \mathring{\pi}_S(\delta r)$:
Suppose x^u is the projection of $u \in r$ and x^v is the projection of $v \in r$.
By the definition of projection, we have
$x^u = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*]))...(\mathring{\pi}_{S_{ns}^*}(u[(R_{ns}^*)]))$
$x^v = v[\alpha(R)](\mathring{\pi}_{S_1^*}(v[R_1^*]))...(\mathring{\pi}_{S_{ns}^*}(v[(R_{ns}^*)]))$
By the definition of difference,
$x = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*]) \ominus \mathring{\pi}_{S_1^*}(v[R_1^*]))...(\mathring{\pi}_{S_{ns}}(u[R_{ns}^*]) \ominus \mathring{\pi}_{S_{ns}}(v[R_{ns}^*]))$
and $\exists\, i \in [1, ..., ns](\mathring{\pi}_{S_i}(u[R_i^*]) \ominus \mathring{\pi}_{S_i}(v[R_i^*]) \neq \phi)$.
From $(\mathring{\pi}_{S_i}(u[R_i^*]) \ominus \mathring{\pi}_{S_i}(v[R_i^*]) \neq \phi)$ we have $u[R_i^*] \ominus v[R_i^*] \neq \phi$ because projection makes a tuple have less attributes.

On RHS: the difference of u and v produce a tuple $y \in (r \ominus \delta r)$:
$y = u[\alpha(R)](u[R_1^*] \ominus v[R_1^*])...(u[R_{nr}^*] \ominus v[R_{nr}^*])$ since $\exists i \in [1, ..., ns](u[R_i^*] \ominus v[R_i^*] \neq \phi)$.
The projection of y produces y' in RHS:
$y' = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*] \ominus v[R_1^*])...(\mathring{\pi}_{S_{ns}}(u[R_{ns}^*] \ominus v[R_{ns}^*])$ and $\exists i \in [1, ..., ns](u[R_i^*] \ominus v[R_i^*] \neq \phi)$.
By induction assumption, all subrelations of x are contained in the according subrelations of y'. Therefore, x is tuple-contained in y'.

(ii) x is in $\mathring{\pi}_S(r)$ and $x[\alpha(R)] \notin \delta r[alpha(R)]$: This prove is the similar to Item (i) in the proof of Equation 3 and omitted.

The containment is proved.

$\square$

Equation 4 reveals that $\mathring{\pi}_S(r \ominus s)$ may not be contained in $(\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(s))$. The following lemma gives the reason.

Lemma 4.2.
$\mathring{\pi}_S(r \ominus \delta r) = \mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r)$ *iff recursively* $\exists\, u \in r \wedge \exists\, v \in \delta r \wedge u[\alpha(A)] = v[\alpha(A)] \wedge \exists\, j \in [1...m](u[(R_j)] \ominus v[(R_j)] \neq \phi \Longrightarrow (\mathring{\pi}_{S_j^*} u[R_j^*] \ominus \mathring{\pi}_{S_j^*} v[R_j^*]) \neq \phi)$.

The proof of the Lemma is the reverse of the proof of Equation 4. Generally, $(u[R_i^*] \ominus v[R_i^*] \neq \phi) \not\Longrightarrow (\mathring{\pi}_{S_i}(u[R_i^*]) \ominus \mathring{\pi}_{S_i}(v[R_i^*]) \neq \phi)$ because the projection makes a tuple shorter. The shortened parts might be the difference of $u[R_i^*]$ and $v[R_i^*]$. Once this difference is shortened, $\mathring{\pi}_{S_i}(u[R_i^*])$ and $\mathring{\pi}_{S_i}(v[R_i^*])$ become the same. So, $(\mathring{\pi}_{S_i}(u[R_i^*]) \ominus \mathring{\pi}_{S_i}(v[R_i^*]) \neq \phi)$ may not be true. When the condition in the lemma is true, the equation becomes true.

The next example shows the importance of the condition in the lemma.

Example 4.1. Let $R = \{A, B^* : \{B\}, C^* : \{C\}\}$ and $S = \{A, B^* : \{B\}\}$. Let r and δr be two instances over R shown in Table 14. We see that $\mathring{\pi}_S(r \ominus \delta r) \neq \mathring{\pi}_S(r) \ominus \mathring{\pi}_T(\delta r)$. This is caused by the first tuple of r and in the first tuple of δr. The order of difference and projection on the two tuples affect the result.

When the two tuples are differenced first, the result is $< a_1, \phi, \{c_1\} >$, The projection of the tuple is $< a_1, \phi >$ which is in the recomputation of $\mathring{\pi}_S(r \ominus \delta r)$.

However, when the two tuples are projected, we obtain $< a_1, \{b_1\} >$ and $< a_1, \{b_1\} >$ respectively. The difference of these two tuples results none in the result.

Based on the two lemmas given above, we propose the following implementation form of IEs for the projection operator.

Theorem 4.2. *Let S be a prime subschema of R. Let r and δr be instances over R. Then, the following two equations hold.*

$$\mathring{\pi}_S(r \oplus \delta r) = \mathring{\pi}_S(r) \oplus \mathring{\pi}_S(\delta r)$$
$$\mathring{\pi}_S(r \ominus \delta r) = \mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(\mathring{\pi}_S(r)) \oplus \mathring{\pi}_S(\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r) \ominus \delta r) \quad (5)$$

Table 14: Relations for Example 4.1

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{c_1\}$
a_2	$\{b_1, b_2\}$	$\{c_1, c_2\}$
a_3	$\{b_2, b_3\}$	$\{c_1, c_3\}$

r

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{c_2\}$
a_2	$\{b_1, b_2\}$	$\{c_1, c_2\}$
a_3	$\{b_3\}$	$\{c_3\}$

δr

A	B^*
	B
a_1	$\{\phi\}$
a_3	$\{b_2\}$

$\mathring{\pi}_S(r \ominus \delta r)$

A	B^*
	B
a_3	$\{b_2\}$

$\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r)$

The first equation has been proved in Lemma 4.1. We now prove the second equation. From Lemma 4.2, the none equivalence of $\mathring{\pi}_S(r \ominus \delta r)$ and $\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r)$ is caused by $\alpha(R)$-overlapping tuples in r and δr. Based on this observation, in Equation 5, we delete from the old view $\mathring{\pi}_S(r \ominus \delta r)$ the tuples that are produced by key value overlapping tuples in r. We then recompute the $\alpha(R)$-overlapping tuples in r and δr by $\mathring{\pi}_S(\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r) \ominus \delta r)$. At last, the recomputed tuples are inserted to the view.

4.3 Incremental Equations for the Selection Operator

Incremental equations for the selection operator can not be in the standard form because of the following lemma.

Lemma 4.3. *Let r and δr be two relations over schema R. Let c be a selection condition defined with Definition 2.6. Then $\mathring{\sigma}_c(r)$ is not always contained in $\mathring{\sigma}_c(r \oplus \delta r)$ and $\mathring{\sigma}_c(r \ominus \delta r)$.*

This lemma is supported by the next example.

Example 4.2. Let r and δr be two relations given in Table 15 and $c = \{B^* \neq \phi, C^* = \phi\}$ be a selection condition. The selections of r, $r \oplus \delta r$, and $r \ominus \delta r$ are also given Table 15. Obviously, $\mathring{\sigma}_c(r)$ is not contained in $\mathring{\sigma}_c(r \oplus \delta r)$ and $\mathring{\sigma}_c(r \ominus \delta r)$.

Since it is not possible to have the standard form of IEs for the selection operator, we derive IEs in the limited standard form for the selection operator if we impose restrictions on r and δr.

Lemma 4.4. *Let r and δr be two relations over schema R. Let c be a selection condition defined with Definition 2.6. Then the following equations are true.*

(i) $\mathring{\sigma}_c(r \ominus \delta r) = \mathring{\sigma}_c(r) \ominus \mathring{\sigma}_c(\delta r)$ *iff* $\exists\,(u \in r \ \wedge\ v \in \delta r \ \wedge\ u[\alpha(A)] = v[\alpha(A)])$, *then recursively* $\forall\, i \in [1, ..., n]$ $(\ c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*]) \ominus \mathring{\sigma}_{c_i}(v[R_i^*])) = true \ \wedge\ c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true \ \wedge\ c_{ri}(\mathring{\sigma}_{c_i}(v[R_i^*])) = true \ \wedge\ \exists i(u[R_i^*] \ominus v[R_i^*] \neq \phi \ \wedge\ \mathring{\sigma}_{c_i}(u[R_i^*]) \ominus \mathring{\sigma}_{c_i}(v[R_i^*]) \neq \phi)$

(ii) $\mathring{\sigma}_c(r \oplus \delta r) = \mathring{\sigma}_c(r) \oplus \mathring{\sigma}_c(\delta r)$ *iff* $\exists\,(u \in r \ \wedge\ v \in \delta r \ \wedge\ u[\alpha(A)] = v[\alpha(A)])$, *then recursively* $\forall\, i \in [1, ..., n]$ $(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true \ \wedge\ c_{ri}(\mathring{\sigma}_{c_i}(v[R_i^*])) = true)$

We now choose to prove the first equation. The proof of the second equation is similar to that of the first one.

Table 15: Relations for Example 4.2

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	ϕ
a_5	$\{b_5\}$	ϕ

r

A	B^*	C^*
	B	C
a_1	$\{b_2\}$	$\{c_2\}$
a_5	$\{b_5, b_6\}$	ϕ

δr

A	B^*	C^*
	B	C
a_1	$\{b_1, b_2\}$	$\{c_2\}$
a_5	$\{b_5, b_6\}$	ϕ

$r \oplus \delta r$

A	B^*	C^*
	B	C
a_5	$\{b_1\}$	ϕ

$r \ominus \delta r$

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	ϕ
a_5	$\{b_5\}$	ϕ

$\mathring{\sigma}_c(r)$

A	B^*	C^*
	B	C
a_5	$\{b_5, b_6\}$	$\{\phi\}$

$\mathring{\sigma}_c(r \oplus \delta r)$

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{\phi\}$

$\mathring{\sigma}_c(r \ominus \delta r)$

Proof.

Proof of Equation (i) in Lemma 4.4:

(1) Base case: when $\beta(R) = \phi$, r and δr are flat. The equation becomes $\mathring{\sigma}_{c_b}(r - \delta r) = \mathring{\sigma}_{c_b}(r) - \mathring{\sigma}_{c_b}(\delta r)$. The correctness of this equation is proved in [13].

(2) Induction: when r is not flat and $\mathring{\sigma}_{c_i}(u[R_i^*] \ominus v[R_i^*]) = \mathring{\sigma}_{c_i}(u[R_i^*]) \ominus \mathring{\sigma}_{c_i}(v[R_i^*])$ where $u \in r$ and $v \in \delta r$, we prove the equation is correct below.

(a) $\mathring{\sigma}_c(r \ominus \delta r) \subseteq \mathring{\sigma}_c(r) \ominus \mathring{\sigma}_c(\delta r)$
For $x \in \mathring{\sigma}_c(r \ominus \delta r)$, there exist a tuple x' in $(r \ominus \delta r)$ such that $c(x')$ is true. x' is computed by tuple $u \in r$ and tuple $v \in \delta r$ in one of two ways.

(1) $u[\alpha(R)] \notin \delta r[\alpha(R)]$ and $x' = u$. Thus,
$x = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*])...(\mathring{\sigma}_{c_n}(u[R_n^*])$ where $c_b(u[\alpha(R)]) = true$ and $\forall\, i, c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true.$
In RHS, the selection of u produces tuple y $(= x)$ in $\mathring{\sigma}_c(r)$:
$y = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*,])...(\mathring{\sigma}_{c_n}(u[R_n^*])$ where $c_b(u[\alpha(R)]) = true$ and $\forall\, i(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true)$.
Since selection does not change the key value of a tuple, we have $u[\alpha(R)] \notin \delta r[\alpha(R)] \Longrightarrow y[\alpha(R)] \notin \mathring{\sigma}_c(\delta r)[\alpha(R)]$. By definition of difference, no tuple in $\mathring{\sigma}_c(\delta r)$ affects y when difference is conducted in RHS. After difference, y is still the same as x. So, x is in RHS.

(2) $u[\alpha(R)] = v[\alpha(R)]$ and x' is the difference of u and v. That is,
$x' = u[\alpha(R)](u[R_1^*] \ominus v[R_1^*])...(u[R_n^*] \ominus v[R_n^*])$ where $\exists\, i(u[R_i^*] \ominus v[R_i^*] \neq \phi)$. In this case, the selection of x' gives
$x = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*] \ominus v[R_1^*]))...(\mathring{\sigma}_{c_n}(u[R_n^*] \ominus v[R_n^*]))$
where $c_b(u[\alpha(R)]) = true$ and $\forall\, i(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*] \ominus v[R_i^*])) = true)$ and $\exists\, i(u[R_i^*] \ominus v[R_i^*] \neq \phi)$.
By induction assumption,
$x = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*,]) \ominus \mathring{\sigma}_{c_1}(v[R_1^*]))...(\mathring{\sigma}_{c_n}(u[R_n^*]) \ominus \mathring{\sigma}_{c_n}(v[R_n^*]))$

where $c_b(u[\alpha(R)]) = true$ and $\forall\, i(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*] \ominus v[R_i^*])) = true)$ and $\exists\, i(u[R_i^*] \ominus v[R_i^*] \neq \phi)$.
In RHS , the selection of u and v results tuple $y^u \in \mathring{\sigma}_c(r)$ and tuple $y^v \in \mathring{\sigma}_c(s)$:
$y^u = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*]))...(\mathring{\sigma}_{c_n}(u[R_n^*]))$ where $c_b(u[\alpha(R)]) = true$ and $\forall\, i(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true)$.
$y^v = v[\alpha(R)](\mathring{\sigma}_{c_1}(v[R_1^*]))...(\mathring{\sigma}_{c_n}(v[R_n^*]))$ where $c_b(v[\alpha(R)]) = true$ and $\forall\, i(c_{ri}(\mathring{\sigma}_{c_i}(v[R_i^*])) = true)$.
Let y be the difference of y^u and y^v (because of $u[\alpha(R)] = v[\alpha(R)]$):
$y = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*]) \ominus \mathring{\sigma}_{c_1}(v[R_1^*,]))...(\mathring{\sigma}_{c_n}(u[R_n^*]) \ominus \mathring{\sigma}_{c_n}(v[R_n^*]))$
where $c_b(u[\alpha(R)]) = true$, $c_b(v[\alpha(R)]) = true$, $\forall\, i(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true$ and $c_{ri}(\mathring{\sigma}_{c_i}(v[R_i^*])) = true)$, and $\exists\, i(\mathring{\sigma}_{c_i}(u[R_i^*]) \ominus \mathring{\sigma}_{c_i}(v[R_i^*]) \neq \phi)$.
The conditions attached to x and to y are the conditions attached tot he equation. When these conditions are true, x equals to y. Hence x is in RHS.

(b) $\mathring{\sigma}_c(r) \ominus \mathring{\sigma}_c(s) \subseteq \mathring{\sigma}_c(r \ominus s)$
This proof is the reverse of the proof of Case (a).

The equation is proved.

□

We use the next example to show the importance of the conditions attached to the equations in Lemma 4.4.

Example 4.3. Let $R = \{A, B^* : \{B\}, C^* : \{C\}\}$ and let r and δr be two instances over R and given in Table 16. Let the selection condition be $c = (A =' a_1', C = \phi)$. LHS=$\mathring{\sigma}_c(r \oplus \delta r) = \phi$; while RHS= $\mathring{\sigma}_c(r) \oplus \mathring{\sigma}_c(\delta r) \neq \phi$. The reason is that the condition of subrelations over C^* being empty is violated by r and δr: the subrelation $\{c_1\}$ in r is not ϕ.

Since the equations in Lemma 4.4 can not be applied to this example. The new view has to be recomputed.

Table 16: Relations for Example 4.3

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{c_1\}$

r

A	B^*	C^*
	B	C
a_1	$\{b_2\}$	ϕ

δr

A	B^*	C^*
	B	C
	ϕ	

$\mathring{\sigma}_c(r \oplus \delta r)$

A	B^*	C^*
	B	C
a_1	$\{b_2\}$	ϕ

$\mathring{\sigma}_c(r) \oplus \mathring{\sigma}_c(\delta r)$

The above lemma indicates that the reason for not being able to derive a standard IE for the selection operator is that r and δr having $\alpha(R)$-overlapping tuples.

The conditions attached to the equations in Lemma 4.4 are recursive. In other words, the conditions have to be tested against subrelations on all levels. This can be very time consuming because of traversing down to deep levels and because there is no index possible for subrelations in a relation. Furthermore, after the test, the conditions may be violated, in this case, we have to recompute to view. Those tuples that have been traversed for testing the conditions will be traversed again for the recomputation. Double traversal causes the test plus recomputation to be more expensive than just recomputation without the pre-test. To overcome the disadvantage of the performance, in the next theorem we avoid the test of the conditions by proposing IEs in the implementation form.

Theorem 4.3.

$$\mathring{\sigma}_c(r \ominus \delta r) = \mathring{\sigma}_{\alpha(R)\notin\delta r[\alpha(R)]}(\mathring{\sigma}_c(r)) \oplus \mathring{\sigma}_c(\mathring{\sigma}_{\alpha(R)\in\delta r[\alpha(R)]}(r) \ominus \delta r) \tag{6}$$
$$\mathring{\sigma}_c(r \oplus \delta r) = \mathring{\sigma}_{\alpha(R)\notin\delta r[\alpha(R)]}(\mathring{\sigma}_c(r)) \oplus \mathring{\sigma}_c(\mathring{\sigma}_{\alpha(R)\in\delta r[\alpha(R)]}(r) \oplus \delta r) \tag{7}$$

In the theorem, $\alpha(R)$-overlapping tuples are recomputed while other tuples are computed incrementally. The IEs in the theorem can applied without limitation.

4.4 Incremental Equations for the Intersection Operator

The next Lemma indicates that the IE for the intersection operator with union is in the standard form. However, the IE for the intersection operator with difference can not be in the standard form since $(r \odot s) \ominus (\delta r \odot s)$ is contained in $(r \ominus \delta r) \odot s$, but not the other way around.

Lemma 4.5. *Let R be a schema and r, δr, and s be instances over R. Then*

$$(r \ominus \delta r) \odot s = (r \odot s) \ominus (\delta r \odot s) \oplus (r \ominus \delta r) \odot s \tag{8}$$
$$(r \oplus \delta r) \odot s = (r \odot s) \oplus (\delta r \odot s) \tag{9}$$

We choose to prove Equation 8. The proof of the other equation can be achieved in a similar way.

Proof.
Proof of Equation 8: To prove the equation, we only need to prove $(r \odot s) \ominus (\delta r \odot s) \overset{n}{\subseteq} (r \ominus \delta r) \odot s$ because of Theorem 3.1.

(1) Base case: when R is flat, the equation is true since in flat case, $(r \ominus \delta r) \odot s = (r \odot s) \ominus (\delta r \odot s)$ is proved in [13].

(2) Induction: we suppose that for $u \in r \ \wedge \ v \in \delta r \ \wedge \ w \in s$, $(u[R_i^*] \odot w[R_i^*]) \ominus (v[R_i^*] \odot w[R_i^*]) \overset{n}{\subseteq} (u[R_i^*] \ominus v[R_i^*]) \odot w[R_i^*]$. We prove $(r \odot s) \ominus (\delta r \odot s) \overset{n}{\subseteq} (r \ominus \delta r) \odot s$.
For $x \in ((r \odot s) \ominus (\delta r \odot s))$, there exist tuple $x^u \in (r \odot s)$ and tuple $x^v \in (\delta r \odot s)$ such that x is the difference of x^u and x^v. By the definition of the difference, x is computed in the following ways from x^u and x^v.

   (a) $x^u[\alpha(R)] = x^v[\alpha(R)]$ then x is the difference of x^u and x^v.
   Because x^u is the intersection of $u \in r$ and $w \in s$ while x^v is the intersection of $v \in \delta r$ and $w \in s$, i.e.,

$x^u = u[\alpha(R)](u[R_1^*] \odot w[R_1^*])...(u[R_{nr}^*] \odot w[R_{nr}^*])$ and
$x^v = v[\alpha(R)](v[R_1^*] \odot w[R_1^*])...(v[R_{nr}^*] \odot w[R_{nr}^*])$.
So x is the difference of x^u and x^v:
$x = u[\alpha(R)]((u[R_1^*] \odot w[R_1^*]) \ominus (v[R_1^*] \odot w[R_1^*]))...((u[R_{nr}^*] \odot w[R_{nr}^*]) \ominus (v[R_{nr}^*] \odot w[R_{nr}^*]))$
where $(u[(R_i^*)] \odot w[R_i^*]) \ominus (v[R_i^*] \odot w[R_i^*]) \neq \phi$.
By induction assumption,
$x = u[\alpha(R)]((u[R_1^*] \ominus v[R_1^*]) \odot w[R_1^*])...((u[R_{nr}^*] \ominus v[R_{nr}^*]) \odot w[R_{nr}^*])$
where $(u[R_i^*] \odot w[R_i^*]) \ominus (v[R_i^*] \odot w[(R_i^*)]) \neq \phi$
In RHS, since $u[\alpha(R)] = v[\alpha(R)]$, the difference of u and v ($u \neq v$, otherwise x does not exist) in $(r \ominus s)$, denoted y', is
$y' = u[\alpha(R)](u[R_1^*] \ominus v[R_1^*])...(u[R_{nr}^*] \ominus v[R_{nr}^*])$
where $(u[R_i^*] \ominus v[R_i^*]) \neq \phi$
By intersecting y' and w ($u[\alpha(R)] = v[\alpha(R)] = w[\alpha(R)]$), we have a tuple y in RHS as
$y = u[\alpha(R)]((u[R_1^*] \ominus v[R_1^*]) \odot w[R_1^*]))...((u[R_{nr}^*] \ominus v[R_{nr}^*]) \odot w[R_{nr}^*]))$
where $(u[R_i^*] \ominus v[(R_i^*)]) \neq \phi$.
Because $(u[R_i^*] \odot w[R_i^*]) \ominus (v[R_i^*] \odot w[R_i^*]) \neq \phi \Longrightarrow ((u[R_i^*] \ominus v[R_i^*]) \odot w[R_i^*]) \neq \phi \Longrightarrow (u[(R_i^*)] \ominus v[R_i^*]) \neq \phi$ (not vice versa), x equals to y or tuple-contained in y. Therefore x is in RHS.

(b) $x^u[\alpha(R)] \notin (\delta r \odot s)[\alpha(R)]$: then $x = x^u$.
x^u is computed by intersecting $u \in r$ and $w \in s$. That is,
$x = x^u = u[\alpha(R)](u[R_1^*] \odot w[R_1^*])...(u[R_{nr}^*] \odot w[R_{nr}^*])$.
In RHS, $u[\alpha(R)] \notin \delta r[\alpha(R)]$. This can be proved by the inverse method. Suppose there is tuple $v \in \delta r$ such that $v[\alpha(R)] = u[\alpha(R)]$, the intersection of v and w ($v[\alpha(R)] = w[\alpha(R)]$) result in a tuple $x^v \in (\delta r \odot s)$ such that $x^v[\alpha(R)] = u[\alpha(R)] = x^u[\alpha(R)]$. This is a contradiction to the condition of $x^u[\alpha(R)] \notin (\delta r \odot s)[\alpha(R)]$. Thus, $u[\alpha(R)] \notin \delta r[\alpha(R)]$.
From this point, we conclude that there is not tuple in δr affecting u during the difference of $(r \ominus s)$. So the intersection of u and w is the same as x. Consequently, x is in RHS.

This proof of $((r \odot s) \ominus (\delta r \odot s)) \underline{\mathfrak{a}} ((r \ominus s) \odot t)$ is done.

The equation is proved. □

If restrictions are placed on relations r and δr, it is possible to derive IEs in the limited standard form for the intersection operator as shown in the following result.

Lemma 4.6. $(r \ominus \delta r) \odot s = (r \odot s) \ominus (\delta r \odot s)$, *if recursively*
$\exists u \in r \ \wedge \ \exists v \in \delta r \ \wedge \ \exists w \in s \ \wedge \ u[\alpha(R)] = v[\alpha(R)] = w[\alpha(R)] \ \wedge$
$\exists i(u[(R_i)] \ominus v[(R_i)] \neq \phi \ \wedge \ (u[(R_i)] \odot w[(R_i)]) \ominus (v[(R_i)] \odot w[(R_i)]) \neq \phi)$.

The condition attached to the lemma is extracted from the proof of Equation 8. We now use Example 4.4 to show that a standard IE is not valid unless the relations satisfy the condition of Lemma 4.6.

Example 4.4. Let r, δr and s be relations given in Table 17. After recomputation and incremental computation, $(r \ominus \delta r) \odot s \neq (r \odot s) \ominus (\delta r \odot s)$. This is because the first tuple in r, δr and s violates the condition.

Table 17: Relations for Example 4.4

A	B^*
	B
a_1	$\{b_1, b_2\}$
a_2	$\{b_2, b_3\}$
a_3	$\{b_4\}$
a_4	$\{b_5\}$

r

A	B^*
	B
a_1	$\{b_1, b_3\}$
a_2	$\{b_1, b_2\}$
a_3	$\{b_3, b_4\}$

δr

A	B^*
	B
a_1	$\{b_1\}$
a_2	$\{b_2, b_3\}$
a_3	$\{b_4\}$
a_4	$\{b_5\}$

s

A	B^*
	B
a_1	ϕ
a_2	$\{b_3\}$
a_4	$\{b_5\}$

$(r \ominus \delta r) \odot s$

A	B^*
	B
a_2	$\{b_3\}$
a_4	$\{b_5\}$

$(r \odot s) \ominus (\delta r \odot s)$

A closer inspection of this example and the previous results indicates that the reason for not being able to derive a standard IE for the intersection operator is caused by $\alpha(R)$-overlapping tuples in r and δr. If we treat these tuples separately then we can derive IEs in the implementation form that is more efficient than recomputation.

Theorem 4.4. *Let R be a schema and r, δr, and s be instances over R. Then*

$$(r \ominus \delta r) \odot s = \mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(r \odot s) \oplus ((\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r)) \ominus \delta r) \odot s \qquad (10)$$

$$(r \oplus \delta r) \odot t = (r \odot s) \oplus (\delta r \odot s) \qquad (11)$$

The next example shows the usage of Equation 10. The example also shows that Equation 10 is more efficient than recomputation.

Example 4.5. Let r, δr and s be relations given in Table 17. The incremental computation using Equation 10 is given in Table 18. We see that $z_1 \oplus z_2$ is the same as recomputation $(r \ominus \delta r) \odot t$ in Table 17. We note that we did not recompute the intersection of tuple $< a_4, \{b_5\} >$ in r and s. This is where Equation 10 is cheaper than recomputation.

Table 18: Use of Equation 10

A	B^*
	B
a_1	$\{b_1\}$
a_2	$\{b_2, b_3\}$
a_3	$\{b_4\}$
a_4	$\{b_5\}$

$r \odot s$

A
a_1
a_2
a_3

$\delta r[A]$

A	B^*
	B
a_4	$\{b_5\}$

z_1

A	B^*
	B
a_1	ϕ
a_2	$\{b_3\}$

z_2

A	B^*
	B
a_1	ϕ
a_2	$\{b_3\}$
a_4	$\{b_5\}$

$z_1 \oplus z_2$

$z_1 = \mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(r \odot s) \qquad z_2 = ((\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r)) \ominus \delta r) \odot s$

4.5 Incremental Equations for Join Operator

Theorem 4.5. *Let R and S be joinable schemas. Let r and δr be instances over R and s be an instance over S. The following two equations hold.*

$$(r \ominus \delta r)\overset{\circ}{\bowtie} s = (r\overset{\circ}{\bowtie} s) \ominus (\delta r\overset{\circ}{\bowtie} s) \oplus (r \ominus \delta r)\overset{\circ}{\bowtie} s \tag{12}$$

$$(r \oplus \delta r)\overset{\circ}{\bowtie} s = (r\overset{\circ}{\bowtie} s) \oplus (\delta r\overset{\circ}{\bowtie} s) \tag{13}$$

We choose to prove the first equation. The proof of the second equation is similar to that of the first one.

Proof.
Proof of Equation 12: To prove the equation, we only need to prove $(r\overset{\circ}{\bowtie} s) \ominus (\delta r\overset{\circ}{\bowtie} s) \ \underline{\Subset}_n\ (r \ominus \delta r)\overset{\circ}{\bowtie} s$ because of Theorem 3.1.

Let $R = \alpha(R)\{R_1^*, ..., R_{nr}^*\}$ and $S = \alpha(R)\{S_1^*, ..., S_{ns}^*\}$. Let r and δr be relations on R and s be a relation on S. Let $T = \alpha(R)\{T_1^*, ..., T_{nt}^*\}$ be joined schema of R and S.

(1) Base case: when $R = S = T = \alpha(R)$, i.e. all schemas are flat and the same, $(r\overset{\circ}{\bowtie} s) \ominus (\delta r\overset{\circ}{\bowtie}_r s) = (r \ominus \delta r)\overset{\circ}{\bowtie} s$. This is because when relations are flat, join operation is equivalent to set intersection operation. The equivalence is proved in [13].

(2) Induction: we assume that for $u \in r \ \wedge\ v \in \delta r \ \wedge\ w \in s$, $(u[R_j^*]\overset{\circ}{\bowtie} w[(S_k)]) \ominus (v[R_j^*]\overset{\circ}{\bowtie} w[(S_k)]) \ \underline{\Subset}_n\ (u[R_j^*] \ominus v[R_j^*])\overset{\circ}{\bowtie} w[(S_k)]$. We prove $(r\overset{\circ}{\bowtie} s) \ominus (\delta r\overset{\circ}{\bowtie} s) \ \underline{\Subset}_n\ (r \ominus \delta r)\overset{\circ}{\bowtie} s$.
For $x \in ((r\overset{\circ}{\bowtie} s) \ominus (\delta r\overset{\circ}{\bowtie} s))$ in LHS, there exist $x^u \in (r\overset{\circ}{\bowtie} s)$ and $x^v \in (\delta r\overset{\circ}{\bowtie} s)$ such that x is the difference of x^u and x^v. By the definition difference, we have

$$\begin{aligned}
&(r\overset{\circ}{\bowtie} t) \ominus (s\overset{\circ}{\bowtie} t)\\
&=\{x | x[\alpha(R)] = x^u[\alpha(R)] = x^v[\alpha(R)] \ \wedge\ \forall\, i(x[T_i^*] = x^u[T_i^*] \ominus x^v[T_i^*]) \ \wedge\\
&\qquad \exists\, i(x[T_i^*] \neq \phi) \text{ or}\\
&\quad\ x[\alpha(R)] = x^u[\alpha(R)] \neq \forall\, x^v[\alpha(R)] \ \wedge\ x = x^u \qquad\qquad \}
\end{aligned}$$

where by the definition of join,

$$\begin{aligned}
x^u \in \{x^u | &\exists\, u \in r \ \wedge\ \exists\, w \in s \ \wedge\ x^u[\alpha(R)] = u[\alpha(R)] = w[\alpha(R)] \ \wedge\ \forall\, i\\
&(x^u[T_i^*] = u[R_j^*]\overset{\circ}{\bowtie} w[S_k^*], \text{ if } R_j \underline{\Subset}^p T_i \ \wedge\ S_k \underline{\Subset}^p T_i \text{ or}\\
&\ x^u[T_i^*] = u[R_j^*], \text{ if } R_j \underline{\Subset}^p T_i \ \wedge\ \nexists\, S_k \Subset T_i \text{ or}\\
&\ x^u[T_i^*] = w[S_k^*], \text{ if } \nexists\, R_j \Subset T_i \ \wedge\ S_k \underline{\Subset}^p T_i\ ) \qquad \} \text{ and}\\
x^v \in \{x^v | &\exists\, v \in \delta r \wedge \exists\, w \in s \ \wedge\ x^v[\alpha(R)] = v[\alpha(R)] = w[\alpha(R)] \ \wedge\ \forall\, i\\
&(x^v[T_i^*] = v[R_j^*]\overset{\circ}{\bowtie} w[S_k^*], \text{ if } R_j \underline{\Subset}^p T_i \ \wedge\ S_k \underline{\Subset}^p T_i \text{ or}\\
&\ x^v[T_i^*] = v[R_j^*], \text{ if } R_j \underline{\Subset}^p T_i \ \wedge\ \nexists\, S_k \Subset T_i \text{ or}\\
&\ x^v[T_i^*] = w[S_k^*], \text{ if } \nexists\, R_j \Subset T_i \ \wedge\ S_k \underline{\Subset}^p T_i\ ) \qquad \}
\end{aligned}$$

We replace x^u and x^v with their definition in the difference. Because the schema of x_u and x_v are the same, each case in x^u matches that in x^v, and does not match any of the other two cases. For this reason, we first consider the first case of x_u and the first case of x_v in the first case of the difference.

We then consider the second and third cases of x_u and x_v respectively in the first case of the difference. After that, we consider each case of x_u and x_v in the second case of the difference. From this procedure, we get the following expression. In the expression, for the simplicity, we do not write the schema conditions explicitly below.

$$
\begin{array}{l}
(r \overset{\circ}{\bowtie} s) \ominus (\delta r \overset{\circ}{\bowtie} s) \\
=\{x | (\exists\, u \in r \ \wedge\ \exists\, v \in \delta r \ \wedge\ \exists\, w \in s \wedge \\
\qquad x[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] = w[\alpha(R)] \wedge \\
\qquad \forall\, i(x[T_i^*] = (u[R_j^*] \overset{\circ}{\bowtie} w[S_k^*]) \ominus (v[R_j^*] \overset{\circ}{\bowtie} w[S_k^*]) \text{ or} \\
\qquad\quad x[T_i^*] = u[R_j^*] \ominus v[R_j^*] \text{ or} \\
\qquad\quad x[T_i^*] = w[S_k^*] \ominus w[S_k^*]) \wedge \\
\qquad \exists\, i(x[T_i^*] \neq \phi) \qquad\qquad\qquad \text{or} \\
\quad (\exists\, u \in r \ \wedge\ \forall\, v \in \delta r \ \wedge\ \exists\, w \in s \wedge \\
\qquad x[\alpha(R)] = u[\alpha(R)] = w[\alpha(R)] \neq v[\alpha(R)] \wedge \\
\qquad \forall\, i(x[T_i^*] = (u[R_j^*] \overset{\circ}{\bowtie} w[S_k^*]) \text{ or} \\
\qquad\quad x[T_i^*] = u[R_j^*] \text{ or} \\
\qquad\quad x[T_i^*] = w[S_k^*]) \qquad\qquad\qquad \}
\end{array}
$$

Now we consider $(r \ominus \delta r) \overset{\circ}{\bowtie} s$ in RHS. Let $y \in (r \ominus \delta r) \overset{\circ}{\bowtie} s$ and $y^u \in (r \ominus \delta r)$. By the definition of join, we have

$$
\begin{array}{l}
(r \ominus \delta r) \overset{\circ}{\bowtie} s \\
=\{y | y[\alpha(R)] = y^u[\alpha(R)] = w[\alpha(R)] \wedge \\
\quad \forall\, i(y[T_i^*] = y^u[R_j^*] \overset{\circ}{\bowtie} w[S_k^*], \text{ if } R_j \underline{\Subset}^p T_i \ \wedge\ S_k \underline{\Subset}^p T_i \text{ or} \\
\qquad y[T_i^*] = y^u[R_j^*], \text{ if } R_j \underline{\Subset}^p T_i \ \wedge\ \nexists\, S_k \Subset T_i \text{ or} \\
\qquad y[T_i^*] = w[S_k^*], \text{ if } \nexists\, R_j \Subset T_i \ \wedge\ S_k \underline{\Subset}^p T_i \qquad) \qquad \}
\end{array}
$$

where by the definition of difference,

$$
\begin{array}{l}
y^u \in \{y^u | \exists\, u \in r \ \wedge\ \exists\, v \in \delta r \ \wedge\ y^u[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] \wedge \\
\qquad\qquad \forall\, j(y^u[R_j^*] = u[R_j^*] \ominus v[R_j^*]) \ \wedge\ \exists\, j(u[R_j^*] \ominus v[R_j^*] \neq \phi) \text{ or} \\
\qquad\quad \exists\, u \in r \ \wedge\ \forall\, v \in \delta r \ \wedge\ y^u[\alpha(R)] = u[\alpha(R)] \neq v[\alpha(R)] \ \wedge\ y^u = u \}
\end{array}
$$

Replace y^u with its definition in the RHS,

$$
\begin{array}{l}
(r \ominus \delta r) \overset{\circ}{\bowtie} s \\
=\{y | \exists\, u \in r \wedge \exists\, v \in \delta r \ \wedge\ \exists\, w \in s \wedge \\
\qquad\quad y[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] = w[\alpha(R)] \wedge \\
\qquad\quad \forall\, i(y[T_i^*] = (u[R_j^*] \ominus v[R_j^*]) \overset{\circ}{\bowtie} w[S_k^*] \text{ or} \\
\qquad\qquad y[T_i^*] = u[R_j^*] \ominus v[R_j^*] \text{ or} \\
\qquad\qquad y[T_i^*] = w[S_k^*]\ ) \\
\qquad\quad \exists\, j(u[R_j^*] \ominus v[R_j^*] \neq \phi) \text{ or} \\
\quad \exists\, u \in r \ \wedge\ \forall\, v \in \delta r \ \wedge\ \exists\, w \in s \wedge \\
\qquad\quad y[\alpha(R)] = u[\alpha(R)] = w[\alpha(R)] \neq v[\alpha(R)] \wedge \\
\qquad\quad \forall\, i(y[T_i^*] = u[R_j^*] \overset{\circ}{\bowtie} w[S_k^*] \text{ or} \\
\qquad\qquad y[T_i^*] = u[R_j^*] \text{ or} \\
\qquad\qquad y[T_i^*] = w[S_k^*]\ ) \qquad\qquad\qquad \}
\end{array}
$$

Now we compare x and y. The second case of x equals to the second case

of y with conditions. With the induction assumption, the first case of x equals to the first case of y with the condition that $\exists\, i(x[T_i^*] \neq \phi) \iff \exists\, j(u[R_j^*] \ominus v[R_j^*] \neq \phi)$.
We now prove that $\exists\, i(x[T_i^*] \neq \phi) \Longrightarrow \exists\, j(u[R_j^*] \ominus v[R_j^*] \neq \phi)$ but not vice versa. In the first case of x, $x[T_i^*]$ is computed in two ways (note that $w[S_k^*] \ominus w[S_k^*]$ always results ϕ while we are discussing not being ϕ).

- $x[T_i^*] = u[R_j^*] \ominus v[R_j^*] \neq \phi$. This is equivalent to $\exists\, (u[R_j^*] \ominus v[R_j^*]) \neq \phi$ of y side.
- $x[T_i^*] = (u[R_j^*] \mathring{\bowtie} w[S_k^*]) \ominus (v[R_j^*] \mathring{\bowtie} w[S_k^*]) \neq \phi$. By the induction assumption, we have $x[T_i^*] = (u[R_j^*] \ominus v[R_j^*]) \mathring{\bowtie} w[S_k^*] \neq \phi$ By the definition of join, $(u[R_j^*] \ominus v[R_j^*]) \mathring{\bowtie} w[S_k^*] \neq \phi \Longrightarrow (u[R_j^*] \ominus v[R_j^*]) \neq \phi$.
  However, from $(u[R_j^*] \ominus v[R_j^*]) \neq \phi$ we can not get $(u[R_j^*] \ominus v[R_j^*]) \mathring{\bowtie} w[S_k^*] \neq \phi$ because the join of two non-empty sets can be an empty set. Consequently, $(u[R_j^*] \ominus v[R_j^*]) \neq \phi \not\Longrightarrow (u[R_j^*] \mathring{\bowtie} w[S_k^*]) \ominus (v[R_j^*] \mathring{\bowtie} w[S_k^*]) \neq \phi$.

Because the condition of x can lead to the condition of y to be true, any x in LHS is contained or tuple-contained in RHS. However, the condition of y can not always lead to the condition of x to be true, y is not always included in RHS.
In summary, we proved that $(r \mathring{\bowtie} s) \ominus (\delta r \mathring{\bowtie} s) \overset{t}{\subseteq} (r \ominus \delta r) \mathring{\bowtie} s$.

$\square$

Equation 12 can be simplified if restrictions are placed on the update δr.

Lemma 4.7. *Let R and S be two joinable schemas. Let r and δr be instances over R and s be an instance over S. The following equation holds.*
$(r \ominus \delta r) \mathring{\bowtie} s = (r \mathring{\bowtie} s) \ominus (\delta r \mathring{\bowtie} s)$ *iff r and δr are $\alpha(R)$-disjoint.*

The proof of the lemma is similar to the proof of Equation 12. We now give an example to show the importance of the condition in the lemma.

Example 4.6. Let r, δr, and s be three relations given in Table 19. The recomputation $(r \ominus \delta r) \mathring{\bowtie} s$ and the incremental computation $(r \mathring{\bowtie} s) \ominus (\delta r \mathring{\bowtie} s)$ of the join operator are also included in the table. Because r and δr have $\alpha(R)$-overlapping tuples, the recomputation contains more tuple than incremental computation does.

Consider the performance and implementation, as we analyzed in Subsection 4.3, we give the following IEs in implementatation form. In the theorem, $\alpha(R)$-overlapping tuples in r and δr are recomputed while other tuples in δr are incrementally computed.

Theorem 4.6. *Let R and S be two joinable schemas. Let r and δr be instances over R and s be an instance over S. The following two equations hold.*

$$(r \ominus \delta r) \mathring{\bowtie} s = \mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(r \mathring{\bowtie} s) \oplus ((\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r) \ominus \delta r) \mathring{\bowtie} s) \quad (14)$$
$$(r \oplus \delta r) \mathring{\bowtie} s = (r \mathring{\bowtie} s) \oplus (\delta r \mathring{\bowtie} s) \quad (15)$$

Table 19: Relations for Example 4.6

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{c_1\}$
a_2	$\{b_1\}$	$\{c_2, c_3\}$
a_3	$\{b_1, b_2\}$	$\{c_1\}$

r

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{c_1\}$
a_2	$\{b_1\}$	$\{c_3\}$
a_3	$\{b_2, b_3\}$	$\{c_1\}$

δr

A	C^*	D^*
	C	C
a_1	$\{c_1\}$	$\{d_1\}$
a_2	$\{c_1\}$	$\{d_2\}$
a_3	$\{c_1\}$	$\{d_3\}$

s

A	B^*	C^*	D^*
	B	C	D
a_2	ϕ	ϕ	$\{d_2\}$
a_3	$\{b_1\}$	ϕ	$\{d_3\}$

$(r \ominus \delta r)\mathring{\bowtie} s$

A	B^*	C^*	D^*
	B	C	D
a_3	$\{b_1\}$	ϕ	ϕ

$(r\mathring{\bowtie} s) \ominus (\delta r\mathring{\bowtie} s)$

4.6 Discussion

In this subsection, we highlight some of the important features of the IEs derived in the last section and discuss the differences between incremental expressions for flat relations and for nested relations.

Firstly, we note that some (but not all) of the incremental expressions are in the standard form since the expressions do not involve recomputing the new view. The performance gain of this type of IEs is obvious. This is the case where the update is an insertion and the operators are expansion, intersection, projection, or join. However, the IEs for some operators, e.g. Equation 8, involve recomputation. These types of IEs do not avoid view recomputation unless restrictions are placed upon the update. A similar situation occurs with the flat relational projection operator where the incremental expression involves view recomputation [13]. This highlights the fact that some views are impossible to maintain efficiently if the only information available is the view itself. This has lead to the development of other techniques which use counts [6] or auxiliary relations [19] to improve efficiency by avoiding view recomputation. Similarly, one expects that views involving nested operators can be more efficiently maintained if more information than just the view is stored.

In comparing the equations derived in the last section and those in [13] for the flat relational model, one notes that not only are the equations in this paper generally more complex but also the symmetry shown in the equations of [13] is absent in the expressions of the nested operators. For example, in flat relations the incremental expressions for selections and joins are similar for both insertions and deletions and are computed as $\mathring{\sigma}_c(r \pm s) = \mathring{\sigma}_c(r) \pm \mathring{\sigma}_c(s)$ and $(r \pm s)\mathring{\bowtie} t = r\mathring{\bowtie} t \pm s\mathring{\bowtie} t$ respectively. This symmetry is absent in the equations derived in Section 4.

Apart from the difference of patterns of IEs, the containment and disjointedness properties used in deriving IEs for flat relations and those used for nested relations are different. The containment and disjointedness for flat relations are defined on set semantics. A tuple contained in a set means that the tuple is a member of the set. In nested relations, however, a tuple is contained in a nested relation does not mean that the tuple exists in the relation. The tuple may be contained in a tuple of the relation. Tuple containment becomes a core concept in the containment for nested relations. Similarly, the disjointedness for nested relations is built on the basis that two tuples are disjoint if one tuple has a subrelation that is disjoint from the corresponding subrelation of the other tuple.

Compared with the equations for flat relations, our equations in the implementation form incrementally compute the view update to the maximum extent and avoid full view recomputation by recomputing only key attribute overlapping tuples. The performance gain from the implementation form is obvious. This is because in the implementation form, we make full use of indexes on the top level to select key attribute overlapping tuples. We avoid to test the complex conditions of IEs in the limited standard form and avoid full view recomputation.

5 Incremental Maintenance of Views with Complex Queries

In this section, we present a technique for the incremental maintenance of PNF views expressed as complex queries using the incremental equations derived previously. Our technique can handle both insertions and deletions. However, for simplicity of exposition we assume that the update is a single insertion to a base relation.

Our technique is firstly to represent the query expression for the view as an expression or operator tree. In this representation, the leaves of the tree are the base relations, the interior nodes are the query operators and the root of the tree represents the final view. Our technique then computes the change to the view in a 'bottom' up fashion starting with the changes to the leaf nodes and then propagating the changes upwards in the expression tree using the incremental equations derived previously. To be more precise, we can express the technique in the following algorithm (ir_i represents the intermediate relation corresponding to node i in the tree, $ir_{i'}$ and $ir_{i''}$ are the intermediate relations corresponding to the child nodes of node i).

Algorithm 5.1. (Maintaining views with complex queries)

```
Input: the operator tree and the insertions $\delta r$ to $r$
Output: update to the view
Do: For each node $i$ do
        if $\delta ir_{i'}$ or $\delta ir_{i''}$ is non-empty, then
          compute $\delta ir_i$ according to IEs derived in previous sections.
        endif;
       endfor;
```

The use of the algorithm is illustrated by the next example.

Example 5.1. Let relation *pstud* be defined in Table 1. Let relation *subjLect* be defined in Table 20. We define a query to list

- the name of a student;
- the subject taken by the student in 1998 if the student gets all marks over 69 for every test;
- the lectures for the taken subject.

This query can be expressed as

$$View = \mathring{\pi}_{\{Name,Subj^*:\{sjName,Year,Lect^*:\{Lect\}\}\}}\big((\mathring{\sigma}_{Subj^*:(Year=98\ \wedge\ Marks:(Mark>79)\neq\phi)\neq\phi}(pstud)) \overset{\circ}{\bowtie} subjLect) \ \big)$$

Table 21 shows the instance of the view.

Table 20: Relation *subjLect*

Name	*Subjs**		
	sjName	*Year*	*Lects**
			Lect
Jack	{ *DB*	98	{*Ben*, *Tom*}
	DB	96	{*Kaven*} }
John	{ *DB*	98	{*Ben*, *Tom*}
	DB	96	{*Kaven*} }

Table 21: The materialized view *View* before update

Name	*Subjs**		
	sjName	*Year*	*Lects**
			Lect
Jack	{ *DB*	98	{*Ben*, *Tom*}}

We now update relation *pstud*. The update to *pstud* is an insertion $\Delta pstud$ given in Table 22. The update is propagated through the following steps.

- $\Delta V_3 = \mathring{\sigma}_{Subj^*:(Year=98\ \wedge\ Marks:(Mark>79)\neq\phi)\neq\phi}(\Delta pstud)$
  $=\{< Sean, \{< DB,\ 98,\ \{< exam,\ 90 >\} >\}, \phi >\}$
- $\Delta V_2 = \Delta V_3 \overset{\circ}{\bowtie} subjLect$
  $=\{< Sean, \{< DB,\ 98,\ \{< exam,\ 90 >\}, \{< Ben >, < Tom >\} > \}, \phi >\}$
- $\Delta V_1 = \mathring{\pi}_{\{Name,Subj^*:\{sjName,Year,Lect^*:\{Lect\}\}\}}(\Delta V_2)$
  $=\{< Sean, \{< DB,\ 98,\ \{< Ben >, < Tom >\} >\} >\}$
- $View = View \oplus \Delta V_1$ is given in Table 23.

In the above procedure, we only computed the view update having the tuple of 'Sean'. We did not compute the tuple 'Jack' that had been in the old view. When tuples in the old view are numerous, our way of maintaining the view can have better performance than recomputation.

Table 22: An update Δr to r

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Sean	{*DB*	98	{*exam*	90} }	ϕ

Table 23: The materialized view *View* after update

Name	*Subjs**		
	sjName	*Year*	*Lects**
			Lect
Jack	{ *DB*	98	{*Ben*,*Tom*}}
Sean	{ *DB*	98	{*Ben*,*Tom*}}

6 Implementation and Performance Analysis of IEs

In this section, we present the results of our performance analysis of the IEs we derived. A detailed description of implementation can be found in [12].

6.1 The database

We employed a university database for the implementation. It contains five relations. The schema of each relation is described in Figure 24. The schema *Stud* has been described in Example 1. The schema *Teach* describes the lecturer names (*lcName*) for each subject in a year. *Lect* is a schema modeling the information of lecturers. *Test* is the schema to describe the test details for a subject in a year. The last schema, *Hobby*, describes hobbies of a student.

Table 24: Schemas of the university database

Name	*Subjs**				*Addrs**
	sjName	*Year*	*Marks**		*Addr*
			TestName	*Mark*	

schema *Stud* of relation *pstud*

sjName	*Year*	*lcNames**
		lcName

schema *Teach* of relation *teach*

lcName	*Salary*	*Speciality*

schema *Lect* of relation *lc*

sjName	*Year*	*TestNames**	
		TestName	*Description*

schema *Test* of relation *test*

Name	*Hobbies**
	Hobby

schema *Hobby* of relation *hobby*

The implementation was performed on a Pentium 166 PC computer with two hard disks, 96 MB of memory, and Microsoft Windows NT 4.0 operation system.

The database management system used was the Informix Database Server with Universal Data Operation (IDS/UDO) version 9.14. Clients interface and programs are connected to the server by TCP/IP *loop-back* connection [9], which is the only option for Windows NT platform. The query language we use for the implementation is the OR-SQL proposed in [17].

6.2 The Cost Model

The cost model for the performance analysis involves the costs of view creation, incremental maintenance, and recomputation. Each cost is the time for computing an item in an IE. We use the IE for the join operator with a deletion update as an example to show the relationship between the costs in the cost model and the items of the equations following.

$$\underbrace{(r \ominus \delta r) \overset{\circ}{\bowtie} s}_{c_{rec}} = \underbrace{\underbrace{\overset{\circ}{\sigma}_{\alpha(R)\notin\delta r[\alpha(R)]}(\overbrace{r \overset{\circ}{\bowtie} s}^{c_{cre}})}_{c_{del}} \underbrace{\oplus}_{c_{ins}} (\underbrace{\overset{\circ}{\sigma}_{\alpha(R)\in\delta r[\alpha(R)]}(r)}_{c_{ovl}} \underbrace{\ominus}_{c_{cmb}} \delta r) \underbrace{\overset{\circ}{\bowtie}}_{c_{inc}} s}_{c_{mtn}} \tag{16}$$

We now detail each cost.

- c_{cre} is the time for creating the materialized view $r \overset{\circ}{\bowtie} s$.
- c_{rec}, on the left hand side, is the time for recomputing the view when an update happens to a deriving relation of the view.
- c_{mtn}' on the right hand side, is the time for incrementally maintaining the view using right hand side of incremental equations when an update happens to a deriving relation of the view. This time consists of the following components.
  - c_{del} is the time for deleting from the old view the tuples derived from $\alpha(R)$-overlapping tuples in r.
  - c_{ins} is the time for inserting the tuples of the view update into the view. Since the tuples in the view update are $\alpha(R)$-disjoint with the view because of the select operation against r, this insertion in fact is the set operation.
  - c_{ovl} is the time for selecting $\alpha(R)$-overlapping tuples from r, the relation that is being updated.
  - c_{cmb} labels the time to conduct PNF union or difference between $\alpha(R)$-overlapping tuples of r and δr.
  - c_{inc} denotes the time for computing the view update using the operator that defines the view (e.g., $\overset{\circ}{\bowtie}$).

After defining all the costs, the total maintenance time is given by:

$$c_{mtn} = c_{del} + c_{ins} + c_{ovl} + c_{cmb} + c_{inc}$$

where costs of c_{del}, c_{ins}, c_{ovl}, and c_{cmb} are not operator specific.

With this cost model, a typical performance analysis diagram is like the one shown in Figure 2. The horizontal axis indicates the different update sizes while the vertical axis indicates the relative time to view creation. There are three lines in the diagram. One is labeled by 'rec' and shows the relative time of view recomputation: c_{rec}/c_{cre}. It goes downward from top-left corner when updates are deletions. When the update size reaches 50% of the original size, it should come down to 50% along the vertical axes.

The second line is labeled by 'inc' and shows the relative time of c_{inc}/c_{cre}. It goes up from the lower-left corner. This line is drawn by using updates that do not have A-overlapping tuples with r. Because there are no A-overlapping tuples in the update, no recomputation and no deletion from the old view are needed for the incremental maintenance. Therefore, It is an ideal line for incremental maintenance. When the size of the update reaches 50%, this line will cross with 'rec' at 50% of the vertical axis. This line and the location of the cross serve to check the correct of the implementation programs.

The third line labeled by 'mtn' is the relative time for general incremental maintenance: c_{mtn}/c_{cre}. It goes up from the lower-left corner. The intersection of the two lines 'rec' and 'mtn' being located at over 50% of the vertical axis and less than 50% of the horizontal axis. The horizontal coordinate of the the intersection point is called the *maintenance limit.* It is the size of an update with which the time of incremental maintenance is equivalent to the time of view recomputation.

The 'mtn' line in the figure is the worst case where all tuples in δr are $\alpha(R)$-overlapping with r and produce tuples in the view update to be inserted into the old view. Lines 'mtn' and 'inc' are the minimum and maximum boundaries for the incremental maintenance. The actual maintenance limit, depending on the update type and $\alpha(R)$-overlapping property, falls within the boundaries.

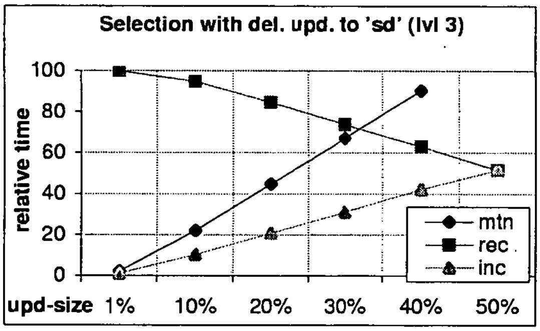

Figure 2: The performance analysis for selection on 3rd level

6.3 Maintenance Limit

In this section, we analyze the maintenance limit for the selection operator and the join operator.

6.3.1 Selection Operator

To study the maintenance limit for the selection operator, we simulate a query of listing all tuples of table *stud* if a student has at least one good mark (≥ 90) for at least one subject. The query is

```
Qs: SELECT * FROM stud s WHERE EXISTS
        (SELECT * FROM table(s.Subjs*) j WHERE EXISTS
            (SELECT * FROM table(j.Marks*) WHERE Mark≥ 90) );
```

The selection condition $Mark \geq 90$ in the query is set up on third level of relation *stud* with the selectivity being 10%.

The view defined with this query is maintained using the right hand side of Equation 6.

The performance analysis diagram has been given Figure 2. From the diagram, we see that the maintenance limit is about 32%.

We also analyzed the maintenance limits for the cases where selection conditions are on the first level and the second level respectively. The maintenance limits for selection condition on all levels are listed in Table 25.

Table 25: Maintenance limits for selection

condition level	1st	2nd	3rd
limits(%)	18	40.5	32

6.3.2 Join Operator

The query we study the maintenance limit of join operator is Q_{jn}.

```
Qjn: SELECT * FROM stud s WHERE EXISTS
     (SELECT * FROM table(s.Subjs*) j, test t
       WHERE j.sjName=t.sjName AND j.Year=t.Year
       AND EXISTS (SELECT * FROM table(j.Marks*) a, table(t.TestNames*) b
         WHERE a.TestName=b.TestName) );
```

In this query, *test* joins *Stud* on the second level and the third level of *Stud*.

We also simulated join operations on the first level and the second. The tree presentation of the join in the three levels is given in Figure 3. The maintenance limits for the three case is given in Table 26.

Table 26: Maintenance limits for join

condition level	1st	2nd	2nd & 3rd
limits(%)	21	44	36

The data in the table is quite similar to that of Table 25. So, we omit the explanation.

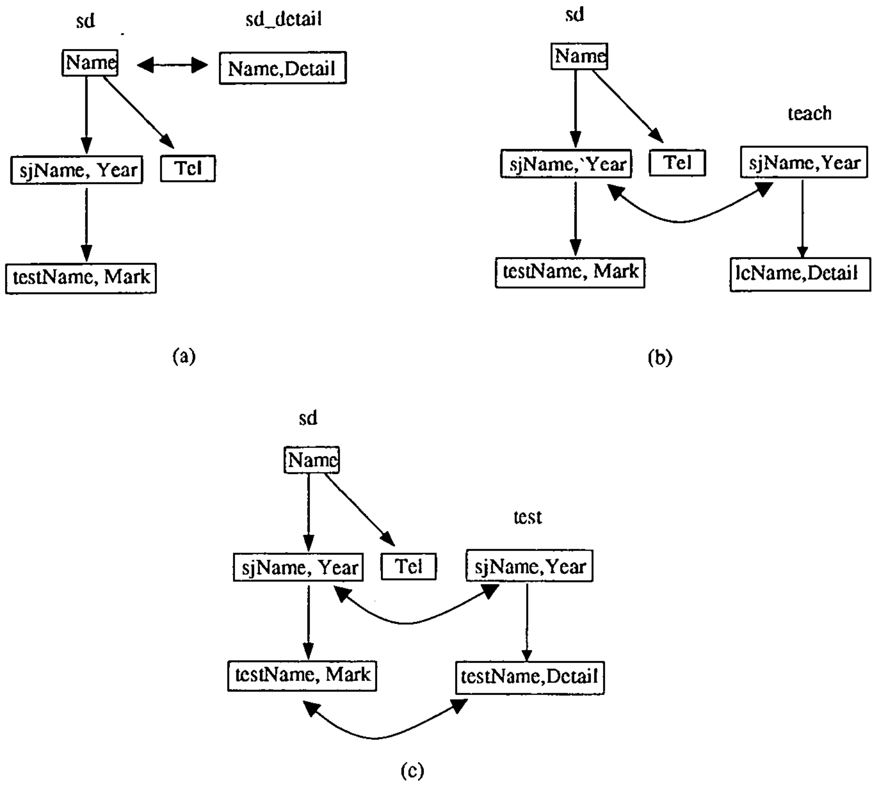

Figure 3: Joins on different levels

The above data is obtained by applying updates to table *stud*, which we call the left operand of the join. We now consider the cases where updates are applied to the right operand of the join.

Line 'inc-lv3' in Figure 6.3.2 is the case where the right operand joins *stud* on the third level. The figure shows that the incremental maintenance cost does not vary with the change of the size of updates to the right operand. This maintenance cost is almost the same as the view creation time. This is because the navigation of *stud* down to the third level consumes most of the time of the join operation. Line 'inc-lv2' and Line 'inc-lv1' of the figure indicate that as the level on which the right operand joins *stud* becomes shallower, the cost of the incremental maintenance changes toward the trend of updates to *stud*.

7 Conclusion

In this paper, we derived IEs for the operators of PNF nested relations. We derived IEs in three forms: the standard form, the limited standard form, and the imple-

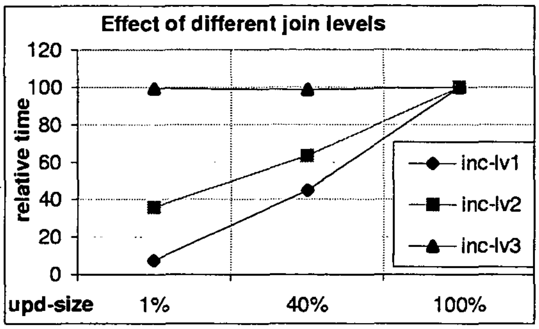

Figure 4: The performances when other joining tables are updated

mentation form. The standard form is the ideal form that we first aimed to achieve. If an IE can not be in the standard form, we proposed the limited standard form. This form aims to reveal the reason why the IE can not be standard. After the limited standard form, by considering performance of testing complex conditions, we have derived IEs in the implementation form to avoid testing such conditions.

In this paper, we also implemented IEs for the nested relations in the Informix Universal Database Server. A database with multi-level nested relations was created in the database server. We implemented the PNF operators using ESQL/C functions. With the operators and the relations in the database, views were created and the incremental equations are implemented. Afterward, the performance of each incremental equation was analyzed.

The performance analysis show that the PNF union and difference operations are the main reasons causing performance decrease of the incremental computation. Generally, the maintenance limits of the incremental equations are between 17–44%. As the number of nested levels increase, the maintenance limit decreases.

Nested relations are closely related to the new emerged semi-structured data protocol XML (eXtensible Markup Language) [1]. Because of this, the IE expressions derived in this paper have the potential to be adapted for the use in maintaining XML views. However, the adaption will not be direct because XML allows missing elements and flexible structures. This leads to null sub relations which challenge the adaption. We leave this as future research work.

References

[1] Serge Abiteboul. On views and xml. *PODS*, pages 1-9, 1999.

[2] Serge Abiteboul and Nicole Bidoit. Non first normal form relations: an algebra allowing data restructuring. *Journal of Computer and System Sciences*, 33(3):361-393, 1986.

[3] Stijn Dekeyser, Bart Kuijpers, and Jan Paredaens. Nested data cubes for olap. *LNCS 1552, Advances in Database Technologies*, 1998.

[4] Timothy Griffin and Leonid Libkin. Incremental maintenance of views with duplicates. *SIGMOD Conference*, pages 328–339, 1995.

[5] Ashish Gupta and Inderpal Mumick. *Materialized Views - Techniques, Implementation, and Applications.* The MIT Press, 1999.

[6] Ashish Gupta and Inderpal S. Mumick. Maintaining view incrementally. *SIGMOD Conference*, 1993.

[7] Ashish Gupta and Inderpal S. Mumick. Maintenance of materialized views: problems, techniques and applications. *IEEE Data Engineering Bulletin, special issues on materialized views and data warehousing*, 18(2), 1996.

[8] Richard Hull. A survey of theoretical research on typed complex database objects. In J. Paredaens, editor, *Databases*, pages 193–255. Academic Press, London, 1987.

[9] Informix. Informix-universal server administrator's guide, version 9.1. 1997. Informix Corporation.

[10] Mark Levene. *LNCS 595, The Nested Universal Relation Database Model.* Springer-verlag, Berlin, 1992.

[11] Jixue Liu and Millist Vincent. Containment and disjointedness in partitioned normal form relations. *Acta Informatica*, 38:325–342, 2002.

[12] Jixue Liu, Millist Vincent, and Mukesh Mohania. Implementation and performance analysis of incremental equations for nested relations. *IDEAS*, pages 398–404, 2000.

[13] Xiaolei Qian and Wiederhold Gio. Incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, 3(3):337–341, 1991.

[14] M. A. Roth, H. F. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational database. *ACM Transactions on Database Systems*, 13(4):389–417, 1988.

[15] M A Roth, H F Korth, and A Silberschatz. Null values in nested relational databases. *Acta Informatica*, 26(7):615–642, 1989.

[16] Mark A. Roth and James E. Kirkpatrick. Algebras for nested relations. *Data Engineering Bulletin*, 11(3):39–47, 1988.

[17] M. Stonebraker and Paul Brown. *Object-relational DBMSs tracking the next great wave.* Morgan Kaufmann Publishers, Inc. California, 1999.

[18] Michael Stonebraker and Dorothy Moore. *Object Relational DBMSs, the Next Great Wave.* Morgan Kaufman publishers Inc., 1996.

[19] M. Vincent, M. K. Mohania, and Y. Kambayashi. A self maintainable view maintenance technique for data warehouses. *COMAD*, pages 7–22, 1997.

[20] Jennify Widom. Research problems in data warehousing. *CIKM*, pages 25–30, 1995.

[21] Jun Yang and Jennifer Widom. Maintaining temporal views over non-historical information sources for data warehousing. *EDBT*, pages 389–403, 1998.

Received February, 2002

Velocity and Distance of Neighbourhood Sequences

András Hajdu[*] and Lajos Hajdu[†]

Abstract

Das et al. [2] defined the notion of periodic neighbourhood sequences. They also introduced a natural ordering relation $\sqsupseteq^*$ for such sequences. Fazekas et al. [4] generalized the concept of neighbourhood sequences, by dropping periodicity. They also extended the ordering to these generalized neighbourhood sequences. The relation $\sqsupseteq^*$ has some unpleasant properties (e.g., it is not a complete ordering). In certain applications it can be useful to compare any two neighbourhood sequences. For this purpose, in the present paper we introduce a norm-like concept, called velocity, for neighbourhood sequences. This concept is in very close connection with the natural ordering relation. We also define a metric for neighbourhood sequences, and investigate its properties.

Keywords: Digital Geometry, Neighbourhood Sequences, Distance, Metric
PACS 68U10

1 Introduction

Distance functions are used in many parts of digital geometry. They are usually defined by digital motions, when we can move in the digital space from one point to another, if they are neighbours in some sense. Rosenfeld and Pfaltz [9] introduced two types of motions in $\mathbb{Z}^2$, the cityblock and chessboard motions. The cityblock motion allows only horizontal and vertical steps, while the chessboard motion diagonal movements as well. By these motions Rosenfeld and Pfaltz defined the distances d_4 and d_8, respectively, as the number of steps needed to get from one point to another. To obtain a better approximation of the Euclidean distance they recommended the alternate use of the cityblock and chessboard motions.

[*]Institute of Mathematics and Informatics, University of Debrecen, H-4010 Debrecen P.O.Box 12. Email: hajdua@math.klte.hu
Research supported by the OTKA grant T032361.
[†]Institute of Mathematics and Informatics, University of Debrecen, H-4010 Debrecen P.O.Box 12. Email: hajdul@math.klte.hu
Research supported in part by the Netherlands Organization for Scientific Research (NWO), the Hungarian Academy of Sciences, by grants T029330 and F034981 of the Hungarian National Foundation for Scientific Research (HNFSR), and by the FKFP grant 3272-13/066/2001.

By allowing arbitrary periodic mixture of the cityblock and chessboard motions, Das et al. [2] introduced the concept of periodic neighbourhood sequences, and generalized it to arbitrary dimension. A distance function can be attached to any neighbourhood sequence A by defining the distance of two points as the number of A-steps needed to get from one of them to the other. In [2] the authors provided a criterion to decide when the distance function corresponding to A is a metric. They also introduced a natural ordering relation on the set of periodic neighbourhood sequences in the following way. Given two such sequences A and B, A is "faster" than B, if for every two points the A-distance is less than or equal to the B-distance of these points. The name of this relation expresses that in this case A spreads faster in the digital space than B. Das [1] studied the structure of the set of periodic neighbourhood sequences with respect to this natural ordering.

By dropping the condition of periodicity, Fazekas et al. [4] generalized the concept of neighbourhood sequences. They extended the "faster" relation to these generalized neighbourhood sequences, and investigated its properties. It turned out that this natural ordering has some unpleasant properties. It fails to be a complete ordering on the set of neighbourhood sequences, moreover, the structure obtained is not even a lattice in higher dimension. However, in certain applications it can be useful to compare any two neighbourhood sequences, i.e. to decide which one spreads "faster". For this purpose, in this article we introduce a norm-like concept, called velocity, on the set of neighbourhood sequences, and investigate its properties. This concept has to be introduced in a way to fit the relation "faster", so we need some preliminaries before defining velocity. Further, we define a metric for neighbourhood sequences. In practical applications it is usually sufficient to consider a finite part of a neighbourhood sequence. In our definition of velocity this can be easily reached by assigning zero weight to the elements of the sequence from some point on. On the other hand, the use of general (infinite) neighbourhood sequences provides a more flexible and exact tool in building up the theoretical background. Obviously, the restriction of the concepts to periodic neighbourhood sequences, yields a notion of velocity and metric defined for them. We note that our results are new already in the periodic case.

In this paper we deal with neighbourhood sequences defined on $\mathbb{Z}^n$. However, there can be applications, where the grid points form another kind of structure (e.g., triangular or hexagonal). For a survey on planar grids, see [8]. The concept of neighbourhood sequences can be easily generalized to these grids, see [7] for the cases of triangular and hexagonal grids. The investigations and concepts of the present paper can be extended to these structures, too.

The aim of this article is to introduce velocity and metric on the set of neighbourhood sequences. We do this in a way to fit the algebraic structure of such sequences. We also work out a possible application scheme for broadcasting information. The structure of this paper is as follows. In the second section we give our notation, and provide some properties of the concepts introduced. In the third section we clarify which conditions should be met by the notion of velocity. In Section 4 the concept of velocity is introduced, and some important properties of this notion are proved. In Section 5 we give some theoretical examples to illustrate the behaviour of velocity, and in the sixth section we show how the theory can be applied for broadcasting information in a general network model. In Section 7 we define a metric on the set of neighbourhood sequences, and study its properties.

2 Notation and basic concepts

In this section, we recall some definitions and notation from [2] and [4] concerning neighbourhood sequences. In what follows, n denotes a positive integer.

Definition 2.1. *Let p be a point in $\mathbb{Z}^n$. The i-th coordinate of p is indicated by $\mathrm{Pr}_i(p)$ $(1 \le i \le n)$. Let k be an integer with $0 \le k \le n$. The points $p,\ q \in \mathbb{Z}^n$ are called k-neighbours, if the following two conditions hold:*

- $|\,\mathrm{Pr}_i(p) - \mathrm{Pr}_i(q)| \le 1 \quad (1 \le i \le n),$
- $\sum\limits_{i=1}^{n} |\,\mathrm{Pr}_i(p) - \mathrm{Pr}_i(q)| \le k.$

The sequence $A = (a(i))_{i=1}^{\infty}$, where $1 \le a(i) \le n$ for all $i \in \mathbb{N}$, is called an n-dimensional (shortly nD) neighbourhood sequence. A is periodic, if for some $l \in \mathbb{N}$, $a(i+l) = a(i)$ $(i \in \mathbb{N})$. For every $i \in \mathbb{N}$ and $j = 1, \dots, n$ put

$$a_j(i) = \min(a(i), j) \quad \text{and} \quad f_j^A(i) = \sum_{k=1}^{i} a_j(k).$$

The set of the nD-neighbourhood sequences will be denoted by S_n.

Let $p,\ q \in \mathbb{Z}^n$, and $A \in S_n$. The point sequence $p = p_0, p_1, \dots, p_m = q$, where p_{i-1} and p_i are $a(i)$-neighbours in $\mathbb{Z}^n$ $(1 \le i \le m)$, is called an A-path from p to q of length m. The A-distance $d(p,q;A)$ of p and q is defined as the length of the shortest A-path between them.

A natural partial ordering relation on S_n can be introduced in the following way (see [2] and [4]). For $A, B \in S_n$ we define the relation $\sqsupseteq^*$ by

$$A \sqsupseteq^* B \quad \Leftrightarrow \quad d(p,q;A) \le d(p,q;B) \text{ for all } p, q \in \mathbb{Z}^n.$$

In case of $A \sqsupseteq^* B$ we say that A is faster than B. There is a strong connection between this relation and the values $f_j(i)$, shown by the following result from [4].

Theorem 2.2. *Let $A, B \in S_n$. Then*

$$A \sqsupseteq^* B \quad \Leftrightarrow \quad f_j^A(i) \ge f_j^B(i) \ \textit{for every } i \in \mathbb{N} \textit{ and } j = 1, \dots, n.$$

3 Preliminaries to introduce velocity

By defining velocity, we assign a positive real number to every neighbourhood sequence. In this section we give some natural conditions, which should be met by this concept.

(I) *Velocity must be sensitive to the behavior of the sequences in different dimensions.*
It can happen that a sequence spreads "faster" than another one in higher dimensions, but they have the same "speed" in lower dimensional subspaces. For example, in 3D the sequences (3,3,3,3,...) and (2,2,2,2,...) have the same velocity on the planes $\{x,y\}$, $\{x,z\}$, $\{y,z\}$ defined by the coordinate axes; or the sequences (1,3,1,3,...) and (2,2,2,2,...) behave differently in the subspaces of $\mathbb{Z}^3$. These features should be reflected in the definition of velocity.

(II) *The elements of the sequences must be weighted with a suitable weight function.*
There are two reasons to establish this condition. First, it is natural to consider the initial elements of the sequences more important than the elements which occur later. The second reason comes from theoretical necessity. Namely, if we want to take into consideration all elements of the sequences, then we have to guarantee the convergence of certain sums or series of the (weighted) elements of the sequences.

(III) *Velocity must be defined such that it fits the natural ordering.*
This condition is very evident: velocity should preserve the ordering $\sqsupseteq^*$. If a neighbourhood sequence is faster than another one, its velocity should be larger as well. As $\sqsupseteq^*$ is only a partial ordering, the opposite statement cannot be true. However, our velocity concept, introduced in the next section, will have the nice property that in a certain sense this opposite statement also holds (cf. Theorem 4.11).

4 Assigning velocity to neighbourhood sequences

According to (II), we first give the concept of a weight system, which will be appropriate in our further investigations.

Definition 4.1. *Let $n \in \mathbb{N}$. The set of functions $\delta_j : \mathbb{N} \to \mathbb{R}$ $(1 \le j \le n)$ is called a weight system, if the following three conditions hold:*

- $\delta_j(i) \ge 0 \quad (1 \le j \le n,\ i \in \mathbb{N})$,
- $\sum\limits_{i=1}^{\infty} \delta_j(i) < \infty \quad (1 \le j \le n)$,
- δ_j *is monotone decreasing* $(1 \le j \le n)$.

In order to meet (I), we introduce the concept of velocity in two steps. First, we assign an n-tuple to every neighbourhood sequence. The elements of this n-tuple reflect the "velocity" of the given neighbourhood sequence in the subspaces of $\mathbb{Z}^n$ of dimensions from 1 to n. Then, we define one descriptive velocity value.

Definition 4.2. *Let $A \in S_n$, and δ_j $(1 \leq j \leq n)$ be a weight system. The j-dimensional velocity of A is defined as*

$$v_j^A = \sum_{i=1}^{\infty} a_j(i)\delta_j(i).$$

Remark 4.3. *Let T be the linear space of the bounded real sequences over $\mathbb{R}$, and let δ_j $(1 \leq j \leq n)$ be a weight system. It is well-known (see e.g., [5]) that for every j, if $\delta_j(i) > 0$ for all $i \in \mathbb{N}$ then with the norm*

$$||(x_i)_{i=1}^{\infty}|| = \sum_{i=1}^{\infty} |x_i|\delta_j(i) \qquad ((x_i)_{i=1}^{\infty} \in T),$$

T becomes a Banach space. Thus, for any $A = (a(i))_{i=1}^{\infty}$, v_j^A could be defined as $v_j^A = ||(a_j(i))_{i=1}^{\infty}||$.

Remark 4.4. *For every $A \in S_n$ we have*

$$\sum_{i=1}^{\infty} \delta_j(i) \leq v_j^A \leq n \sum_{i=1}^{\infty} \delta_j(i).$$

We define the velocity of A by the help of the j-dimensional velocities.

Definition 4.5. *Let $A \in S_n$. The velocity of A is given by*

$$v^A = \frac{1}{n} \sum_{j=1}^{n} v_j^A.$$

Remark 4.6. *By the definition of v_j^A $(j = 1, \ldots, n)$ and v^A, we have that for every $\varepsilon > 0$ there exists some $k_0 \in \mathbb{N}$ such that for any $k > k_0$*

$$v_j^A - \sum_{i=1}^{k} a_j(i)\delta_j(i) < \varepsilon, \quad (j = 1, \ldots, n),$$

and also

$$v^A - \frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{k} a_j(i)\delta_j(i) < \varepsilon.$$

This shows that regardless of the system δ_j, the j-dimensional velocities and the velocity of A is "determined" by the first "few" terms of A. In particular, if we take $\delta_j(i) = 0$ for $1 \leq j \leq n$ with some $i \in \mathbb{N}$, then only the first $i - 1$ terms of A contribute to v^A. This consideration can be important in certain practical applications.

In the next section we analyze the behavior of the velocity with respect to various weight systems. Now, we show how conditions (I), (II) and (III) are met by this velocity concept.

The velocity vector $(v_1^A, v_2^A, \ldots, v_n^A)$, thus also v^A is obviously sensitive to the behavior of the sequence A in subspaces of $\mathbb{Z}^n$ of dimensions from 1 to n. Thus, (I) is satisfied. As we use a weight system to define $(v_1^A, v_2^A, \ldots, v_n^A)$ and v^A, the requirements of (II) are also met. The following theorem verifies that our velocity concept satisfies condition (III), too.

Theorem 4.7. *Let $A, B \in S_n$ with $A \sqsupseteq^* B$, and let δ_j $(j = 1, \ldots, n)$ be a weight system. Then, $v_j^A \geq v_j^B$ for every $j = 1, \ldots, n$.*

Proof. Put $A = (a(i))_{i=1}^{\infty}$ and $B = (b(i))_{i=1}^{\infty}$, and fix some j with $1 \leq j \leq n$. Let $k \in \mathbb{N}$ be arbitrary. Since δ_j is monotone decreasing, we can write

$$\delta_j(k-1) = \delta_j(k) + \varepsilon_j(k-1),$$

$$\delta_j(k-2) = \delta_j(k) + \varepsilon_j(k-1) + \varepsilon_j(k-2),$$

$$\vdots$$

$$\delta_j(1) = \delta_j(k) + \varepsilon_j(k-1) + \varepsilon_j(k-2) + \cdots + \varepsilon_j(1),$$

with $\varepsilon_j(m) \geq 0$ $(m = 1, \ldots, k-1)$. Put $\varepsilon_j(k) = \delta_j(k)$. Using these relations, by a simple calculation we get

$$\sum_{i=1}^{k} a_j(i)\delta_j(i) - \sum_{i=1}^{k} b_j(i)\delta_j(i) = \sum_{m=1}^{k} \varepsilon_j(m) \left(\sum_{h=1}^{m} a_j(h) - \sum_{h=1}^{m} b_j(h) \right). \qquad (\star)$$

Observe that as $\sum_{h=1}^{m} a_j(h) = f_j^A(m)$ and $\sum_{h=1}^{m} b_j(h) = f_j^B(m)$, by Theorem 2.2 $A \sqsupseteq^* B$ implies

$$\sum_{h=1}^{m} a_j(h) \geq \sum_{h=1}^{m} b_j(h)$$

for every m with $1 \leq m \leq k$. As $\varepsilon_j(m) \geq 0$ $(m = 1, \ldots, k)$, $(\star)$ yields

$$\sum_{i=1}^{k} a_j(i)\delta_j(i) \geq \sum_{i=1}^{k} b_j(i)\delta_j(i).$$

By letting $k \to \infty$, we obtain $v_j^A \geq v_j^B$. □

Remark 4.8. *By the definition of the velocity, the above theorem implies that if $A \sqsupseteq^* B$, then $v^A \geq v^B$.*

In the following two remarks we explain why some alternative ways of introducing velocity would not be appropriate.

Remark 4.9. *The monotonity of δ_j is necessary to have Theorem 4.7. Indeed, let $A, B \in S_2$ be defined by*

$$A = (2,\ 1,\ 1,\ 1,\ 1,\dots) \quad \textit{and} \quad B = (1,\ 2,\ 1,\ 1,\ 1,\dots).$$

Moreover, let δ_1 be arbitrary, and put

$$\delta_2(i) = \begin{cases} \frac{1}{4}, & \textit{if } i = 1, \\ \frac{1}{2^{i-1}}, & \textit{otherwise.} \end{cases}$$

Clearly, $A \sqsupseteq^* B$ holds, but $v_2^A = \frac{6}{4}$ and $v_2^B = \frac{7}{4}$. Thus, $v_2^B \geq v_2^A$, and also $v^B \geq v^A$ in this case.

Remark 4.10. *It would be possible to define v_j^A in a more general way. Namely, for any $m \in \mathbb{R}$ with $m > 0$, we could put*

$$v_{j,m}^A = \left(\sum_{i=1}^{\infty} (a_j(i))^m \delta_j(i) \right)^{\frac{1}{m}}.$$

However, on one hand the case $m < 1$ does not seem to be interesting. On the other hand, in case of $m > 1$ it is easy to find sequences $A, B \in S_n$ and a weight system δ_j $(j = 1, \dots, n)$ such that Theorem 4.7, hence condition (III) fails for them.

As one can easily see, it can happen that with some weight system δ_j, $v_j^A \geq v_j^B$ for every $j = 1, \dots, n$, but $A \sqsupseteq^* B$ does not hold. However, in some sense we can reverse Theorem 4.7, even in case of positive weight systems, i.e. when $\delta_j(i) > 0$ for all $i \in \mathbb{N}$, $j = 1, \dots, n$. More precisely, we have

Theorem 4.11. *Let $A, B \in S_n$. If for any positive weight system δ_j $(1 \leq j \leq n)$, $v_j^A \geq v_j^B$ holds for all $j = 1, \dots, n$, then $A \sqsupseteq^* B$.*

Proof. Let $k \in \mathbb{N}$ be arbitrary, and for every j with $1 \leq j \leq n$ set

$$\delta_j^{(k)}(i) = \begin{cases} 1, & \text{if } i \leq k, \\ \frac{1}{2^{i-k}n}, & \text{otherwise.} \end{cases}$$

Clearly, the system $\delta_j^{(k)}$ $(j = 1, \dots, n)$ is a weight system. Thus, by our assumptions we have

$$\sum_{i=1}^{\infty} a_j(i)\delta_j^{(k)}(i) = v_j^A \geq v_j^B = \sum_{i=1}^{\infty} b_j(i)\delta_j^{(k)}(i).$$

Hence, for every $j = 1, \dots, n$

$$\sum_{i=1}^{k} a_j(k) + \sum_{h=k+1}^{\infty} \frac{a_j(h)}{2^{h-k}n} \geq \sum_{i=1}^{k} b_j(k) + \sum_{h=k+1}^{\infty} \frac{b_j(h)}{2^{h-k}n}$$

holds. Replacing $\sum_{i=1}^{k} a_j(k)$ and $\sum_{i=1}^{k} b_j(k)$ by $f_j^A(k)$ and $f_j^B(k)$, respectively, we get

$$f_j^A(k) - f_j^B(k) \geq \sum_{h=k+1}^{\infty} \frac{b_j(h)}{2^{h-k}n} - \sum_{h=k+1}^{\infty} \frac{a_j(h)}{2^{h-k}n} \geq \frac{1}{n} - 1.$$

Since $f_j^A(k) - f_j^B(k)$ is an integer, we may infer that

$$f_j^A(k) - f_j^B(k) \geq 0.$$

By Theorem 2.2 the proof is complete. □

Remark 4.12. *It can be easily verified that the condition $v_j^A \geq v_j^B$ for all $j = 1, \ldots, n$ cannot be replaced by $v^A \geq v^B$.*

5 Examples of weight systems

In this section we give examples of weight systems, and analyze the behavior of the velocity concept. We investigate exponentially decreasing systems, and calculate the velocity of some concrete sequences with respect to different weight systems.

Let $c > 1$, and put

$$\delta_j(i) = \frac{1}{c^{i-1}} \text{ for every } j = 1, \ldots, n \text{ and } i \in \mathbb{N}.$$

Obviously, δ_j is a weight system with

$$\sum_{i=1}^{\infty} \delta_j(i) = \frac{c}{c-1} \quad (j = 1, \ldots, n).$$

Consider the nD-neighbourhood sequences

$$A = (h,\ 1,\ 1,\ 1,\ 1,\ \ldots) \text{ and } B = (1,\ n,\ n,\ n,\ n,\ \ldots), \text{ where } 2 \leq h \leq n.$$

Then

$$v^A = v_j^A = \frac{1}{c-1} + h \text{ and } v^B = v_j^B = \frac{n}{c-1} + 1 \quad (j = 1, \ldots, n).$$

Clearly, the sequences A and B cannot be compared by the ordering $\sqsupseteq^*$. We show how the relation between the velocity values of A and B change according to the choice of the parameter c.

First, suppose that $c > n$. Then, we have

$$v^A = v_j^A = \frac{1}{c-1} + h \geq \frac{1}{c-1} + 2 = \frac{c}{c-1} + 1 > \frac{n}{c-1} + 1 = v_j^B = v^B.$$

In general, using this weight system we obtain a very strong condition, namely that $v^A > v^B$ if and only if A precedes B lexicographically.

Now, let $c = 2$. In this case we have

$$v^A = v_j^A = 1 + h \leq 1 + n = v_j^B = v^B,$$

with equality only for $h = n$.

Finally, set $c < 2$. Now, by a simple calculation, we get $v^B = v_j^B > v_j^A = v^A$.

Summarizing, using such a weight system, we can get rid of the (sometimes excessive) importance of the first "few" elements of a neighbourhood sequence. Especially, for every $k \in \mathbb{N}$, by choosing a suitable c, we can have $v_j^B > v_j^A$ for the nD-sequences

$$A = (\underbrace{n\ ,n\ ,\dots\ ,n}_{k},\ 1,\ 1,\ 1,\ \dots) \text{ and } B = (\underbrace{1\ ,1\ ,\dots\ ,1}_{k},\ n,\ n,\ n,\ \dots).$$

On the other hand, by the appropriate choice of c we can give large significance to the first "few" elements of the neighbourhood sequences, ignoring their later elements.

By choosing other (e.g., polinomially decreasing) types of weight systems we can have different properties. The weight system should be chosen appropriately for the actual application, as we can see from a practical example given in the following section.

6 An application for broadcasting information

In this section we give an application scheme of neighbourhood sequences and velocity in a network model, where the members of the network are the points of $\mathbb{Z}^2$. As we mentioned in the introduction, neighbourhood sequences and velocity can be introduced also for other types of grids. Hence, this application scheme could be used in such cases, too.

Our network model can be considered as a variant of the Manhattan Street Network (MSN) introduced by Maxemchuk in [6]. In our scheme, the clients are connected with horizontal/vertical and diagonal edges (see Figure 1). There is an information source at the center (origin) which broadcasts information to the other members of the network. The system is based on priority, that is if a client is "closer" to the origin than another one, it has greater priority, and receives the information earlier. We can think of subscription systems for instance, where clients pay different fees according to their positions with respect to the information source.

It is worth indexing the clients by their "reachability" from the origin. For this purpose, if a client sits on the point $(x, y) \in \mathbb{Z}^2$, then its index will be given by the first few (significant) elements of the slowest neighbourhood sequence A, for which $d((x,y),(0,0);A)$ is minimal. The clients with the same index have equal priority, so they should pay the same fee (especially, clients indexed by "1" have the greatest priority).

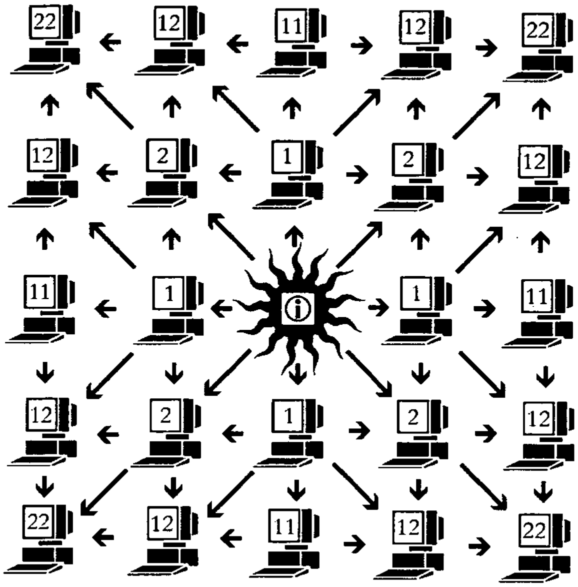

Figure 1: 2D priority-based model for broadcasting information

In this model, we use 2D-neighbourhood sequences to deliver the information to the clients. Suppose that the cost of broadcasting information decreases with the number of 2-s in the chosen neighbourhood sequence. The most expensive sequence is $(2, 2, \dots)$, while the cheapest one is $(1, 1, \dots)$. Knowing the importance of the information, we have to choose one of the cheapest sequences, which is still "fast" enough. That is, we take a neighbourhood sequence, whose velocity fits the importance of the information to be sent. According to the size of the network, it is sufficient to consider the first "few" terms of the sequences (i.e. to work with a weight system in which $\delta_j(N) = 0$ for $j = 1, \dots, n$ with an appropriate $N \in \mathbb{N}$).

By choosing different weight systems, we can increase or decrease the initial priority of the clients in the network. If we do not take much care of the clients residing far from the source, we need to choose a weight system, which decreases rapidly. In the opposite case we can take a very slowly decreasing weight system.

This network model can be easily extended to higher dimensions. In this case, we can take more advantage of the behavior of neighbourhood sequences in lower dimensional subspaces. For example, in $\mathbb{Z}^3$, if we know in advance that a special type of information is important only for a group of clients, we can place them onto or close to the (x, y), (y, z) and (x, z) planes. Thus, for the distribution of this special kind of information we can choose quite a cheap neighbourhood sequence, which consists of mainly 1 and 2 values. To have a similar possibility in 2D, we have to put such clients near the coordinate axes, and use sequences containing mostly 1-s. For a known higher dimensional generalization of MSN, see [3].

Finally, we note that our application scheme may be used at any other areas which are based on regular network topologies. Such areas are e.g., chip architecture design, or parallel processing.

7 Metric space of the neighbourhood sequences

We introduce a metric on the set of neighbourhood sequences in a similar fashion as we did it for velocity. From this point on, we assume that for every weight system $\Delta = \{\delta_j \mid j = 1, \ldots, n\}$ we have $\delta_n(i) > 0$ $(i \in \mathbb{N})$, unless we state the contrary.

Definition 7.1. *Let $\Delta = \{\delta_j \mid j = 1, \ldots, n\}$ be a weight system and $A, B \in S_n$. The distance ϱ_Δ of these sequences is defined by*

$$\varrho_\Delta(A, B) = \frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{\infty} |a_j(i) - b_j(i)| \; \delta_j(i).$$

Remark 7.2. *One can easily verify that in case of any weight system Δ, the function ϱ_Δ is a metric on S_n. At this point we make use of our assumption that $\delta_n(i) > 0$ for every $i \in \mathbb{N}$.*

Remark 7.3. *The metric space (S_n, ϱ_Δ) is bounded. Its diameter is*

$$diam(S_n, \varrho_\Delta) = \varrho_\Delta(\,(1,\ 1,\ \ldots),\ (n,\ n,\ \ldots)\,) = \frac{n-1}{n} \sum_{j=1}^{n} \sum_{i=1}^{\infty} \delta_j(i).$$

Remark 7.4. *In practical applications usually only "finite" sequences are needed. Thus, it may be useful to consider those sequences identical which agree in their first "few" elements. This can be done in the following way. Let $N \in \mathbb{N}$, and $\Delta = \{\delta_j \mid j = 1, \ldots, n\}$ be a weight system such that $\delta_j(N+1) = 0$ for all j (including $j = n$), but $\delta_n(N) > 0$. Consider two neighbourhood sequences equivalent if and only if their first N elements coincide. Then Definition 7.1 provides a metric on the classes of S_n induced by this equivalence relation.*

In what follows, we establish some useful and interesting properties of the metric spaces (S_n, ϱ_Δ).

Theorem 7.5. *For any weight system Δ, (S_n, ϱ_Δ) is a complete metric space.*

Proof. Let Δ be any weight system. We prove the theorem by showing that every Cauchy sequence in S_n has a limit. We actually construct this limit sequence in the proof. Let $(A_k)_{k=1}^{\infty}$ be a Cauchy sequence in (S_n, ϱ_Δ), and let $m \in \mathbb{N}$. By the Cauchy-property of $(A_k)_{k=1}^{\infty}$, there exists some $k_0 \in \mathbb{N}$ such that for every $k_1, k_2 > k_0$, $\varrho_\Delta(A_{k_1}, A_{k_2}) < \delta_n(m)$. Hence the first m elements of the neighbourhood sequences A_{k_1} and A_{k_2} are identical. Define the sequence A in the following way. For every $m \in \mathbb{N}$ choose a $k_0 \in \mathbb{N}$, such that the m-th elements of A_{k_1} and A_{k_2} with $k_1, k_2 \geq k_0$ are equal. Let $a(m)$ be this element, and put $A = (a(m))_{m=1}^{\infty}$. Clearly, A is well defined. By the construction of A we immediately get that $\lim_{k \to \infty} A_k = A$. □

A sequence $(A_k)_{k=1}^{\infty}$ is monotone increasing (resp. decreasing), if $A_{i+1} \sqsupseteq^* A_i$ (resp. $A_i \sqsupseteq^* A_{i+1}$) holds for every $i \in \mathbb{N}$.

Theorem 7.6. *Every monotone increasing or decreasing sequence $(A_k)_{k=1}^{\infty}$, with $A_k \in S_n$ $(k \in \mathbb{N})$ is convergent.*

Proof. As in the previous proof, we construct the limit of $(A_k)_{k=1}^{\infty}$. We may assume that $(A_k)_{k=1}^{\infty}$ is monotone increasing, the proof in the other case is similar.

Put $A_k = (a^{(k)}(i))_{i=1}^{\infty}$. As $(A_k)_{k=1}^{\infty}$ is increasing, so is $(a^{(k)}(1))_{k=1}^{\infty}$. As $n \geq a^{(k)}(1)$ $(k \in \mathbb{N})$, there exists some $k_0 \in \mathbb{N}$ such that for any $k_1, k_2 \geq k_0$ we have $a^{(k_1)}(1) = a^{(k_2)}(1)$. Put $a(1) = a^{(k_0)}(1)$. Suppose that $a(i)$ is already given for $i \leq m$, and define $a(m+1)$ in the following way. Choose $t_0 \in \mathbb{N}$ such that for $t_1, t_2 \geq t_0$ and $1 \leq i \leq m$, $a^{(t_1)}(i) = a^{(t_2)}(i)$ holds. Since $(A_k)_{k=t_0}^{\infty}$ is increasing, so is the sequence $(a^{(k)}(m+1))_{k=t_0}^{\infty}$. As $n \geq a^{(k)}(m+1)$ for every $k \in \mathbb{N}$, there exists some $s_0 \in \mathbb{N}$ such that for any $s_1, s_2 \geq s_0$ we have $a^{(s_1)}(m+1) = a^{(s_2)}(m+1)$. Put $a(m+1) = a^{(s_0)}(m+1)$.

From the construction of $A = (a(m))_{m=1}^{\infty}$ it is clear that for every $m \in \mathbb{N}$ there exists some $k_0 \in \mathbb{N}$, such that if $k \geq k_0$, the first m elements of A_k and A coincide. Thus, $\lim\limits_{k\to\infty} A_k = A$, and the theorem follows. □

The next result shows that the Bolzano-Weierstrass theorem is true in the constructed metric spaces.

Proposition 7.7. *For any weight system Δ, every subset of (S_n, ϱ_Δ) of infinite cardinality has an accumulation point.*

Proof. Let H be an infinite subset of S_n. We construct an accumulation point of H. Let $a(1)$ be a number which is the first element of infinitely many sequences in H. Suppose that $a(i)$ with $i \leq m$ is already defined. Let $a(m+1)$ be a number which is the $(m+1)$-th element of infinitely many such sequences in H, whose first m elements are $a(1), a(2), \ldots, a(m)$. Put $A = (a(m))_{m=1}^{\infty}$. Clearly, A is an accumulation point of H. □

Periodic neighbourhood sequences can play important role in certain applications. Our last result shows that they form a dense subset of (S_n, ϱ_Δ). As the set of periodic neighbourhood sequences is countable, this also yields that (S_n, ϱ_Δ) is a separable metric space.

Theorem 7.8. *For any weight system Δ, the set of periodic neighbourhood sequences is dense in (S_n, ϱ_Δ).*

Proof. Let $A \in S_n$ and $\varepsilon > 0$. By the definition of ϱ_Δ there exists some $k_0 \in \mathbb{N}$, such that if the first k_0 elements of $B \in S_n$ is the same as those of A, then $\varrho_\Delta(A, B) < \varepsilon$ holds. So put $b(i) = a(i \bmod k_0)$ $(i \in \mathbb{N})$, and $B = (b(i))_{i=1}^{\infty}$. Clearly, B is periodic and $\varrho_\Delta(A, B) < \varepsilon$, thus the proof is complete. □

8 Conclusion

In this paper, we introduce velocity and metric for the set of neighbourhood sequences. We show that these notions fit well the structure of such sequences. By their help we can compare neighbourhood sequences more precisely, than using only the natural partial ordering relation. We also work out a possible application scheme for broadcasting information.

Acknowledgements

The authors are grateful to Professor Mátyás Arató for his valuable remarks, and to the referee for his useful comments.

References

[1] Das, P.P.: Lattice of octagonal distances in digital geometry, *Pattern Recognition Lett.* **11** (1990), 663-667.

[2] Das, P.P., Chakrabarti, P.P. and Chatterji, B.N.: Distance functions in digital geometry, *Inform. Sci.* **42** (1987), 113-136.

[3] Dally, W.J.: Performance analysis of k-ary n-cube interconnection networks, *IEEE Transaction on Computers* **39** (1990), 775-785.

[4] Fazekas, A., Hajdu, A. and Hajdu, L.: Lattice of generalized neighbourhood sequences in nD and ∞D, *Publ. Math. Debrecen* **60** (2002), 405-427.

[5] Hill, E.: Methods in classical and functional analysis, Addison-Wesley Pub. Co., Reading Mass., 1972.

[6] Maxemchuk, N.F.: The Manhattan street network, in *Proc. GLOBECOM'85*, New Orleans, LA, Dec. 1986, 255-261.

[7] Nagy, B.: Finding shortest path with neighbourhood sequences on triangular grids, *ITI - ISPA 2001*, Pula, Croatia (2001), 55-60.

[8] Radványi, A.: On the rectangular grid representation of general CNN networks, *Int. J. Circ. Theor. Appl.* **30** (2002), 181-193.

[9] Rosenfeld, A. and Pfaltz, J.L.: Distance functions on digital pictures, *Pattern Recognition* **1** (1968), 33-61.

Received April, 2002

Acta Cybernetica **16** (2003) 147–154.

Modelling a Sender-Receiver System

Cristian Vidraşcu [*]

Abstract

In this paper we present a sender-receiver system with an unlimited buffer modelled by a jumping Petri net, and then we prove some properties of the system.

Keywords: parallel/distributed systems, Petri nets, jumping Petri nets, modelling, verification.

1 Introduction and Preliminaries

A Petri net is a mathematical model used for the specification and the analysis of parallel/distributed systems. An introduction about Petri nets can be found in [4].

One formal analysis method for Petri nets is that of place and transition invariants, which were first introduced in [3]. Place and transition invariants are useful to prove dynamic properties, like reachability, boundedness, home state, liveness and fairness properties.

It is well-known that the behaviour of some distributed systems cannot be adequately modelled by classical Petri nets. Many extensions which increase the computational and expressive power of Petri nets have been thus introduced. One direction has led to various modifications of the firing rule of nets. One of these extensions is that of jumping Petri net, introduced in [5].

Let us briefly recall the basic notions and notations concerning Petri nets and jumping Petri nets in order to give the reader the necessary prerequisites for the understanding of this paper (for details the reader is referred to [1], [4], [2]). Mainly, we will follow [2], [5].

A *Place/Transition net*, shortly *Petri net*, (finite, with infinite capacities), is a 4-tuple $\Sigma = (S, T, F, W)$, where S and T are two finite non-empty sets (of *places* and *transitions*, resp.), with $S \cap T = \emptyset$, $F \subseteq (S \times T) \cup (T \times S)$ is the *flow relation* and $W : (S \times T) \cup (T \times S) \to \mathbb{N}$ is the *weight function* of Σ verifying $W(x, y) = 0$ iff $(x, y) \notin F$.

A *marking* of a Petri net Σ is a function $M : S \to \mathbb{N}$; it will be sometimes identified with a $|S|$-dimensional vector. The operations and relations on vectors are componentwise defined. $\mathbb{N}^S$ denotes the set of all markings of Σ.

[*]Faculty of Computer Science, "Al. I. Cuza" University of Iaşi, România. E-mail: vidrascu@infoiasi.ro

A *marked Petri net* is a pair $\gamma = (\Sigma, M_0)$, where Σ is a Petri net and M_0, called the *initial marking* of γ, is a marking of Σ.

Let Σ be a Petri net, $t \in T$ and $w \in T^*$. The functions $t^-, t^+ : S \to \mathbb{N}$ and $\Delta t, \Delta w : S \to \mathbb{Z}$ are defined by: $t^-(s) = W(s,t)$, $t^+(s) = W(t,s)$, $\Delta t(s) = t^+(s) - t^-(s)$, and

$$\Delta w(s) = \begin{cases} 0, & \text{if } w = \lambda \\ \sum_{i=1}^{n} \Delta t_i(s), & \text{if } w = t_1 t_2 \ldots t_n \ (n \geq 1) \end{cases}, \text{ for all } s \in S.$$

The sequential behaviour of a Petri net Σ is given by the so-called *firing rule*, which consists of

- the *enabling rule*: a transition t is *enabled* at a marking M in Σ (or t is *fireable* from M), abbreviated $M[t\rangle_\Sigma$, iff $t^- \leq M$;
- the *computing rule*: if $M[t\rangle_\Sigma$, then t may *occur* yielding a new marking M', abbreviated $M[t\rangle_\Sigma M'$, defined by $M' = M + \Delta t$.

In fact, any transition t of Σ establishes a binary relation on $\mathbb{N}^S$, denoted by $[t\rangle_\Sigma$ and given by: $M[t\rangle_\Sigma M'$ iff $t^- \leq M$ and $M' = M + \Delta t$.

If $t_1, t_2, \ldots, t_n$ $(n \geq 1)$ are transitions of Σ, $[t_1 t_2 \ldots t_n\rangle_\Sigma$ will denote the classical product of the relations $[t_1\rangle_\Sigma, \ldots, [t_n\rangle_\Sigma$. Moreover, the relation $[\lambda\rangle_\Sigma$ is considered, by defining $[\lambda\rangle_\Sigma = \{(M, M) | M \in \mathbb{N}^S\}$.

Let $\gamma = (\Sigma, M_0)$ be a marked Petri net, and $M \in \mathbb{N}^S$. The word $w \in T^*$ is called a *transition sequence* from M in Σ if there exists a marking M' of Σ such that $M[w\rangle_\Sigma M'$. Moreover, the marking M' is called *reachable* from M in Σ. The set of all reachable markings from M_0 is called *the reachability set* of γ, and it is denoted by $[M_0\rangle_\gamma$.

A place $s \in S$ is *bounded* if there exists $k \in \mathbb{N}$ such that $M(s) \leq k$, for all $M \in [M_0\rangle_\gamma$. The net γ is *bounded* if all its places are bounded.

A transition $t \in T$ is *live* if for any reachable marking $M \in [M_0\rangle_\gamma$, there exists a marking M' reachable from M such that t is fireable from M'. The net γ is *live* if all its transitions are live.

In order to be able to define the notion of the incidence matrix for a Petri net $\Sigma = (S, T, F, W)$, it is necessary to have a total ordering of the sets S and T. Without loss of generality, it will be assumed that, if these sets are of the form

$$S = \{s_1, \ldots, s_m\}, \text{ and } T = \{t_1, \ldots, t_n\},$$

then they are totally ordered by the natural order on the indexes of the elements:

$$S: \ s_1 < \ldots < s_m, \text{ and } T: \ t_1 < \ldots < t_n.$$

The *incidence matrix* of a Petri net Σ is the $m \times n$-dimensional matrix I_Σ defined by

$$I_\Sigma(i,j) = \Delta t_j(s_i), \ \forall\ 1 \leq i \leq m, \ \forall\ 1 \leq j \leq n.$$

The notion of incidence matrix is extended also to marked Petri nets (Σ, M_0) through the unmarked underlying net Σ.

An *S-invariant* (or *place invariant*) of Σ is any m-dimensional vector J of integer numbers which satisfies the equation $J \cdot I_\Sigma = \mathbf{0}$.

The characterization theorem of S-invariants says that, if J is an S-invariant of a marked Petri net $\gamma = (\Sigma, M_0)$, then the relation

$$J \cdot M = J \cdot M_0$$

holds for any $M \in [M_0\rangle_\gamma$. In other words, this theorem says that any S-invariant of γ gives the weights for the places of a subnet of γ in which the tokens are preserved (through these weights).

Jumping Petri nets ([5]) are an extension of Petri nets, which allows them to do "spontaneous jumps" from one marking to another one (this is similar to λ-moves in automata theory).

A *jumping Petri net* is a pair $\gamma = (\Sigma, R)$, where Σ is a Petri net and R is a binary relation on the set of markings of Σ (i.e. $R \subseteq \mathbb{N}^S \times \mathbb{N}^S$), called the *set of (spontaneous) jumps* of γ.

Let $\gamma = (\Sigma, R)$ be a jumping Petri net. The pairs $(M, M') \in R$ are referred to as *jumps* of γ. Σ is called the *underlying Petri net* of γ. A *marking* of γ is any marking of its underlying Petri net. If γ has finitely many jumps (i.e. R is finite), then γ is called a *finite jumping Petri net.*

For any jump $r = (M, M') \in R$, the function $\Delta r : S \to \mathbb{Z}$ is defined by $\Delta r(s) = M'(s) - M(s)$, for all $s \in S$. If the set of jumps R has finitely many variations (i.e. the set $\Delta R = \{\Delta r \mid r \in R\}$ is finite), then γ is called a *Δ-finite jumping Petri net.*

A *marked jumping Petri net* is defined similarly as a marked Petri net, by changing "Σ" into "Σ, R".

Pictorially, a jumping Petri net will be represented as a classical net and, moreover, the relation R will be separately listed.

The behaviour of a jumping Petri net γ is given by the *j-firing rule*, which consists of

- the *j-enabling rule*: a transition t is *j-enabled* at a marking M (in γ), abbreviated $M[t\rangle_{\gamma,j}$, iff there exists a marking M_1 such that $MR^*M_1[t\rangle_\Sigma$ (R^* being the reflexive and transitive closure of R);
- the *j-computing rule*: if $M[t\rangle_{\gamma,j}$, then the marking M' is *j-produced* by occurring t at M, abbreviated $M[t\rangle_{\gamma,j}M'$, iff there exists two markings M_1, M_2 such that $MR^*M_1[t\rangle_\Sigma M_2R^*M'$.

The notions of *transition j-sequence* and *j-reachable marking* are defined similarly as for Petri nets (the relation $[\lambda\rangle_{\gamma,j}$ is defined by $[\lambda\rangle_{\gamma,j} = R^*$). The *set of all j-reachable markings* of a marked jumping Petri net γ is denoted by $[M_0\rangle_{\gamma,j}$ (M_0 being the initial marking of γ).

All other notions from Petri nets (i.e. boundedness, liveness, etc.) are defined for jumping Petri nets similarly as for Petri nets, by considering the notion of *j-reachability* instead of *reachability* from Petri nets.

Some jumps of a marked jumping Petri net may be never used. Thus a marked jumping Petri net $\gamma = (\Sigma, R, M_0)$ is called *R-reduced* iff, for any jump $(M, M') \in R$, $M \neq M'$ and $M \in [M_0\rangle_{\gamma,j}$.

The notion of place invariants for Δ-finite jumping Petri nets, and results regarding them, can be found in [6]. We will briefly present this notion.

As in the case of Petri nets, in order to be able to define the notion of the incidence matrix for a Δ-finite jumping Petri net $\gamma = (\Sigma, R)$, where $\Sigma = (S, T, F, W)$ is the underlying Petri net of γ, it is necessary to have a total ordering of the sets S, T and ΔR. Without loss of generality, it will be assumed that, if these sets are of the form

$$S = \{s_1, \ldots, s_m\},\ T = \{t_1, \ldots, t_n\},\ \text{ and }\ \Delta R = \{\Delta r_1, \ldots, \Delta r_p\},$$

then they are totally ordered by the natural order on the indexes of the elements:

$$S:\ s_1 < \ldots < s_m,\ T:\ t_1 < \ldots < t_n,\ \text{ and }\ \Delta R:\ \Delta r_1 < \ldots < \Delta r_p.$$

The *incidence matrix* of a Δ-finite jumping Petri net $\gamma = (\Sigma, R)$ is the $m \times (n + p)$-dimensional matrix I_γ defined by

$$I_\gamma(i,j) = \begin{cases} I_\Sigma(i,j) & ,\forall\ 1 \leq j \leq n \\ I_R(i, j-n) & ,\forall\ n+1 \leq j \leq n+p \end{cases} ,\forall\ 1 \leq i \leq m,$$

where I_Σ is the incidence matrix of the underlying Petri net of γ and I_R is the $p \times n$-dimensional matrix given by

$$I_R(i,j) = \Delta r_j(s_i),\ \forall\ 1 \leq i \leq m,\ \forall\ 1 \leq j \leq p\ .$$

The notion of incidence matrix is extended also to marked Δ-finite jumping Petri nets (Σ, R, M_0) through the unmarked underlying net (Σ, R).

An *S-invariant* (or *place invariant*) of γ is any m- dimensional vector J of integer numbers which satisfies the equation $J \cdot I_\gamma = \mathbf{0}$. The S-invariant $J > \mathbf{0}$ is called *minimal* if there exists no S-invariant J' such that $\mathbf{0} < J' < J$.

The characterization theorem of S-invariants ([6]) says that, if J is an S-invariant of a marked Δ-finite jumping Petri net $\gamma = (\Sigma, R, M_0)$, then the relation

$$J \cdot M = J \cdot M_0$$

holds for any $M \in [M_0\rangle_{\gamma,j}$. As in the case of Petri nets, the meaning of this theorem is that any S-invariant of γ gives the weights for the places of a subnet of γ in which the tokens are preserved (through these weights).

The paper is organized as follows. Section 2 presents an example of a sender-receiver system modelled by a jumping Petri net, and section 3 presents the verification of the system properties using the place invariant method. Section 4 concludes this paper.

2 Sender-receiver with unlimited buffer

This section presents an example of using jumping Petri nets to model and analyse real systems.

Let us consider a system consisting of a sender (producer) and a receiver (consumer). The sender produces and sends messages to the receiver, one by one, through an asynchoronous channel (a buffer with unlimited capacity for storing messages). The receiver receives and consumes, one by one, the messages from channel. Moreover, the sender can take a break at any moment, but we impose the restriction that the receiver can enter his inactive state only if the sender is inactive and there is no message pending in the channel.

The same system, but with a limited buffer, was modelled by a Petri net in [4]. Unfortunately, this system with an unlimited buffer cannot be modelled by a Petri net because zero tests of a location with infinite capacity cannot be simulated by Petri nets (a proof of this fact can be found in [2], where a similar system with an unlimited buffer is modelled by a Petri net with inhibitor arcs).

A modelling of this system by a finite jumping Petri net $\gamma = (\Sigma, R, M_0)$ is presented in Figure 1, with the following interpretation of places:

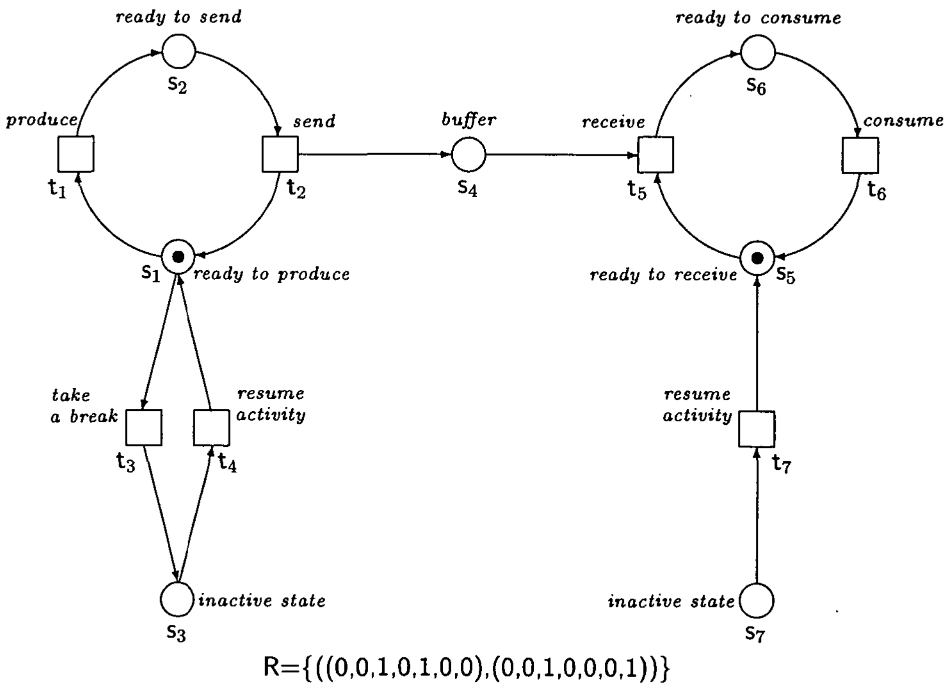

Figure 1: Sender-receiver system with unlimited buffer

- s_1 marked = the sender is ready to produce a message or to take a break;
- s_2 marked = the sender is ready to send the last produced message;
- s_3 marked = the sender is inactive (in a break);
- s_4 = the unlimited buffer for storing messages;
- s_5 marked = the receiver is ready to receive a message or to take a break;
- s_6 marked = the receiver is ready to consume the last received message;
- s_7 marked = the receiver is inactive (in a break).

The interpretations of transition firings are the following:

- t_1 = the sender produces a message;
- t_2 = the sender sends a message;
- t_3 = the sender becomes inactive (takes a break);
- t_4 = the sender resumes his activity;
- t_5 = the receiver receives a message;
- t_6 = the receiver consumes a message;
- t_7 = the receiver resumes his activity.

The entering of the receiver in his inactive state, possible only when the sender is inactive and there are no messages in the buffer, is modelled by the jump of this net, which occurs from the marking $M' = (0,0,1,0,1,0,0)$ to the marking $M''=(0,0,1,0,0,0,1)$.

We say that the sender-receiver system with an unlimited buffer is *modelled correctly*, if it has the following properties:

(**P$_1$**) At any moment, the sender is in one of the states "ready to produce", "ready to send" or "inactive";

(**P$_2$**) At any moment, the receiver is in one of the states "ready to receive", "ready to consume" or "inactive";

(**P$_3$**) The buffer can contain any number of messages;

(**P$_4$**) The receiver can enter his inactive state only if the sender is in his inactive state and there are no messages in the buffer;

(**P$_5$**) The system is live, i.e. it will never reach a deadlock state.

In the next section we will show how the verification of these properties can be done.

3 Verification of system properties

Using S-invariants, we prove in this section the correctness of our modelling.

Theorem 3.1. *The jumping Petri net from Figure 1 models correctly the sender-receiver system with unlimited buffer.*

Proof. Let $\gamma = (\Sigma, R, M_0)$ be the finite jumping Petri net from Figure 1. It is easy to verify that the vectors

$$J_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad J_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix},$$

are S-invariants. Moreover, these are the only minimal S-invariants of γ.

Let $M \in [M_0\rangle_{\gamma,j}$ be an arbitrary j-reachable marking of γ. Using the S-invariant J_1 and the characterization theorem of S-invariants, we find that

$$(*) \quad M(s_1) + M(s_2) + M(s_3) = 1,$$

which proves $(\mathbf{P_1})$. Similarly, using J_2 we obtain that

$$(**) \quad M(s_5) + M(s_6) + M(s_7) = 1,$$

which proves $(\mathbf{P_2})$.

In order to prove $(\mathbf{P_3})$, let us notice the following fact. Given any $k \in \mathbb{N}$, by firing the transition sequence $w = (t_1 t_2)^k$ at the marking M_0, a new marking $M \in [M_0\rangle_{\gamma,j}$ will be produced, with $M(s_4) = k$ and $M(s) = M_0(s)$ for all other places of the net. This means that the buffer can contain any number of messages.

In order to prove $(\mathbf{P_4})$, let us notice that, if M is an arbitrary j-reachable marking in which the receiver is inactive (i.e. $M(s_7) = 1$), then M can be reached only by the occurence of the jump of the net γ (because $M \notin [M_0\rangle_\Sigma$, i.e. the marking M is not reachable in the underlying Petri net of γ). It is obviously that the jump of γ can occur only if the sender is inactive and the message channel is empty.

For proving the net γ is live, i.e. it never reaches a deadlock state, we will show that at any j-reachable marking $M \in [M_0\rangle_{\gamma,j}$ there exists at least one transition of γ which is fireable at M. Indeed, from the equality $(*)$ follows that either the transitions t_1 and t_3 are fireable at M, if $M(s_1) = 1$, or the transition t_2 is fireable at M, if $M(s_2) = 1$, or the transition t_4 is fireable at M, if $M(s_3) = 1$. Therefore, the net from Figure 1 is live, which proves $(\mathbf{P_5})$.

This concludes the proof of the system properties. □

Let us remark that from the last argument from above follows also that the sender is live (i.e. the net γ w.r.t. the set of transitions $\{t_1, t_2, t_3, t_4\}$ is live).

Moreover, the receiver (i.e. the net γ w.r.t. the set $\{t_5, t_6, t_7\}$) is not live, but "almost live", i.e. it never deadlocks excepting the case when the sender is active and the message channel is empty. Indeed, from the equality $(**)$ follows that the only possible cases are the following ones:

i) the transition t_7 is fireable at M, if $M(s_7) = 1$;

ii) the transition t_6 is fireable at M, if $M(s_6) = 1$;

iii) the transition t_5 is fireable at M, if $M(s_5) = 1$ and $M(s_4) > 0$;

iv) the transition t_7 is j-fireable at M (after occuring first the jump of the net at M), if $M(s_5) = 1$, $M(s_4) = 0$ and $M(s_3) = 1$;

v) the case $M(s_5) = 1$, $M(s_4) = 0$ and $M(s_3) = 0$, i.e. the case in which the sender is active ("ready to produce" or "ready to send") and the message channel is empty, is the only case when the receiver has no directly possible action, but only after an action of the sender (either the producing of a message, or the sending of a message, or the entering of the sender in his inactive state).

4 Conclusion

In this paper we have modelled a sender-receiver system with an unlimited buffer by a finite jumping Petri net, and we have proved the correctness of our modelling by using S-invariants.

References

[1] E. Best, C. Fernandez: *Notations and Terminology on Petri Net Theory*, Arbeitspapiere der GMD 195, 1986.

[2] T. Jucan, F.L. Ţiplea: *Petri Nets. Theory and Practice*, The Romanian Academy Publishing House, Bucharest, 1999.

[3] K. Lautenbach: *Liveness in Petri Nets*, Internal Report GMD-ISF 72-02.1, 1972.

[4] W. Reisig: *Petri Nets. An Introduction*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985.

[5] F.L. Ţiplea, T. Jucan: *Jumping Petri Nets*, Foundations of Computing and Decision Sciences, vol. 19, no.4, 1994, pp. 319-332.

[6] C. Vidraşcu: *S-Invariants for Δ-finite Jumping Petri Nets*, Proc. of the 7^{th} International Symposium on Automatic Control and Computer Science, SACCS 2001, 7 pp.

Received January, 2002

Discovering Associations in Very Large Databases by Approximating*

Shichao Zhang† and Chengqi Zhang‡

Abstract

Mining association rules has posed great challenge to the research community. Despite efforts in designing fast and efficient mining algorithms, it remains a time consuming process for very large databases. In this paper, we adopt a slightly different approach to this problem, which can mine approximate association rules quickly. By considering the database as a set of records that are randomly appended, we can apply the central limit theorem to estimate the size of a random subset of the database, and discover both positive and negative association rules by generating all possible useful itemsets from the random subset. However, because of approximation errors, it is possible for some valid rules to be missed, while other invalid rules may be generated. To deal with this problem, we adopt a two phase approach. First, we discover all promising approximate rules from a random sample of the database. Second, these approximate results are used as heuristic information in an efficient algorithm that requires only one-pass of the database to validate rules that have support and confidence close to the desired support and confidence values. We evaluated the proposed technique, and our experimental results demonstrate that the approach is efficient and promising.

Keywords: Data mining, data processing, approximating rule, assisting knowledge discovery, data analysis.

1 Introduction

One of the main challenges in data mining is to identify association rules for very large databases that comprise millions of transactions and items. Some recent efforts have focused on designing efficient algorithms [2, 4, 7, 15], employing partitioning techniques [6, 9, 14], supporting incremental updating and exploiting parallelism [10, 13, 16]. The main "limitation" of these approaches, however, is that

*This research is partial supported by a large grant from the Australian Research Council (DP0343109) and partial supported by large grant from the Guangxi Natural Science Funds

†Faculty of Information Technology, University of Technology, Sydney, PO Box 123, Broadway NSW 2007, Australia, and Guangxi Teachers University, Gulin, P R China. Email: zhangsc@it.uts.edu.au

‡Faculty of Information Technology, University of Technology, Sydney, PO Box 123, Broadway NSW 2007, Australia. Email: chengqi@it.uts.edu.au

they require multiple passes over the database. For very large databases that are typically disk resident, this requires reading the database completely for each pass resulting in a large number of disk I/Os.

An alternative approach is to take a sample of the database, and determine association rules that are valid on the sample database. In other words, the problem of mining the association rules becomes a 3-step procedure:

(1) Generate a random subset of a given large database;
(2) Generate all large itemsets in the random subset;
(3) Generate all the rules with both support and confidence greater than or equal to minimum support and minimum confidence respectively.

As the sample size is typically very much smaller than the original database size, the association rules on the sample can be obtained at a much faster time. We shall refer to these association rules (obtained from the sample) as *approximate association rules*. The key issue in this approach is to pick a right sample that is representative of the database, so that the approximate association rules are indeed the association rules that hold on the database.

In this paper, we reexamine mechanisms for the 3 steps discussed above. To obtain a random sample of the database, we apply the central limit theorem. As we shall see shortly, the use of the central limit theorem allows us to cut down the sample size by about half compared to known techniques [11, 12]. For the second subtask, a new algorithm for generating all possible useful itemsets for mining rules with both positive and negative itemsets is proposed. Finally, the last subtask is solved by generating all positive and negative association rules.

Unfortunately, because of approximation errors, it is possible for some valid rules to be missed, while other invalid rules may be generated. To deal with this problem, we adopt a two phase approach. First, we discover all promising approximate rules from a random sample of the database. Second, these approximate results are used as heuristic information in an efficient algorithm that requires only one-pass of the database to validate rules that have support and confidence close to the desired support and confidence values. We evaluated the proposed technique, and our experimental results demonstrate that the approach is efficient and promising.

The rest of this paper is organized as follows. In the next section, we briefly review some concepts and definitions. In Section 3, we apply the central limit theorem to mine approximate association rules. In order to discover both of positive and negative association rules, an algorithm to generate all possible useful itemsets is also proposed. In Section 4, we evaluate the effectiveness of the proposed approach experimentally. In Section 5, we propose a method to (1) assist knowledge discovery and (2) determine the validation of the rules with support or confidence close to the user-specified thresholds. Finally, we summarize our contributions in section 6.

2 Basic Concepts

One of the most widely used data mining model for association rules is the support-confidence framework established by Agrawal, Imielinski, and Swami [1]. We shall review some of the concepts here.

Let $I = \{i_1, i_2, \cdots, i_N\}$ be a set of N distinct literals called *items*. D is a set of variable length transactions over I. A transaction is a set of items, i.e., a subset of I. A transaction has an associated unique identifier called TID.

In general, a set of items (such as the antecedent or the consequent of a rule) is referred to as an *itemset*. For simplicity, an itemset $\{i_1, i_2, i_3\}$ is sometimes written as $i_1 i_2 i_3$.

For an itemset $A \subseteq I$ and a transaction $T \in D$, A is purchased (occurred) in T (or T contains A) if $\forall a \in A(\exists i((1 \leq i \leq n) \wedge (T(i) = a)))$, where '$T(i)$' is i^{th} element of T.

The number of items in an itemset is the *length* (or the *size*) of an itemset. Itemsets of some length k are referred to as a k-itemsets.

An itemset has an associated measure of statistical significance called *support*, denoted as *supp*. For an itemset $A \subseteq I$, $supp(A) = s$, if the fraction of transactions in D containing A equals s. An itemset A is a large itemset if $supp(A) \geq min_{supp}$, where 'min_{supp}' is a user specified minimum support.

While A indicates the occurrence of an itemset A, the *negation* of A means the nonoccurrence of A, stood for $\overline{A}$ [1]. The support of $\overline{A}$ is as $supp(\overline{A}) = 1 - supp(A)$. Generally, for itemsets $A = \{i_1, \cdots, i_m\}$ and $B = \{j_1, \cdots, j_n\}$, the support of $\overline{A} \cup B$ is $supp(\overline{A} \cup B) = supp(B) - supp(A \cup B) = supp(\{j_1, \cdots, j_n\}) - supp(\{i_1, \cdots, i_m, j_1, \cdots, j_n\})$.

An *association rule* is an implication of the form $A \Rightarrow B$ (or written as $A \rightarrow B$), where $A, B \subseteq I$, and $A \cap B = \emptyset$. A is called the *antecedent* of the rule, and B is called the *consequent* of the rule.

An association rule $A \rightarrow B$ has a measure of its strength called *confidence* (denoted as $conf$) defined as the ratio $supp(A \cup B)/supp(A)$, where $A \cup B$ means that both A and B are present in transactions.

The work in this paper extends traditional associations to include association rules of forms $A \rightarrow \overline{B}$, $\overline{A} \rightarrow B$, and $\overline{A} \rightarrow \overline{B}$, which indicate negative associations between itemsets. We call rules of the form $A \rightarrow B$ *positive association rules*, and rules of the other forms *negative association rules*. Negative rules indicate that the presence of some itemsets will imply the absence of other itemsets in the same transactions. Negative rules are also very useful in association analysis, although they are hidden and different from positive association rules.

The problem of mining association rules is to generate all rules $A \rightarrow B$ that have both support and confidence greater than or equal to some user specified minimum support (min_{supp}) and minimum confidence (min_{conf}) thresholds respectively, i.e.

[1]An itemset A is often taken as an event in computations meaning that A is true in a transaction if item i presents in the transaction for $\forall i \in A$. $\overline{A}$ is taken as an event in computations meaning that $\overline{A}$ is true in a transaction if item i does not present in the transaction for $\exists i \in A$. That is, $\overline{A}$ is different from $I - A$.

for regular associations:

$$supp(A \cup B) \geq min_{supp}, \qquad conf(A \to B) = \frac{supp(A \cup B)}{supp(A)} \geq min_{conf}.$$

It can be decomposed into the following two subproblems.

(1) All itemsets that have support greater than or equal to the user specified minimum support are generated. That is, generating all large itemsets.

(2) Generate all the rules that have minimum confidence in the following naive way: For every large itemset X and any $B \subset X$, let $A = X - B$. If the rule $A \to B$ has the minimum confidence (or $supp(X)/supp(A) \geq min_{conf}$), then it is a valid rule.

Example 1. *Let* $T_1 = \{i_1, i_2, i_4\}$, $T_2 = \{i_1, i_2, i_4\}$, $T_3 = \{i_2, i_3, i_4\}$, $T_4 = \{i_2, i_3, i_4\}$, *and* $T_5 = \{i_1, i_2\}$ *be the only transactions in a database. Let the minimum support and minimum confidence be 0.6 and 0.85 respectively. Then the large itemsets are the following:* $\{i_1\}$, $\{i_2\}$, $\{i_4\}$, $\{i_1, i_2\}$ *and* $\{i_2, i_4\}$. *The valid rules are* $i_1 \to i_2$ *and* $i_4 \to i_2$.

3 Mining Approximate Rules

In probability theory, if a situation is such that only two outcomes, often called success and failure, are possible, it is usually called a *trial*. The variable element in a trial is described by a probability distribution on a sample space of two elements, 0 representing failure and 1 success; this distribution assigning the probability $1-\theta$ to 0 and θ to 1, where $0 \leq \theta \leq 1$. Suppose we consider n independent repetitions of a given trial. The variable element in these is described by a probability distribution on a sample space of 2^n points, the typical point being $x = (x_1, x_2, \cdots, x_n)$, where each x_i is 0 or 1, and x_i represents the result of the ith trial. The appropriate probability distribution is defined by

$$p_\theta(x) = \theta^{m(x)}(1-\theta)^{n-m(x)},$$

where $m(x) = \sum_{i=1}^{n} x_i$ is the number of 1s in the results of the n trials, this being so since the trials are independent.

Given an x in this situation it seems reasonable to estimate θ by $m(x)/n$, the proportion of successes obtained. This seems in some sense to be a 'good' estimate of θ.

In data mining, a database D can be taken as a trial. For any itemset A, it is 1 if the itemset A occurs in a transaction T (written as $T(A)$), else it is 0 (written as $\neg T(A)$). Let P be the set of all transactions that the itemset A occurs in, and Q be the set of all transactions that the itemset A doesn't occur in. Then P and Q are partitions of D as follows.

$$P = \{T | T(A)\},$$
$$Q = \{T | \neg T(A)\}.$$

Given a database, its n transactions can be viewed as n independent data stored in the database. Certainly, each transaction has two possible outcomes for an itemset A, which are 1 and 0. Suppose the probability of A occurring in the database is p and the probability of A not occurring is $q = 1 - p$. Since the database is static, we can say that probability p of A occurring in each transaction is the same for each transaction. Hence, this given database can be taken as a Bernoulli trial.

3.1 The Application of Central Limit Theorem

The central limit theorem is one of the most remarkable results in probability theory. Loosely put, it states that the sum of a large number of independent random variables has a distribution that is approximately normal. Hence it not only provides a simple method for computing approximating probabilities for sums of independent random variables, but it also helps explain the remarkable fact that the empirical frequencies of so many natural populations exhibit bell-shaped (that is, normal) curves. In its simplest form the central limit theorem is as follows.

Let $X_1, X_2, \cdots, X_n$ be a sequence of independent and identically distributed random variables, each having finite mean $E(X_i) = \mu$ and $Var(X_i) = \sigma^2$. Then the distribution of

$$\frac{X_1 + \cdots + X_n - n\mu}{\sigma\sqrt{n}}$$

tends to the standard normal as $n \to \infty$. That is,

$$P\{\frac{X_1 + \cdots + X_n - n\mu}{\sigma\sqrt{n}} \leq a\} \to \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{a} e^{-x^2/2} dx \quad as \ n \to \infty. \tag{1}$$

Readers are referred to [5] for other concepts and theorems.

We now set up a new mining model in this subsection, which applies central limit theorem to mine approximate association rules from large databases.

Theorem 1. *Let I be the set of items in database D, $A \subseteq I$ an itemset, $\eta > 0$ the degree of asymptotic to association rules and $\xi \geq 0$ the upper probability of $P[|Ave(X_n) - p| \leq \eta]$, where $Ave(X_n)$ is the average of A occurring in n transactions in D and p is the probability of A in D. Suppose records in D are matched Bernoulli trials. If n random records of D is enough for determining the approximate association rules in D according to central limit theorem, n must be as follows:*

$$n \geq \frac{z^2((1+\xi)/2)}{4\eta^2} \tag{2}$$

where $z(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2/2} dy$ is a standard normal distribution function, which can find out it from the Appendix in [5].

Proof. From the given conditions in this theorem, we take

$$P(|Ave(X_n) - p| \leq \eta) = \xi$$

Clearly,

$$\begin{aligned}P(|Ave(X_n) - p| \leq \eta) &= P(-\eta \leq (Ave(X_n) - p) \leq \eta)\\ &= P(\frac{-\eta}{1/(2\sqrt{n})} \leq \frac{Ave(X_n) - p}{1/(2\sqrt{n})} \leq \frac{\eta}{1/(2\sqrt{n})})\\ &\approx N(2\eta\sqrt{n}) - N(-2\eta\sqrt{n})\\ &= 2N(2\eta\sqrt{n}) - 1\end{aligned}$$

where $N()$ is the distribution function of the standard normal distribution. And for this probability to equal ξ we need

$$N(2\eta\sqrt{n}) = \frac{1}{2}(1 + \xi)$$

which is satisfied by

$$2\eta\sqrt{n} = z((1 + \xi)/2)$$

the required value for n then is

$$n \geq \frac{z^2((1 + \xi)/2)}{4\eta^2}$$

□

Example 2. *Suppose a new process is available for doping silicon chips, used in electronic devices. p (unknown) is the probability that each chip produced in this way is defective. We assume that the defective chips are independent of each other. How many chips, n, must we produce and test so that the proportion of defective chips found ($Ave(X_n)$) does not differ from p by more than* 0.01, *with probability at least* 0.99*? That is, we want n such that*

$$P(|Ave(X_n) - p| < 0.01) > 0.99,$$

$\eta = 0.01, \xi = 0.99, z(0.995) = 2.57$, *we have*

$$n = \frac{2.57^2}{4 * 0.01^2} = 16513,$$

considerably smaller than the value n = 27000 *that is needed by using the model in Chernoff bounds [11, 12].*

Based on Theorem 1, the random target database can be obtained in two steps: (1) generating a set X of pseudo-random numbers, where $|X| = n$ and (2) generating the random database RD (instance set) from D using pseudo-random number set X. That is, for any $x_i \in X$, get $(x_i + 1)^{th}$ record of D and append it into RD.

Note that generating random database RD of the given database D doesn't mean to establish a new database RD. It only needs to build a view RD over D.

3.2 Mining Approximate Association Rules

In this subsection, we construct a new model for discovering both of positive and negative association rules. For this goal, an algorithm of generating all positive and negative large itemsets is also proposed.

Positive and Negative Large Itemsets

For mining general approximate association rules, all positive and negative large itemsets in a random database would be generated. For example, if one of $A \to \overline{B}$, $\overline{A} \to B$ and $\overline{A} \to \overline{B}$ can be discovered, then one of $supp(A \cup \overline{B}) \geq min_{supp}$, $supp(\overline{A} \cup B) \geq min_{supp}$ and $supp(\overline{A} \cup \overline{B}) \geq min_{supp}$ must hold. This means that $supp(A \cup B) < min_{supp}$. However, itemsets such as $A \cup B$, are not generated as large itemsets into the set of all large itemsets. In order to mine negative rules, we present a procedure to generate all positive and negative large itemsets in a random database as follows.

```
Procedure 1. PNLargeItemsets
    Input: D: database; $min_{supp}$: minimum support;
    Output: PL: large itemsets; NL: negative large itemsets;
    Begin
  (1) generate sample RD of a given database D;
      let $PL \leftarrow \emptyset$; $NL \leftarrow \emptyset$;
  (2) let $L_1 \leftarrow \{large\ 1\text{-}itemsets\}$; $PL \leftarrow PL \cup L_1$;
  (3) for ($k = 2$; ($L_{k-1} \neq \emptyset$); $k++$) do
      begin //Generate all possible positive and negative k-itemsets of interest in RD.
    (3.1)  let $L_k \leftarrow \{\{x_1, \ldots, x_{k-2}, x_{k-1}, x_k\} \mid \{x_1, \ldots, x_{k-2}, x_{k-1}\} \in L_{k-1} \wedge \{x_1, \ldots x_{k-2}, x_k\} \in L_{k-1}\}$;
    (3.2)  for each transaction t in RD do
             begin
           //Check which k-itemsets are included in transaction t.
              let $t_{Tem} \leftarrow$ the k-itemsets in t that are also contained in $L_k$;
              for each itemset A in $t_{Tem}$ do
                let $A.count \leftarrow A.count + 1$;
             end
    (3.3) //Selecting all positive k-itemsets in $L_k$
           let $Tem_k \leftarrow \{C | C \in L_k \wedge (supp(C) = (C.count/|RD|) >= min_{supp})\}$;
           let $PL \leftarrow PL \cup Tem_k$;
           //Selecting all negative k-itemsets in $L_k$
             let $NL \leftarrow NL \cup (L_k - Tem_k)$;
      end
  (4) output PL and NL;
    End.
```

The procedure *PNLargeItemsets* generates all positive and negative itemsets in the sample *RD*. The initialization and generating sample *RD* of a given database *D* are done in Step (1). Step (2) counts the frequencies of itemsets in *RD*. Step (3) generates all positive and negative itemsets of interest.

Rules of Interest

In [8], Piatetsky-Shapiro argued that a rule $X \to Y$ is not interesting if

$$supp(X \to Y) \approx supp(X)supp(Y)$$

According to probability interpretation [3]: $supp(X \cup Y) = P(X \cup Y)$ and $conf(X \to Y) = P(Y|X) = P(X \cup Y)/P(X)$ Then Piatetsky-Shapiro's argument can be denoted as

$$P(X \cup Y) \approx P(X)P(Y).$$

This means that $X \to Y$ cannot be extracted as a rule if $P(X \cup Y) \approx P(X)P(Y)$. Actually, $P(X \cup Y) \approx P(X)P(Y)$ denotes X is approximately independent to Y in probability theory. A statistical definition [3] of dependence of the sets X and Y is

$$Interest(X,Y) = \frac{P(X \cup Y)}{P(X)P(Y)},$$

with the obvious extension to more than two sets. This formula is referred to as the *interest* of Y given X is one of the main measurements of uncertainty of association rules. Certainly, the further the value is from 1, the more the dependence. Or for $1 > \lambda > 0$, if $|\frac{P(X \cup Y)}{P(X)P(Y)} - 1| \geq \lambda$, then $X \to Y$ is a rule of interest.

By Piatetsky-Shapiro's argument, we can divide $Interest(X,Y)$ into 3 cases as follows:

(1) if $P(X \cup Y)/(P(X)P(Y)) = 1$, then $P(X \cup Y) = P(X)P(Y)$ or Y and X are independent;
(2) if $P(X \cup Y)/(P(X)P(Y)) > 1$, or $P(X \cup Y) > P(X)P(Y)$, then Y is positively dependent to X;
(3) if $P(X \cup Y)/(P(X)P(Y)) < 1$, or $P(X \cup Y) < P(X)P(Y)$, then Y is negatively dependent to X, or $\overline{Y}$ is positively dependent to X.

In this way, we can define another form of interpretation of rules of interest as follows. For $1 > \lambda > 0$, (a) if $\frac{P(X \cup Y)}{P(X)P(Y)} - 1 \geq \lambda$, then $X \to Y$ is a rule of interest; and (b) if $-(\frac{P(X \cup Y)}{P(X)P(Y)} - 1) \geq \lambda$, then $X \to \overline{Y}$ is a rule of interest.

Theorem 2. *Let I be the set of items in database D, $X, Y \subseteq I$ be itemsets, $X \cap Y = \emptyset$, $P(X) \neq 0$ and $P(Y) \neq 0$. min_{supp}, min_{conf} and $\lambda > 0$ are given by users or experts. If*

(1) $supp(X \cup Y) \geq min_{supp}$, $conf(X \to Y) \geq min_{conf}$, and $P(X \cup Y) - P(X)P(Y) \geq \lambda$, then $X \to Y$ can be extracted as a rule of interest.

(2) $supp(X \cup \overline{Y}) \geq min_{supp}$, $supp(Y) \geq min_{supp}$, $conf(X \to \overline{Y}) \geq min_{conf}$, and $-(P(X \cup Y) - P(X)P(Y)) \geq \lambda$, then $X \to \overline{Y}$ can be extracted as a rule of interest.

Proof. From assumption of the above theorem, we have

$$\frac{|(P(X \cup Y) - P(X)P(Y))|}{P(X)P(Y)} \geq \frac{\lambda}{P(X)P(Y)},$$

or

$$|\frac{P(X \cup Y)}{P(X)P(Y)} - 1| \geq \frac{\lambda}{P(X)P(Y)}.$$

Because $0 < P(X)P(Y) \leq 1$, so $\lambda/(P(X)P(Y)) \geq \lambda$. Hence,

$$|\frac{P(X \cup Y)}{P(X)P(Y)} - 1| \geq \lambda,$$

That is, $X \to Y$ can be extracted as a rule of interest. □

Mining Positive And Negative Rules

By our definition on interest, if $P(X \cup Y) \approx P(X)P(Y)$, X is approximately independent to Y in probability theory; if the greater the value of $P(X \cup Y) - P(X)P(Y) > 0$ is, the more the positive dependence; and if the smaller the value of $P(X \cup Y) - P(X)P(Y) < 0$ is, the more the negative dependence. However, $-P(X)P(Y) \leq P(X \cup Y) - P(X)P(Y) \leq P(X)(1 - P(Y))$. In order to reflect this relationship between $P(X \cup Y)$ and $P(X)P(Y)$, we propose the *probability ratio* (PR) model here. Under the PR model, we define the measure PR to determine the degree in which the valid rule $X \to Y$ is interesting.

$$PR(Y|X) = \begin{cases} \frac{P(X \cup Y) - P(X)P(Y)}{P(X)(1-P(Y))}, & if \ \ P(X \cup Y) \geq P(X)P(Y), \\ & \quad\ \ P(X)(1 - P(Y)) \neq 0. \\ \frac{P(X \cup Y) - P(X)P(Y)}{P(X)P(Y)}, & if \ \ P(X \cup Y) < P(X)P(Y), P(X)P(Y) \neq 0. \end{cases}$$

Certainly, PR has some properties as follows.

Property 1. *PR satisfies the following:*

$$PR(Y|X) + PR(\overline{Y}|X) = 0.$$

Proof. We shall only prove the property holds when $P(X \cup Y) \geq P(X)P(Y)$. The others can be derived in a similar manner. Since

$$P(X \cup Y)/P(X) = P(Y|X), P(X \cup \overline{Y})/P(X) = P(\overline{Y}|X), P(Y|X) + P(\overline{Y}|X) = 1,$$

and

$$P(\overline{Y}|X) = 1 - P(Y|X) \leq 1 - P(Y) = P(\overline{Y}).$$

Therefore,

$$PR(Y|X) = \frac{P(X \cup Y) - P(X)P(Y)}{P(X)(1 - P(Y))} = \frac{P(Y|X) - P(Y)}{1 - P(Y)},$$

$$PR(\overline{Y}|X) = \frac{P(X \cup \overline{Y}) - P(X)P(\overline{Y})}{P(X)P(\overline{Y})} = \frac{P(\overline{Y}|X) - P(\overline{Y})}{P(\overline{Y})}.$$

Hence,

$$PR(Y|X) + PR(\overline{Y}|X) = \frac{P(X \cup Y) - P(X)P(Y)}{P(X)(1 - P(Y))} + \frac{P(X \cup \overline{Y}) - P(X)P(\overline{Y})}{P(X)P(\overline{Y})}$$

$$= \frac{P(Y|X) - P(Y)}{1 - P(Y)} + \frac{P(\overline{Y}|X) - P(\overline{Y})}{P(\overline{Y})}$$

$$= \frac{P(Y|X) - P(Y)}{1 - P(Y)} + \frac{(1 - P(Y|X)) - (1 - P(Y))}{1 - P(Y)} = 0.$$

So, we have $PR(Y|X) + PR(\overline{Y}|X) = 0$. □

We now apply the PR model to measure the uncertainties of association rules.

(1) For an association rule $A \rightarrow B$, its *supp*$(A \cup B)$ is $P(A \cup B)$ and, $PR(B|A)$ is taken as the confidence of the rule. The task of mining this association rule is defined as follows. For itemset $A \cup B$, if *supp*$(A \cup B) \geq min_{supp}$ and $PR(B|A) \geq min_{conf}$, then $A \rightarrow B$ can be extracted as a valid rule.

(2) For an association rule $A \rightarrow \overline{B}$, $PR(\overline{B}|A) = -PR(B|A)$ according to Property 1. Therefore, if *supp*$(A \cup \overline{B}) \geq min_{supp}$, *supp*$(B) \geq min_{supp}$, $PR(B|A) < 0$ and $PR(\overline{B}|A) \geq min_{conf}$, then $A \rightarrow \overline{B}$ can extracted as a valid rule.

(3) For $\overline{A} \rightarrow B$, $PR(B|\overline{A})$ is taken as the confidence of the rule. The task of mining this association rule is defined as follows. For itemset $\overline{A} \cup B$, if *supp*$(A) \geq min_{supp}$, *supp*$(\overline{A} \cup B) \geq min_{supp}$ and $PR(B|\overline{A}) \geq min_{conf}$, then $\overline{A} \rightarrow B$ can be extracted as a valid rule.

(4) For an association rule $\overline{A} \rightarrow \overline{B}$, $PR(\overline{B}|\overline{A}) = -PR(B|\overline{A})$ according to Property 1. Therefore, if *supp*$(\overline{A} \cup \overline{B}) \geq min_{supp}$, *supp*$(A) \geq min_{supp}$, *supp*$(B) \geq min_{supp}$, $PR(B|\overline{A}) < 0$ and $PR(\overline{B}|\overline{A}) \geq min_{conf}$, then $\overline{A} \rightarrow \overline{B}$ can extracted as a valid rule.

Note that the requirements that $supp(B) \geq min_{supp}$ and $supp(A) \geq min_{supp}$ ensure the probability significance of rules with negative itemsets.

We now demonstrate how to apply this model to discover association rules with the data in Example 1. Let $min_{supp} = 0.2$ and $min_{conf} = 0.4$.

Example 3. *For itemset $A \cup C$, $P(A) = 0.6$, $P(C) = 0.4$ and $P(A \cup C) = 0$, we have $P(A \cup C) < P(A)P(C)$. This means that the disbelief increases, or $A \rightarrow \overline{C}$ may be extracted as a rule of interest. Furthermore,*

$$PR(C|A) = \frac{P(A \cup C) - P(A)P(C)}{P(A)P(C)} = \frac{0 - 0.6 \bullet 0.4}{0.6 \bullet 0.4} = -1,$$

According to our model, $A \rightarrow \overline{C}$ can be extracted as a valid rule due to $PR(\overline{C}|A) = -PR(C|A) = 1 > min_{conf}$, $supp(A \cup \overline{C}) = 0.6 > min_{supp}$, and $supp(C) = 0.4 > min_{supp}$.

As we have seen, our PR model is both reasonable and comprehensive. And general association rules can be easily discovered. Furthermore, we can obtain the following theorem that facilitates the extraction of interesting rules.

Theorem 3. *Let I be the set of items in database D, $X, Y \subseteq I$ be itemsets, $X \cap Y = \emptyset$, $P(X) \neq 0$ and $P(Y) \neq 0$. min_{supp}, min_{conf} and $\lambda > 0$ are given by users or experts. Then*

(1) if $supp(X \cup Y) \geq min_{supp}$ and $PR(Y|X) \geq Max\{min_{conf}, \lambda\}$, then $X \rightarrow Y$ can be extracted as a rule of interest;

(2) if $supp(X \cup \overline{Y}) \geq min_{supp}$, $supp(Y) \geq min_{supp}$ and $PR(\overline{Y}|X) \geq Max\{min_{conf}, \lambda\}$, then $X \rightarrow \overline{Y}$ can be extracted as a rule of interest;

(3) if $supp(\overline{X} \cup Y) \geq min_{supp}$, $supp(X) \geq min_{supp}$ and $PR(Y|\overline{X}) \geq Max\{min_{conf}, \lambda\}$, then $\overline{X} \rightarrow Y$ can be extracted as a rule of interest;

(4) if $supp(\overline{X} \cup \overline{Y}) \geq min_{supp}$, $supp(Y) \geq min_{supp}$, $supp(X) \geq min_{supp}$ and $PR(\overline{Y}|\overline{X}) \geq Max\{min_{conf}, \lambda\}$, then $\overline{X} \rightarrow \overline{Y}$ can be extracted as a rule of interest.

Proof. As before, we only prove (1) of the above theorem since the rest can be obtained similarly. We first prove that (1) holds. Since $PR(Y|X) \geq Max\{min_{conf}, \lambda\}$, according to the assumption in (1), we have $PR(Y|X) \geq min_{conf}$ and $PR(Y|X) \geq \lambda$.

On the other hand, because $PR(Y|X) \geq 0$, and using the Property 1, $PR(Y|X) + PR(\overline{Y}|X) = 0$, we have $-PR(\overline{Y}|X) \geq \lambda$.

According to previous interpretation of rules of interest, $X \rightarrow \overline{\overline{Y}}$ can be extracted as a rule of interest. That is

$$X \rightarrow Y$$

can be extracted as a rule of interest. □

Algorithm

Let D be a database, $|D|$ the total number of transactions in D, I the set of all items in D, and for $X \subseteq I$, $|X|$ the number of transactions in D that contain itemset X, min_{supp}, min_{conf}, λ and γ given by users. The algorithm of discovering association rules in our probability ratio model is constructed as follows.

```
Algorithm 1. PRModel
  Input: D: database, $min_{supp}, min_{conf}$, $\lambda$ and $\gamma$: threshold values;
  Output: approximate rules;
(1) Determine the sample size, n, based on the central limit theorem;
    Generate the sample database with n transactions;
    call routine PNLargeItemsets;
(2) for any large itemset A in PL begin
        for any itemset $X \subset A$ begin
          let $Y = A - X$;
          if $|PR(Y|X)| \geq Max\{min_{conf}, \lambda\}$ then
            output the rule $X \rightarrow Y$
              with confidence $PR(Y|X)$ and support $P(A)$;
          end
    for any itemset A in NL begin
        for any itemset $X \subset A$ begin
          let $Y = A - X$;
          if ($supp(X \cup \overline{Y}) \geq min_{supp}$ and $supp(Y) \geq min_{supp}$
              and $|PR(\overline{Y}|X)| \geq Max\{min_{conf}, \lambda\}$) then
            output the rule $X \rightarrow \overline{Y}$
              with confidence $PR(\overline{Y}|X)$ and support $P(A)$;
          end
          if ($supp(\overline{X} \cup Y) \geq min_{supp}$ and $supp(X) \geq min_{supp}$
              and $|PR(Y|\overline{X})| \geq Max\{min_{conf}, \lambda\}$) then
            output the rule $\overline{X} \rightarrow Y$
              with confidence $PR(Y|\overline{X})$ and support $P(A)$;
          end
          if ($supp(\overline{X} \cup \overline{Y}) \geq min_{supp}$ and $supp(X) \geq min_{supp}$
              and $supp(Y) \geq min_{supp}$ and $|PR(\overline{Y}|X)| \geq Max\{min_{conf}, \lambda\}$)
            then
            output the rule $\overline{X} \rightarrow \overline{Y}$
              with confidence $PR(\overline{Y}|\overline{X})$ and support $P(A)$;
          end
        end
    end
    endall.
```

Algorithm *PRModel* generates all positive association rules in PL and negative association rules in NL. Step (1) calls procedure *PNLargeItemsets* to generate the sets PL and NL with positive and negative large itemsets in the database D. Step (2) firstly generates positive association rules of interest of the form: $X \rightarrow Y$, in PL. If $PR(Y|X) \geq min_{conf}$, $X \rightarrow Y$ is extracted as a valid rule. If $PR(X|Y) \geq min_{conf}$, $Y \Rightarrow X$ is extracted as a valid rule. Secondly, the step generates negative association rules of interest of the forms $\overline{X} \rightarrow Y$, $Y \rightarrow \overline{X}$, $\overline{X} \rightarrow \overline{Y}$, and $\overline{Y} \rightarrow \overline{X}$, in NL.

4 An Experimental Study

To study the effectiveness of our model, we have performed several experiments. Our server is Oracle 8.0.3, and the algorithm is implemented on Sun SparcServer using Java, and JDBC API is used as the interface between the program and Oracle. The database used in our experiments has the following conceptual scheme

$$Report(sno, test, grade, area)$$

where *sno* is the primary key about student numbers, *test* is an attribute about examinations of subjects, *grade* is an attribute about students' grades with (A, B, C, D, E) as its domain, *area* is an attribute about students' nationality with a domain $(China, Singapore, \cdots)$. In order to illustrate the efficiency of our approximate rule model, we list partially the experimental results, which are the large itemsets and their supports. For more details, please refer to Appendix A.

Let $min_{supp} = 0.2$ and $min_{conf} = 0.6$. Some results are listed in Table 1.

We evaluated three methods: the traditional approach where the entire database is used (denoted D), the sampling approach based on Chernoff bounds [11, 12] (denoted LRD), and the proposed approach using the central limit theorem (denoted CRD). As shown in Table 1, the supports for the various useful itemsets are very close among the three methods. For example, the supports of item "China" are 37%, 36.78% and 36.48% for D, LRD and CRD respectively. This shows that relevant itemsets can be determined based on a small sample of the database. In our case, LRD requires only 15000 records which is only 15% of the original database size, while CRD makes use of no more than 7% of the original database size. We also note that the running time of mining the original database is 815 seconds. The time for LRD is 436 seconds (consisting of 207 seconds for LRD and 229 seconds for approximate rules), while that of CRD is only 241 seconds (consisting 101 seconds for LRD and 140 seconds for approximate rules). The significant reduction is clearly due to the smaller size of the samples. We also note that CRD is more efficient than LRD, making CRD a promising approach for mining association rules.

Referring to the Table, some of the rules of interest are $China \rightarrow B$, $China \rightarrow \overline{C}$, $China \rightarrow \overline{Singapore}$, $Singapore \rightarrow C$, $B \rightarrow \overline{C}$. However, from the example, we also note the following problems, which we shall investigate shortly.

(i) Some rules such as $China \rightarrow \overline{Singapore}$ and $B \rightarrow \overline{C}$ are also extracted as rules of interest.

(ii) Due to the probability significance and the constraint condition of min_{supp}, some rules such as $China \rightarrow \overline{D}$, $Singapore \rightarrow \overline{D}$, $China \rightarrow \overline{E}$ and $Singapore \rightarrow \overline{E}$, can't be extracted as negative rules of interest in our model. In some context, these rules are useful for applications. But mining rules such as $China \rightarrow \overline{Tom}$ has no significance, where "Tom" is name of some student.

Table 1: Some itemsets in the original database.

DB	useful Itemset	Support	size of sample	running time
D	China	37%	100000	815
	Singapore	50%		
	B	33.2%		
	C	42.05%		
	China, B	27.75%		
	Singapore, C	35%		
	China, Singapore	0%		
	China, C	3.1%		
	B, C	0%		
LRD	China	36.78%	15000	436
	Singapore	50.43%		
	B	33.43%		
	C	42.17%		
	China, B	27.83%		
	Singapore, C	34.97%		
	China, Singapore	0%		
	China, C	2.87%		
	B, C	0%		
CRD	China	36.48%	6724	241
	Singapore	50.82%		
	B	33, 45%		
	C	42.3%		
	China,B	27.71%		
	Singapore, C	35.07%		
	China, Singapore	0%		
	China, C	3.01%		
	B, C	0%		

As we have seen, if all data are randomly appended in to a given large database, the association rules can be approximated by our model using central limit theorem. The experiments also show the effectiveness of the proposed approach. Before closing this section, we shall make the following claim.

Claim 1. *Consider the given database D, we have*

(1) If all data are randomly appended into a given large database, association rules can be approximated by our model using limit theorems.

(2) If $A \rightarrow B$ can extracted as a rule in our model, it must be a rule of interest.

(3) The model in central limit theorem is more efficient than the model in Chernoff bounds.

We now explain these arguments. (1) can directly be proven by the above algorithm and Theorem 1; (2) can be obtained from Theorem 3 and Algorithm 1.

For Theorem 1 and model based on Chernoff bounds [11, 12], we can compare the efficiency between Chernoff bounds and central limit theorem as follows.

$$\frac{1}{2\eta^2} ln \frac{2}{1-\xi} || \frac{z^2_{(1+\xi)/2}}{4\eta^2},$$

where "||" is a comparison symbol, or

$$ln \frac{2}{1-\xi} || \frac{z^2_{(1+\xi)/2}}{2},$$

where $(1+\xi)/2 \geq 0.5$. According to the list of standard normal distribution function, the following inequality holds for $1 \geq \xi > 0$

$$ln \frac{2}{1-\xi} \geq \frac{z^2_{(1+\xi)/2}}{2}.$$

Hence,

$$\frac{1}{2\eta^2} ln \frac{2}{1-\xi} \geq \frac{z^2_{(1+\xi)/2}}{4\eta^2},$$

Thus, the model in central limit theorem is more efficient than the model in large number law, i.e., (3) in Claim holds.

□

5 Assisting Knowledge Discovery

As has been shown, our model is efficient to discover approximate association rules in large databases. However, if the support of an itemset A is in the neighbour of min_{supp}, then A can be sometimes be treated as a large itemset and sometimes not as a large itemset due to approximation errors. In other words, some such itemsets are large itemsets in D but not in RD, and some such itemsets are not large itemsets in D but they are large itemsets in RD. This is a weakness of our model. For example, consider a random subset RD of a given large database D. Let $min_{supp} = 0.2$ and the probability of error to be tolerated be 0.05. Let two itemsets A and B in D with probabilities (supports) 0.18 and 0.23 respectively. Assume also that A and B are generated with probabilities 0.21 and 0.194 respectively, in the random database RD. This means that A is a large itemset in RD and B is not a large itemset in RD due to approximating error 0.05. They are unexpected results.

On the other hand, if we cannot compromise the validity of mined rules, or when certain support and confidence are necessary for some applications, $\eta > 0$ can be expected to be much smaller. This implies that we have to end up with a very large sample of the database, which diminishes the gains of sampling.

However, because of the randomness of data in a given database, we can roughly generate a possible large itemset set at first. Then this set is used as heuristic information to obtain large itemsets with only one pass through the given database. In this way, we can use such heuristic information to (1) assist knowledge discovery where accuracy is important or certain support and confidence is desirable, and (2) determine if an itemset in the neighbour of min_{supp} in the random subset of a given database is a large itemset.

Definition 1. *If an itemset A in RD is greater than or equal to $min_{supp} - \eta$, then it is reasonable in probability to conjecture that A is a large itemset in the database D. And itemset such as A is called hopeful large itemset in D. Reversedly, if an itemset A in RD is less than $min_{supp} - \eta$, then it is reasonable and comprehensive in probability to believe that A is impossible as a large itemset in the database D.*

Apparently, assessing hopeful large itemset are not only useful to the itemsets in the neighbour of min_{supp}, but also efficient to assist non-approximate knowledge discovery in databases. We now present the algorithm of accomplishing such two tasks as follows.

Procedure 2. *TLargeItemset*

```
Input: $\eta$: accuracy of results, $\xi$: probability of requirements, $min_{supp}$: minimum support,
        D: original database, HLIsSet: set of hopeful large itemsets;
Output: LI: large itemsets D;
  Begin
   let $LI \leftarrow \emptyset$;
   for each transaction $\tau$ of D do
     for each itemset $\alpha$ of HLIsSet do
       if $\alpha \in \tau$ then
         let $Count_\alpha \leftarrow Count_\alpha + 1$;
   for each itemset $\alpha$ of HLIsSet do
     if $supp(\alpha) \geq min_{supp}$ then
       let $LI \leftarrow LI \cup \{\alpha\}$;
   output the set LI of all large itemsets in D;
  end;
```

Again, if the confidence of a rule $A \rightarrow B$ is in the neighbour of min_{conf}, then $A \rightarrow B$ can be sometimes extracted as a valid rule and sometimes not as a valid rule due to the approximate error. The problem of the neighbour of min_{conf} can be addressed using a similar method as that for the neighbour of min_{supp}.

Now, we can describe the model of applying our method to assist non-approximate knowledge discovery in databases as follows. For a given large database D, with the users specified min_{supp} and min_{conf}, the following steps are performed.

(1) Generate a random subset RD of D according to our model in this paper;
(2) Generate the set $HLIsSet$ of all hopeful large itemsets in RD with support greater than or equal to $max\{0, min_{supp} - approximate\ error\}$;
(3) Generate all large itemsets in D with support greater than or equal to min_{supp} according to the set of hopeful large itemsets and Procedure 2;
(4) Generate all the rules with both support and confidence greater than or equal to minimum support and minimum confidence respectively, according to the large itemsets in the given database.

Certainly, applying approximate results to assist knowledge discovery needs only rough estimation, such as $\eta = 0.01$ and $\xi = 0.9$ are enough to generate all hopeful large itemsets. On the other hand, Algorithm 1 is linear. It can be guaranteed by the following theorem.

Theorem 4. *For given large database D, let $n = |D|$, m be the time of generating the set $HLIsSet$ of all hopeful large itemsets in random subset RD of D. Then the time of generating all large itemset in D is at least $O(m + n^2)$.*

Proof. According to the above definition, Algorithm 1 and Procedure 2, it needs only one pass to read the given database D. And each reading takes $t + t'$ to read a transaction from D and count itemsets in $HLIsSet$, where t the time to read a transaction from D, and t' the time to count all itemsets in $HLIsSet$. So, n reading incurs time of $n(t + t')$. Hence, the time to generate all large itemsets in D is $m + n(t + t')$. Let t'' be the time to count an itemset. Then $t' = t''|HLIsSet|$ for general databases. Because $|HLIsSet|$ is at least $O(n)$, and t and t'' are two small constants, so $m + n(t + t') = m + n(t + t''|HLIsSet|)$ is at least $O(m + n^2)$. □

In order to handle the problem caused by both the neighbour of min_{supp} and the neighbour of min_{conf}, we can use two methods as follows. One of them is to take $max\{0, min_{supp} - \eta\}$ and $max\{0, min_{conf} - \eta\}$ as the minimum support and minimum confidence respectively, for applications that need only approximate results. If an application requires more accurate results or certain support and confidence, the following method can be performed.

For a given large database D, min_{supp} and min_{conf} are given by users.

(1) Generate a random subset RD of D;
(2) Generate all hopeful large itemsets in RD with support greater than or equal to $max\{0, min_{supp} - approximate\ error\}$;

(3) Generate the set $RSET$ of all the rules with both support and confidence greater than or equal to minimum support ($max\{0, min_{supp} - \eta\}$) and minimum confidence ($max\{0, min_{conf} - \eta\}$) respectively, according to the hopeful large itemsets in RD;

(4) For the subset PS of $RSET$ with both support and confidence in the neighbour of min_{supp} and the neighbour of min_{conf} respectively, generate the set VRS of all rules in PS that is valid in D;

(5) Output $(RSET - PS) \cup VRS$.

Theorem 5. *For given large database D, min_{supp} and min_{conf} are given by users. $A \to B$ can be extracted as an approximate rule in the above model if and only if $A \to B$ is a valid rule in D.*

Proof. We first prove ($\to$). According to the above assumption, if

(1) $supp(A \cup B) \geq max\{0, min_{supp} - \eta\}$; and

(2) $conf(A \to B) \geq max\{0, min_{conf} - \eta\}$;

hold in random subset RD of D. By (4) and (5) in the above definition, we can obtain

(i) $supp(A \cup B) \geq min_{supp}$; and

(ii) $conf(A \to B) \geq min_{conf}$;

This means, $A \to B$ is still a valid rule in D.

The proof of ($\Leftarrow$) can be directly obtained from Theorem 1, Theorem 3, and the above definition.

So, $A \to B$ can be extracted as an approximate rule in the above model if and only if $A \to B$ is a valid rule in D. □

6 Conclusions

Mining association rules is an expensive process. Mining approximate association rules on a sample of a large database can reduce the computation cost significantly. Srikant and Agrawal [11] suggested a method to select the sample of a given large database for estimating the support of candidates using Chernoff bounds. Also, Toivonen [12] applied the Chernoff bounds to discover association rules in large databases. However, previous approximate models based on Chernoff bounds may require a large sample size compared to the central limit theorem for discovering association rules in large databases. In this paper, we have addressed the issue of mining association rules, and have made the following contributions:

(1) Presented a method of applying the theorems to estimate the size of random database that enables us to mine approximate association rules.

(2) Proposed the algorithm to discover approximate association rules with negative itemsets. In particular, an algorithm of generating all possible useful (positive and negative large) itemsets is also presented.

(3) Demonstrated the effectiveness of our approach experimentally. Our results show that the approximating model is more efficient than models based on Chernoff bounds [11, 12].

(4) Proposed a method to (a) assist knowledge discovery and (b) determine the validation of a rule in the neighbour of min_{supp} or the neighbour of min_{conf} in the given database.

Acknowledgements

The authors would like to thank two anonymous reviewers for their detailed constructive comments on the first version of this paper.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, Mining association rules between sets of items in large databases. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1993:207-216.

[2] R. Agrawal, R. Srikant, Fast algorithms for mining association rules. In: *Proceedings of the 20th VLDB Conference*, 1994:487-499.

[3] S. Brin, R. Motwani and C. Silverstein, Beyond Market Baskets: Generalizing Association Rules to Correlations. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997: 265-276.

[4] S. Brin, R. Motwani, J. Ullman and S. Tsur, Dynamic Itemset Counting and Implication Rules for Market Basket data. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997: 255-264.

[5] R. Durrett, Probability: Theory and Examples, *Duxbury Press*, 1996.

[6] E. Omiecinski and A. Savasere, Efficient mining of association rules in large dynamic databases. In: *Proceedings of 16th British National Conference on Databases BNCOD 16*, Cardiff, Wales, UK, 1998.7: 49-63.

[7] J. Park, M. Chen, and P. Yu, Using a Hash-based Method with Transaction Trimming for Mining Association Rules. *IEEE Trans. Knowledge and Data Eng.*, vol. 9, 5(1997): 813-824.

[8] G. Piatetsky-Shapiro, Discovery, analysis, and presentation of strong rules. In: *Knowledge discovery in Databases*, G. Piatetsky-Shapiro and W. Frawley (Eds.), AAAI Press/MIT Press, 1991: 229-248.

[9] A. Savasere, E. Omiecinski, and S. Navathe, An efficient algorithm for mining association rules in large databases. *Proceedings of the 21st International Conference on Very Large Data Bases.* Zurich, Switzerland, 1995.8: 688-692.

[10] T. Shintani and M. Kitsuregawa, Parallel mining algorithms for generalized association rules with classification hierarchy. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1998: 25-36.

[11] R. Srikant and R. Agrawal, Mining generalized association rules. *Future Generation Computer Systems*, Vol. 13, 1997: 161-180.

[12] H. Toivonen, Sampling large databases for association rules. *Proceedings of the 22nd VLDB Conference*, 1996: 134-145.

[13] Shichao Zhang, Aggregation and maintenance for databases mining. *Intelligent Data Analysis: an international journal*, Vol. 3(6) 1999: 475-490.

[14] Shichao Zhang and Xindong Wu, Large Scale Data Mining Based on Data Partitioning. *Applied Artificial Intelligence: an international journal*, Vol. 15 2(2001): 129-139.

[15] Chengqi Zhang and Shichao Zhang, *Association Rules Mining: Models and Algorithms*. Springer, LNAI 2307, p.243, 2002.

[16] Shichao Zhang and Chengqi Zhang, Anytime Mining for Multi-User Applications. *IEEE Transactions on Systems, Man and Cybernetics* (Part B), Vol. 32 No. 4(2002).

Appendix A

The *size* of the given database is 100000 and $min_{supp} = 0.2$. In order to illustrate the efficiency of our approximate rule model, we partly list the experimental results, which are the large itemsets and their supports. The variables a, b, and x_0 are the initialized values used in the random number generator. In order to test the approximation, we list three different supports of each itemset from different samples with the same size as follows.

Some itemsets of PL and NL in original database

```
 /* 1-items */
Item = China, count = 37000, support = 37%
Item = Singapore, count = 50000, support = 50%
Item = B, count = 33200, support = 33.2%
Item = C, count = 42050, support = 42.05%

/* 2-items */
Itemset = {China, B}, count = 27750, support = 27.75%
Itemset = {Singapore, C}, count = 35000, support = 35%
Itemset = {China, Singapore}, count = 0, support = 0%
Itemset = {China, C}, count = 3100, support = 3.1%
Itemset = {B, C}, count = 0, support = 0%
```

Some itemsets of *PL* and *NL* in models based on Chernoff bounds

```
/* parameter value */
  Accuracy of result: 0.01
  Probability of requirements: 0.9
  RandomDBSize: 15000
  a = 53
  b = 113
  x0= 17

  /* 1-item */
 Item = China, count = 5517, support = 36.78%
 Item = Singapore, count = 7565, support = 50.43%
 Item = B, count = 5015, support = 33.43%
 Item = C, count = 6326, support = 42.17%

/* 2-items */
Itemset = {China, B}, count = 4175, support = 27.83%
Itemset = {Singapore, C}, count = 5246, support = 34.97%
Itemset = {China, Singapore}, count = 0, support = 0%
Itemset = {China, C}, count = 431, support = 2.87%
Itemset = {B, C}, count = 0, support = 0%
----------------------------------------------------------
/* parameter value */
  Accuracy of result: 0.01
  Probability of requirements: 0.9
  RandomDBSize: 15000
  a = 53
  b = 113
  x0= 43

  /* 1-item */
 Item = China, count = 5543, support = 36.95%
 Item = Singapore, count = 7449, support = 49.66%
 Item = B, count = 4946, support = 32.97%
 Item = C, count = 6300, support = 42%

/* 2-items */
Itemset = {China, B}, count = 4109, support = 27.39%
Itemset = {Singapore, C}, count = 5249, support = 34.99%
Itemset = {China, Singapore}, count = 0, support = 0%
Itemset = {China, C}, count = 411, support = 2.74%
Itemset = {B, C}, count = 0, support = 0%
```

```
----------------------------------------------------------
/* parameter value */
  Accuracy of result: 0.01
  Probability of requirements: 0.9
  RandomDBSize: 15000
  a = 53
  b = 113
  x0= 97

  /* 1-item */
 Item = China, count = 5513, support = 36.75%
 Item = Singapore, count = 7568, support = 50.45%
 Item = B, count = 5012, support = 33.41%
 Item = C, count = 6332, support = 42.21%

/* 2-items */
Itemset = {China, B}, count = 4172, support = 27.81%
Itemset = {Singapore, C}, count = 5252, support = 35.01%
Itemset = {China, Singapore}, count = 0, support = 0%
Itemset = {China, C}, count = 473, support = 3.15%
Itemset = {B, C}, count = 0, support = 0%
```

Some itemsets of PL and NL in central limit theorem

```
/* parameter value */
  Accuracy of result: 0.01
  Probability of requirements: 0.9
  RandomDBSize: 6724
  a = 53
  b = 113
  x0= 17

  /* 1-item */
 Item = China, count = 2453, support = 36.48%
 Item = Singapore, count = 3417, support = 50.82%
 Item = B, count = 2249, support = 33.45%
 Item = C, count = 2844, support = 42.3%

/* 2-items */
Itemset = {China, B}, count = 1863, support = 27.71%
Itemset = {Singapore, C}, count = 2358, support = 35.07%
Itemset = {China, Singapore}, count = 0, support = 0%
Itemset = {China, C}, count = 202, support = 3.01%
Itemset = {B, C}, count = 0, support = 0%
----------------------------------------------------------
```

```
/* parameter value */ }
  Accuracy of result: 0.01
  Probability of requirements: 0.9
  RandomDBSize: 6724
  a = 53
  b = 113
  x0= 43

  /* 1-item */
 Item = China, count = 2468, support = 36.7%
 Item = Singapore, count = 3350, support = 49.82%
 Item = B, count = 2216, support = 32.96%
 Item = C, count = 2829, support = 42.07%

/* 2-items */
Itemset = {China, B}, count = 1830, support = 27.22%
Itemset = {Singapore, C}, count = 2359, support = 35.08%
Itemset = {China, Singapore}, count = 0, support = 0%
Itemset = {China, C}, count = 196, support = 2.91%
Itemset = {B, C}, count = 0, support = 0%
----------------------------------------------------------
/* parameter value */
  Accuracy of result: 0.01
  Probability of requirements: 0.9
  RandomDBSize: 6724
  a = 53
  b = 113
  x0= 97

  /* 1-item */
 Item = China, count = 2456, support = 36.53%
 Item = Singapore, count = 3412, support = 50.74%
 Item = B, count = 2255, support = 33.54%
 Item = C, count = 2837, support = 42.19%

/* 2-items */
Itemset = {China, B}, count = 1867, support = 27.77%
Itemset = {Singapore, C}, count = 2350, support = 34.95%
Itemset = {China, Singapore}, count = 0, support = 0%
Itemset = {China, C}, count = 204, support = 3.04%
Itemset = {B, C}, count = 0, support = 0%
```

Received October, 2000

Mining Dynamic databases by Weighting*

Shichao Zhang† and Li Liu‡

Abstract

A dynamic database is a set of transactions, in which the content and the size can change over time. There is an essential difference between dynamic database mining and traditional database mining. This is because recently added transactions can be more 'interesting' than those inserted long ago in a dynamic database. This paper presents a method for mining dynamic databases. This approach uses weighting techniques to increase efficiency, enabling us to reuse frequent itemsets mined previously. This model also considers the novelty of itemsets when assigning weights. In particular, this method can find a kind of new patterns from dynamic databases, referred to *trend patterns*. To evaluate the effectiveness and efficiency of the proposed method, we implemented our approach and compare it with existing methods.

1 Introduction

In real-world applications, a business database is dynamic, in which (1) its content updates over time and (2) transactions are continuously being added. For example, the content and size of the transaction database of a supermarket change time by time, and different branches of Wal-Mart receive 20 million transactions a day. This generates an urgent need for efficiently mining dynamic databases.

While traditional data mining is developed for knowledge discovery in static databases, some algorithms have recently been developed for mining dynamic databases [4, 5, 6, 13]. However, there is an essential difference between dynamic database mining and traditional database mining. This is because recently added transactions can be more 'interesting' than those inserted long ago in a dynamic database. Actually, some items such as suits, toys, and some foods are with smart in market basket data. For example, "jean" and "white shirt" were often purchased in a duration from a department store, and "black trousers" and "blue T-shirt" were often purchased in another duration. The department store made different

*This research is partially supported by the Australian Research Council Discovery Grant (DP0343109) and partially supported by a large grant from the Guangxi Natural Science Funds

†Faculty of Information Technology, University of Technology, Sydney, PO Box 123, Broadway NSW 2007, Australia, and Guangxi Teachers University, Gulin, P R China. Email: zhangsc@it.uts.edu.au

‡Faculty of Information Technology, University of Technology, Sydney, PO Box 123, Broadway NSW 2007, Australia. Email: liliu@it.uts.edu.au

decisions on buying behavior according to such different purchased models. This means, some goods are very often purchased in a duration in market basket data, and they are solely purchased in another duration. These items are called *smart goods.* Apparently, most of smart items may not be frequent itemsets in a market basket data set. But they are useful to making decisions on latest buying behavior, referred to *trend pattern.*

Consequently, mining trend patterns is an important issue in mining market basket data. Indeed, since new data may represent the changing trend of customer buying patterns, we should intuitively have more confidence on the new data than on the old data. therefore, the novelty of data should be highlighted in mining models. However, mining customer buying behavior based on support-confidence framework (see [1]) can only reflect the frequency of itemsets but the trend of data. In this paper, a new method is proposed for mining association rules in dynamic databases. The proposed approach is based on the idea of weighted methods and aims at incorporating both the size of a database and the novelty of the data in the database.

The rest of this paper is organized as follows. In the next section, we first show our motivation, and then briefly recall some related work, concepts and definitions. In Section 3, a weight model of mining association rules for incremental databases is proposed. A competition is set up for tackling the problem of infrequent itemsets in Section 4. In Section 5, we show the efficiency of the proposed approach by experiments, and finally, we summarize our contributions in the last section.

2 Preliminaries

This section recall some previous work and concepts needed.

2.1 Related Work

Data mining can be used to discover useful information from data like 'when a customer buys milk, he/she also buys Bread' and 'customers like to buy Sunshine products'.

Strictly speaking, *data mining is a process of discovering valuable information from large amounts of data stored in databases, data warehouses, or other information repositories.* This valuable information can be such as patterns, associations, changes, anomalies and significant structures [19]. That is, data mining attempts to extract potentially useful knowledge from data.

Recently, mining association rules from large databases has received much attention [1, 7, 12, 15, 17]. However, most of them [1, 10, 12] presuppose that the goal pattern to be learned is stable over time. In real world, a database is often updated. Accordingly, some algorithms have recently been developed for mining dynamic databases [4, 5, 6, 13].

One possible approach to the update problem of association rules is to re-run the association rule mining algorithms [1] on the whole updated database. This approach, though simple, has some obvious disadvantages. All the computation done initially at finding the frequent itemsets prior to the update are wasted and all frequent itemsets have to be computed again from scratch. An incremental approach for learning from databases is due to [6], which uses the maintaining ideas in machine learning [13].

The most prevailing dynamic database mining model should be the FUP model proposed by Cheung *et al* [4] (The model will be detailed in Subsection 2.3). The FUP model reuses information from the old frequent itemsets. That is, old frequent itemsets and promising itemsets are required to be kept. This can significantly reduce the size of the candidate set to be searched against the original large database. Like the Aporiori algorithm [1], the FUP model employs the frequencies of itemsets to mine association rules. However, the FUP approach only need to scan the new data set for generating all the candidates. To deal with more dynamics, the authors also proposed an extended FUP algorithm, called $FUP2$ [5], for general updating operations, such as insertion, deletion and modification on databases.

However, previous models (such as the FUP model) use *retrace technique* to handle the problem that smaller itemsets become frequent itemsets during maintenance. The retrace technique is to re-mine the whole data set. Unfortunately, because the change is unpredicatable in applications, the technique may be repeatedly applied leading to poor performance.

In particular, previous models don't work well for the novelty of data. This paper will present techniques to address the above problems. Before figuring out our approach, we now present some well-known concepts for data mining and knowledge discovery used throughout this paper.

2.2 Basic Concepts

Let $I = \{i_1, i_2, \cdots, i_N\}$ be a set of N distinct literal called *items*. D is a set of variable length transactions over I. Each transaction contains a set of items $i_1, i_2, \cdots, i_k \in I$. A transaction has an associated unique identifier called TID. An *association rule* is an implication of the form $A \Rightarrow B$ (or written as $A \rightarrow B$), where $A, B \subset I$, and $A \cap B = \emptyset$. A is called the *antecedent* of the rule, and B is called the *consequent* of the rule.

In general, a set of items (such as the antecedent or the consequent of a rule) is called an *itemset*. The number of items in an itemset is the *length* (or the *size*) of an itemset. Itemsets of some length k are referred to as a k-itemsets. For an itemset $A \cdot B$, if B is an m-itemset then B is called an *m-extension* of A.

Each itemset has an associated measure of statistical significance called *support*, denoted as *supp*. For an itemset $A \subset I$, $supp(A) = s$, if the fraction of transactions in D containing A equals s. A rule $A \rightarrow B$ has a measure of its strength called *confidence* (denoted as $conf$) defined as the ratio $supp(A \cup B)/supp(A)$.

Definition 1. (support-confidence model): *If an association rule $A \rightarrow B$ has both support and confidence greater than or equal to some user specified minimum support* ($minsupp$) *and minimum confidence* ($minconf$) *thresholds respectively, i.e. for regular associations:*

$$supp(A \cup B) \geq minsupp$$

$$conf(A \rightarrow B) = \frac{supp(A \cup B)}{supp(A)} \geq minconf$$

then $A \rightarrow B$ can be extracted as a valid rule.

Mining association rules can be decomposed into the following two issues.

(1) All itemsets that have support greater than or equal to the user specified minimum support are generated. That is, generating all frequent itemsets.

(2) Generate all the rules that have minimum confidence in the following naive way: For every frequent itemset X and any $B \subset X$, let $A = X - B$. If the rule $A \rightarrow B$ has the minimum confidence (or $supp(X)/supp(A) \geq minconf$), then it is a valid rule.

Example 1. *Let $T_1 = \{A, B, D\}$, $T_2 = \{A, B, D\}$, $T_3 = \{B, C, D\}$, $T_4 = \{B, C, D\}$, and $T_5 = \{A, B\}$ be the only transactions in a database. Let the minimum support and minimum confidence be 0.6 and 0.85 respectively. Then the frequent itemsets are the following: $\{A\}$, $\{B\}$, $\{D\}$, $\{A, B\}$ and $\{B, D\}$. The valid rules are $A \rightarrow B$ and $D \rightarrow B$.*

2.3 The FUP Model

For comparison, we now present the *FUP* model [4]. Let D be a given database, D^+ the incremental data set to D, A be an itemset that occurs in D, A^+ stands for A occurring in D^+, Then A is a frequent itemset in $D \cup D^+$ only if the support of A is great than or equal to $minsupp$. We now define the *FUP* model as follows.

Definition 2. (*FUP* model): *An association rule $A \rightarrow B$ can be extracted as a valid rule in $D \cup D^+$ only if it has both support and confidence greater than or equal to minsupp and minconf respectively. Or*

$$supp(A \cup B) = \frac{t(A \cup B) + t(A^+ \cup B^+)}{c(D) + c(D^+)} \geq minsupp$$

$$conf(A \rightarrow B) = \frac{supp(A \cup B)}{supp(A)} \geq minconf.$$

where $c(D)$ and $c(D^+)$ are the cardinalities of D and D^+, respectively; $t(A)$ and $t(A^+)$ denote the number of tuples that contain itemset A in D and the number of tuples that contain itemset A in D^+, respectively.

According to the FUP model, the update problem of association rules can be reduced to finding the new set of frequent itemsets [4]. It can be divided into the following subproblems:

(1) Which old frequent itemsets will be become small in the updated database.

(2) Which old small itemsets will be become frequent in the updated database.

(3) Tackle these itemsets: delete the association rules $A \to B$ that $A \cup B$ became small in the updated database; apply the mining algorithms into the itemsets that became frequent in the updated database.

(4) How long would a database system be processed so as to update the factors of all itemsets.

The FUP algorithm works iteratively and its framework is Apriori-like (For details of the Apriori algorithm, please see [1]). At the kth iteration it performs three operations as follows:

1. Scan D^+ for any k-itemsets, A. If the support of A in D^+ is greater than, or equal to, *minsupp*, A is put into L'_k that is the set of frequent itemsets in D^+.
2. For any k-itemsets A in L'_k, if A is not in K_k that is the set of frequent itemsets in D, the support of A in $D \cup D^+$ is computed. If the support of A in $D \cup D^+$ is smaller than *minsupp*, A is removed from the candidate set of D^+.
3. A scan is conducted on D to update the support of A for each itemset in the candidate set of D^+.

3 Mining Strategies for Dynamic Databases

As we argued previously, the dynamic of databases is represented in two cases: (1) the content updates over time and (2) the size changes incrementally. When some transactions of a database are deleted or modified, it says that the content of the database has been updated. And this database is referred to an *updated database.* When some new transactions are inserted or appended into a database, it says that the size of the database has been changed. And this database is referred to *incremental database.* This section designs efficient strategies for mining updated databases and incremental databases.

3.1 Pattern Maintenance for Updated Databases

The update operation includes deletion and modification on databases. Consider transaction database

$$TD = \{\{A,B\};\{A,C\};\{A,B,C\};\{B,C\};\{A,B,D\}\}$$

where the database has several transactions, separated by a semicolon, and each transaction contains several items, separated by a comma.

The update operation on TD can basically be

Case-1 Deleting transactions from the database TD. For example, after deleting transaction $\{A, C\}$ from TD, the updated database is TD_1 as

$$TD_1 = \{\{A, B\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

Case-2 Deleting attributes from a transaction. For example, after deleting B from transaction $\{A, B, C\}$ in D, the updated database is TD_2 as

$$TD_2 = \{\{A, B\}; \{A, C\}; \{A, C\}; \{B, C\}; \{A, B, D\}\}$$

Case-3 Modifying existing attributes of a transaction in the database TD. For example, after modifying the attribute C to D for the transaction $\{A, C\}$ in TD, the updated database is TD_3 as

$$TD_3 = \{\{A, B\}; \{A, B\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

Case-4 Modifying a transaction in the database TD. For example, after modifying the transaction $\{A, C\}$ to $\{A, C, D\}$ for TD, the updated database is TD_4 as

$$TD_4 = \{\{A, B\}; \{A, C, D\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

Mining updated databases generates a significant challenge: the maintenance of their patterns. To capture the changes of data, for each time of updating a database, we may re-mine the updated database. This is a time-consuming procedure. In particular, when a database mined is very large and the changed content of each updating transaction is relatively small, re-mining the database is not an intelligent strategy, where an updating transaction is a set of update operations. Our strategy for updated databases is to scan the changed contents when the *information amount* of the changed contents is relatively small;, otherwise, the updated database is re-mined. The information amount is defined as follows.

Let D be a database with n transactions and the average number of attributes per transaction be m. Then the information amount of D is

$$Amount(D) = mn$$

Let UO be an updating transaction, consisting of the k update operations, N_1, N_2, $\cdots$, N_k, on the database D. As we have seen, each update operation, N_i, can generates two sets: $Additems(N_i)$ and $deleteitems(N_i)$. $Additems(N_i)$ is the set of records, in which items are added to the database D. And $deleteitems(N_i)$ is the set of records, in which items are deleted from the database D. For example,

- $Additems(UO_1) = \{\}$ and $deleteitems(UO_1) = \{\{A, C\}\}$ for the example in Case-1;
- $Additems(UO_2) = \{\}$ and $deleteitems(UO_2) = \{\{B\}\}$ for the example in Case-2;

- $Additems(UO_3) = \{\{D\}\}$ and $deleteitems(UO_3) = \{\{C\}\}$ for the example in Case-3; and
- $Additems(UO_4) = \{\{D\}\}$ and $deleteitems(UO_4) = \{\}$ for the example in Case-4.

For N_i $(1 \leq i \leq k)$ in UO, the information amount of N_i is the sum of $Amount(Additems(N_i))$ and $Amount(deleteitems(N_i))$. Then the information amount of UO is

$$Amount(UO) = \sum_{i=1}^{k}(Amount(Additems(N_i)) + Amount(deleteitems(N_i)))$$

For the above example, we have

(1) $Amount(UO_1) = 2$ for Case-1, where UO_1 is the update operation in the corresponding example;

(2) $Amount(UO_2) = 1$ for Case-2, where UO_2 is the update operation in the corresponding example;

(3) $Amount(UO_3) = 2$ for Case-3, where UO_3 is the update operation in the corresponding example; and

(4) $Amount(UO_4) = 1$ for Case-4, where UO_4 is the update operation in the corresponding example.

For an updating transaction UO on a database D, if $Amount(UO)/Amount(D) > \gamma$, the updated database D must be mined, where γ is a minimal relative information amount threshold. Otherwise, we only need to mine the changed contents in the updated database.

Let

$$\begin{aligned} Additems(UO) &= Additems(N_1) \cup Additems(N_2) \cup \cdots \cup Additems(N_k) \\ deleteitems(UO) &= deleteitems(N_1) \cup deleteitems(N_2) \cup \cdots \cup deleteitems(N_k) \end{aligned}$$

When mining the changed contents over the database D, we need to mine both $Additems(UO)$ and $deleteitems(UO)$. Let $X_{Additems}$ be the number of itemsets X occurring in $Additems(UO)$ and $X_{deleteitems}$ be the number of itemsets X occurring in $deleteitems(UO)$. The the number, $f(X)$, of itemsets X is

$$f(X) = X_{Additems} - X_{deleteitems}$$

There may be some *hopeful itemsets* in $Additems(UO)$. Let A be an itemsets in the changed contents and $|D|$ be the number of records in D. The relative support of A, $rsupp(A)$, is defined as

$$rsupp(A) = \frac{f(A)}{|D|}$$

When $rsupp(A)$ is large enough, the itemset A may be a frequent itemset in the updated database. This means that we may scan the database D for checking whether or not a hopeful itemset is frequent.

On the other hand, if the total information amount of the updating transaction UO and M updating transactions ($allCC = \{UO_1, UO_2, \cdots, UO_M\}$) is greater than γ, the database D must also be re-mined, where $UO_1 \leq \gamma$, $UO_2 \leq \gamma$, $\cdots$, $UO_M \leq \gamma$ and the M updating transactions have done before the updating transaction UO.

Including the above idea, the updating transaction UO leads to that the database D must be re-mined if

$$\begin{aligned} Mut(UO, allCC) &= Amount(UO)/Amount(D) > \gamma \bigvee \\ &\quad \frac{Amount(UO) + \sum_{i=1}^{M} Amount(UO_i)}{Amount(D)} > \gamma \end{aligned} \tag{1}$$

Below we design the algorithm for updated database mining.

Algorithm 1. *UpdatedDBMining;*

Input *D: original database; UO: set of update operations; FS: set of frequent itemsets in D; γ: minimal relative information amount; α: relative minimal support for itemsets in the changed contents;*

Output *newFS: set of frequent itemsets in the updated database;*

```
1.  compute $Amount(D)$;
2.  let $CC \leftarrow$ the changed contents;
3.  let $allCC \leftarrow \emptyset$
4.  compute $Amount(UO)$;
5.  if $Mut(UO, allCC)$ then begin
6.     mine the updated database and put frequent itemsets into $newFS$;
7.     $allCC \leftarrow \emptyset$;
8.  endif
9.     else begin
10.      $hopeset \leftarrow \emptyset$;
11.    $allCC \leftarrow$ the updating transaction $UO$;
12.      mine the changed contents $CC$;
13.      $Cset \leftarrow$ the set of items in $CC$;
14.      for any $A$ in $Cset$ do
15.        if $f(A) \geq \alpha$ then
16.          $hopeset \leftarrow hopeset \cup \{A\}$;
17.      $Candidate \leftarrow$ the set of itemsets in $FS$, in which each itemset contains at
    least an item in $Cset$;
```

```
18.     scan the updated database for hopeset and Candidate;
19.     generate newFS by FS, hopeset and Candidate;
20.     end else;
21. output newFS;
22. end procedure;
```

The algorithm *UpdatedDBMining* generates frequent itemsets in updated databases. The initialization is carried out in steps 1-4. Step 5 checks whether *Mut(UO, allCC)* is true or not. When *Mut(UO, allCC)* is true, the updated database must be mined in step 6 and the set *allCC* is emptied in step 7. Otherwise, we only need mine the changed contents in steps 9-20. The set *hopeset* is used to save all hopeful itemsets in the changed contents. Step 11 puts the updating transaction *UO* into *allCC*. Step 12 mines the changed contents. Steps 14-16 generate the set of hopeful itemsets. Step 17 generates the set of itemsets in *FS*, in which their supports have been changed. Step 18 takes one scan on the updated database for tackling both *hopeset* and *Candidate*, so as to generate all frequent itemsets in the updated database in step 19. Step 21 outputs all frequent itemsets in the updated database.

We now illustrate the use of this procedure by an example as follows.

Consider the above database TD. Let $minsupp = 0.4$. Then $Amount(TD) = 12$ and the frequent itemsets in TD are

$$A, 0.8; B, 0.8; C, 0.6; AB, 0.6; AC, 0.4; BC, 0.4$$

where there are 6 frequent itemsets, separated by a semicolon, and each frequent itemset contains 2 items, its name and frequency, separated by a comma.

Let $\gamma = 0.2$, $\alpha = 0.25$ and the first updating transaction is UO_1, in which the transaction $\{A, C\}$ is deleted from TD. Then the updated database is UD_1 as

$$UD_1 = \{\{A, B\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

And

$$\begin{aligned} Additems(UO_1) &= \emptyset \\ deleteitems(UO_1) &= \{\{A, C\}\} \end{aligned}$$

Because UO_1 is the first updating transaction, $allCC = \emptyset$. For UO_1, $Amount(UO_1) = 2$,

$$Amount(UO_1)/Amount(TD) = 2/12$$

and $Mut(UO_1, allCC)$ is not true. Consequently, we only need to mine the changed contents $\{A, C\}$ and the itemsets are A, C and AC. By the definition of the relative support of itemsets,

$$\begin{aligned} rsupp(A) &= \frac{f(A)}{|TD|} = \frac{1}{5} = 0.2 < \alpha \\ rsupp(C) &= \frac{f(C)}{|TD|} = \frac{1}{5} = 0.2 < \alpha \\ rsupp(AC) &= \frac{f(AC)}{|TD|} = \frac{1}{5} = 0.2 < \alpha \end{aligned}$$

Therefore,

$$\begin{aligned} hopeset &= \emptyset \\ allCC &= \{UO_1\} \end{aligned}$$

For the set of frequent itemsets in D, we have

$$Candidate = \{A, C, AB, AC, BC\}$$

By scanning the updated database for *Candidate*, we have

$$newFS = \{A, 0.6; B, 0.8; C, 0.4; AB, 0.6; BC, 0.4\}$$

Note that the size of the updated database can approximately be equal to the size of D when D is relatively large. Accordingly, the above example takes 5 as the size of the updated database, aiming at showing how to deal with the frequent itemsets using the changed contents.

Now, let the second updating transaction is UO_2, in which the transaction $\{B, C\}$ in UD_1 is modified as $\{A, B, C\}$. Then the updated database is UD_2 as

$$UD_1 = \{\{A, B\}; \{A, B, C\}; \{A, B, C\}; \{A, B, D\}\}$$

And

$$\begin{aligned} Additems(UO_2) &= \{\{A\}\} \\ deleteitems(UO_2) &= \emptyset \end{aligned}$$

For UO_2, $Amount(UO_2) = 1$ and

$$Amount(UO_2)/Amount(TD) = 1/12$$

Because $allCC = \{UO_1\}$, $Mut(UO_2, allCC)$ is true. Consequently, we need to mine the updated database UD_2 and the itemsets are as follows

$$newFS = \{A, 1; B, 1; C, 0.4; AB, 0.75; BC, 0.5; AC, 0.5; ABC, 0.5\}$$

The above examples have shown our strategy for effectively maintaining frequent itemsets in updated databases. We will focus on identifying trend patterns from incremental databases in the following sections.

3.2 Pattern Maintenance for Incremental Databases

The incremental operation includes insertion and appending on databases. Consider transaction database

$$D = \{\{F, H, I, J\}; \{E, H, J\}; \{E, F, H\}; \{E, I\}\}$$

where the database has several transactions, separated by a semicolon, and each transaction contains several items, separated by a comma.

The incremental operation on a database TD can be

(1) inserting transactions into the database TD. For example, after inserting two transactions $\{A, C\}$ and $\{A, B, C\}$ into TD before the transaction $\{E, H, J\}$, the incremental database is TD_1 as

$$TD_1 = \{\{F, H, I, J\}; \{A, C\}; \{A, B, C\}; \{E, H, J\}; \{E, F, H\}; \{E, I\}\}$$

(2) Appending transactions into the database TD. For example, after appending transactions $\{B, C\}$ and $\{A, B, D\}$ to TD, the incremental database is TD_2 as

$$TD_2 = \{\{F, H, I, J\}; \{E, H, J\}; \{E, F, H\}; \{E, I\}; \{B, C\}; \{A, B, D\}\}$$

¿From the above observations, both the inserting and appending operations do not change the original contents in the database TD. Therefore, we can take an incremental operation transaction as the union of the database TD and the incremental dataset D^+, where the dataset D^+ is a set of transactions that are added to TD by the incremental operation transaction.

Example 2. *Let TD be a set of a transaction database with 10 transactions in Table 1 which is obtained from a grocery store, where $A = bread$, $B = coffee$, $C = tea$, $D = sugar$, $E = beer$, $F = butter$ and $h = biscuit$. Assume that D^+ is a set of transactions in Table 2, which are new sales records in the grocery store, where $G = choclate$.*

The databases TD and D^+ are sets of data that represents the customer behaviors during two terms. And D^+ illustrates the latest customer behavior, whereas TD presents the old customer behavior. A frequent itemsets in D^+ is referred to *trend pattern*. Trend patterns are very important in marketing because they are useful in the decision-making of merchandize buying. For example, H is a trend pattern, which is frequently purchased in the new duration.

However, by using the support-confidence framework, H is not a frequent itemset in $TD \cup D^+$ when $minsupp = 0.4$. Therefore, trend pattern discovery has become a key issue in incremental database mining. On the other hand, to capture the novelty of data, for each incremental operation transaction, we may also re-mine the incremental database. In particular, when an original database mined is very large and the dataset generated by an incremental operation transaction is relatively small, re-mining the incremental database is time-consuming. Like the

Table 1: Transaction databases in TD

Transaction ID	Items
T_1	A, B, D, H
T_2	A, B, C, D
T_3	B, D, H
T_4	B, C, D
T_5	A, C, E
T_6	B, D, F
T_7	A, F
T_8	C, F
T_9	B, C, F
T_{10}	A, B, C, D, F

Table 2: Transaction databases in D^+

Transaction ID	Items
N_1	A, B, H
N_2	B, C, G, H
N_3	B, H

updated database mining, our strategy for mining incremental databases is to scan the incremental dataset when the information amount of the incremental dataset is relatively small; otherwise, the incremental database is re-mined.

For an incremental dataset D^+ added to a database D, if $Amount(D^+)/Amount(D) > \beta$, the incremental database $D \cup D^+$ must be mined, where β is a minimal relative information amount threshold. Otherwise, we only need to mine the incremental dataset and synthesize the patterns in D^+ and D by weighting (see Sections 4 and 5).

If the total information amount of the incremental dataset D^+ and M incremental datasets ($allaet = \{D_1^+, D_2^+, \cdots, D_M^+\}$) is greater than β, the database D must also be re-mined, where $D_1^+ \leq \beta$, $D_2^+ \leq \beta$, $\cdots$, $D_M^+ \leq \beta$ and the M datasets are added to D before the incremental dataset D^+ is.

Intuitively, the constraint $Amount(D^+)/Amount(D) > \beta$ indicates that the incremental dataset D^+ contains a information amount large enough to drives the mining of the incremental database. However, D^+ may only contain few records with much information. For example, let D^+ only contain a records with 200 distinct items in Example 2 and these items are also different from items in D. Certainly, the constraint $Amount(D^+)/Amount(D) > \beta$ holds. This leads to the mining of the incremental database. By using the support-confidence framework, there are no frequent itemsets in D^+ because the frequency of each item in D^+ is 1. Accordingly, we must take into account the size, $|D^+|$, of D^+ in our mining strategy. In this paper, the constraint is constructed as

$$constraint(D^+, D) = \frac{|D^+| * Amount(D^+)}{|D| * Amount(D)}$$

Including the above idea, the incremental dataset D^+ leads to that the database D must be re-mined if

$$\begin{aligned} Mid(D^+, allset) &= constraint(D^+, D) > \beta \bigvee \\ & constraint(D^+, D) + \sum_{i=1}^{M} constraint(D_i^+, D) > \beta \end{aligned} \quad (2)$$

Below we design the algorithm for incremental database mining.

Algorithm 2. *IncrementalDBMining;*

Input *D: original database; D^+: incremental dataset; FS: set of frequent itemsets in D; β: minimal relative information amount; minsupp: minimal support;*

Output *weightedFS: set of frequent itemsets by weighting;*

```
1. compute $Amount(D)$;
2. let $allset \leftarrow \emptyset$
3. compute $Amount(D^+)$;
4. if $Mid(D^+, allset)$ then begin
5.    mine the database $D \cup D^+$ and put frequent itemsets into $weightedFS$;
6.    $allset \leftarrow \emptyset$;
7.    endif
8.    else begin
9.    $allset \leftarrow$ the incremental dataset $D^+$;
10.     mine the incremental dataset $D^+$;
11.     $Pset \leftarrow$ the set of frequent itemsets in $D^+$;
12.     weight the support and confidence of $A$ in $D$ and $D^+$
13.     if $supp(A) \geq minsupp$ then
14.       $weightedFS \leftarrow A$;
15.     end else;
16. output $weightedFS$;
17. end procedure;
```

The algorithm *IncrementalDBMining* generates frequent itemsets in incremental databases. The initialization is carried out in steps 1-3. Step 4 checks whether $Mid(D^+, allset)$ is true or not. When $Mid(D^+, allset)$ is true, the incremental database must be mined in step 5 and the set *allset* is emptied in step 6. Otherwise, we only need mine the incremental dataset D^+ in steps 9-15. Step 9 puts the incremental dataset D^+ into *allset* when $Mid(D^+, allset)$ is not true. Step 10 mines the incremental dataset D^+. Steps 11-14 generate the set of frequent itemsets by weighting. Step 16 outputs all frequent itemsets in the incremental database.

The algorithm *IncrementalDBMining* includes a weighting procedure which is used to identify trend patterns. We will present the weighting technique in the following Sections.

4 Weight Method

Let D be a given database, D^+ the incremental dataset to D, A be an itemset that occurs in D, A^+ stands for A occurring in D^+. The support of A in the incremental database $D_1 = D \cup D^+$ is as follows

$$supp(A) = \frac{t(A)}{c(D) + c(D^+)} + \frac{t(A^+)}{c(D) + c(D^+)} \tag{3}$$

Where $c(D)$ and $c(D^+)$ are the cardinalities of D and D^+, respectively; and $t(A)$ and $t(A^+)$ denote the number of tuples that contain itemset A in D and the number of tuples that contain itemset A in D^+, respectively.

Let $supp_1(A) = t(A)/c(D)$ and $supp_2(A) = t(A^+)/c(D^+)$ stand for the supports of A in D and D^+, respectively. Then the equation (3) can be represented as

$$supp(A) = \frac{c(D)}{c(D) + c(D^+)} supp_1(A) + \frac{c(D^+)}{c(D) + c(D^+)} supp_2(A) \tag{4}$$

Let

$$\begin{aligned} k_1 &= \frac{c(D)}{c(D) + c(D^+)} \\ k_2 &= \frac{c(D^+)}{c(D) + c(D^+)} \end{aligned}$$

where k_1 and k_2 are the ratios of D and D^+ in the incremental database D_1, respectively. And the equation (4) can be represented as

$$supp(A) = k_1 * supp_1(A) + k_2 * supp_2(A) \tag{5}$$

For the equation (5), we can take k_1 and k_2 as the weights of D and D^+ in the incremental database D_1. This means, if a dataset has a larger number of transactions, the weight of the dataset is higher. And if a dataset has few transactions, the dataset is assigned a lower weight. Therefore, traditional data mining techniques, such as the support-confidence framework, can really be regarded as trivial weighting methods.

Consider the database D and the incremental dataset D^+ in Example 2. When $minsupp = 0.4$, the frequent itemsets in D and D^+ are listed in Tables 3 and 4, respectively.

Table 3: Frequent itemsets in D

Item	Number of Transactions	Support $p(X)$	Item	Number of Transactions	Support $p(X)$
A	5	0.5	B	7	0.7
C	6	0.6	D	6	0.6
F	5	0.5	BC	4	0.4
BD	5	0.5			

Table 4: Frequent itemsets in D^+

Item	Number of Transactions	Support $p(X)$
B	3	1
H	3	1
BH	3	1

After the transactions in D^+ are added to D to form the incremental database $D_1 = D \cup D^+$, the frequent itemsets in D_1 are listed in Table 5.

Table 5: Frequent itemsets in D_1

Item	Number of Transactions	Support $p(X)$
A	6	0.4615
B	10	0.769
C	7	0.5385
D	6	0.4615

From Tables 3, 4 and 5, the desirable patterns H and BH are not frequent itemsets in the incremental database D_1. To identify trend patterns such as H and BH, the novelty of data must be emphasized. In this paper, we propose to assign the incremental dataset D^+ a higher weight for stressing the novelty of data. For example, for the database D and the incremental dataset D^+, we have

$$\begin{aligned} k_1 &= \frac{c(D)}{c(D)+c(D^+)} = \frac{10}{13} = 0.769 \\ k_2 &= \frac{c(D^+)}{c(D)+c(D^+)} = \frac{3}{13} = 0.231 \end{aligned}$$

Taking into account the above idea, we assign D a weight $w_1 = 0.66$ and D^+ a weight $w_2 = 0.34$. And the support of an itemset X in D_1 is as follows

$$supp(X) = w_1 * supp_1(X) + w_2 * supp_2(X) \tag{6}$$

Hence, for the itemsets B, H and BH, we have

$$\begin{aligned} supp(B) &= 0.66 * supp_1(B) + 0.34 * supp_2(B) \\ &= 0.66 * 0.7 + 0.34 * 1 = 0.802 \\ supp(H) &= 0.66 * supp_1(H) + 0.34 * supp_2(H) \\ &= 0.66 * 0.2 + 0.34 * 1 = 0.472 \\ supp(BH) &= 0.66 * supp_1(BH) + 0.34 * supp_2(BH) \\ &= 0.66 * 0.2 + 0.34 * 1 = 0.472 \end{aligned}$$

This means that both H and BH are frequent itemsets in D_1 according to the equation (6). And all frequent itemsets in D_1 are listed in Table 6.

Table 6: Frequent itemsets in D_1

Item	Number of Transactions	Weighted Support $supp(X)$
A	6	0.445
B	10	0.802
C	7	0.51
H	5	0.472
BH	5	0.472

Comparison with Table 6, the supports of itemsets A, C and D are decreased in Table 7 because they are not frequent itemsets in the incremental dataset D^+. In particular, itemset D is not a frequent itemset in D_1 because

$$\begin{aligned} supp(D) &= 0.66 * supp_1(B) + 0.34 * supp_2(B) \\ &= 0.66 * 0.6 + 0.34 * 0 = 0.396 \end{aligned}$$

On the other hand, the supports of itemsets B, B and BH are increased in Table 7 because they are strongly supported in the incremental dataset D^+.

The above results have shown the following fact. In the weighting model, some infrequent itemsets (for example H and BH) can be interested, whereas some frequent itemsets (for example D) can be uninterested.

We now define the weighting model for maintaining association rules in incremental databases.

Definition 3. (Weighting model): *An association rule $X \to Y$ can be extracted as a valid rule in $D \cup D^+$ only if it has both support and confidence greater than or equal to minsupp and minconf respectively. Or*

$$supp_w(X \cup Y) = w_1 * supp_1(X \cup Y) + w_2 * supp_2(X \cup Y) \geq minsupp \quad (7)$$

$$conf_w(X \to Y) = \frac{supp_w(X \cup Y)}{supp_w(X)} \geq minconf \quad (8)$$

The confidence of the rule $X \to Y$ can be directly weighted as follows

$$conf_w(X \to Y) = w_1 * conf_1(X \to Y) + w_2 * conf_2(X \to Y) \geq minconf \quad (9)$$

Generally, for D, D_1, $\cdots$, D_n with weights $w_1, w_2, \cdots, w_{n+1}$, we define the weighted support, $supp_w(X)$, of itemset X as follows.

$$supp_w(X) = w_1 * supp(X) + w_2 * supp_1(XY) + \cdots + w_{n+1} * supp_n(X) \quad (10)$$

where, $supp(X)$, $supp_1(X)$, $\cdots$, $supp_n(X)$ are the the supports of the itemset X in D, D_1, $\cdots$, D_n respectively.

Let $X \to Y$ be an association rule in D, we define the weighted support $supp_w(X \cup Y)$ and confidence $conf_w(X \to Y)$ for $X \to Y$ as follows

$$\begin{aligned} supp_w(X \cup Y) &= w_1 * supp(X \cup Y) + w_2 * supp_1(X \cup Y) + \cdots && (11) \\ &\cdots + w_{n+1} * supp_n(X \cup Y) \\ conf_w(X \to Y) &= w_1 * conf(X \to Y) + w_2 * conf_1(X \to Y) + \cdots && (12) \\ &\cdots + w_{n+1} * conf_n(X \to Y) \end{aligned}$$

where, $supp(X \cup Y)$, $supp_1(X \cup Y)$, $\cdots$, $supp_n(X \cup Y)$ are the the supports of the rule $X \to Y$ in D, D_1, $\cdots$, D_n respectively; $conf(X \to Y)$, $conf_1(X \to Y)$, $\cdots$, $conf_n(X \to Y)$ are the confidences of the rule $X \to Y$ in D, D_1, $\cdots$, D_n respectively.

We now present the algorithm for weighting the support and confidence of association rules.

Let D be the given database, D^+ the incremental data set, *supp* and *conf* the support and confidence functions of rules in D, $supp^+$ and $conf^+$ the support and confidence functions of rules in D^+, *minsupp*, *minconf*, *mincruc*: threshold values given by user, where *mincruc* ($< Min\{minsupp, minconf\}$) is the crucial value that an infrequent itemset can become frequent itemset in a system.

Procedure 1. *Weighting*

Input: D^+: *database; minsupp, minconf, mincruc: threshold values; R: rule set; CS, CS′: sets of itemsets;*

Output: $X \to Y$: *rule; CS, CS′: sets of itemsets;*

```
(1) input $w_1 \leftarrow$ the weight of $D$;
    input $w_2 \leftarrow$ the weight of $D^+$;
    let $RR \leftarrow R$; $R \leftarrow \emptyset$; $temp \leftarrow \emptyset$;
    let $Itemset \leftarrow$ all itemsets in $D^+$;
    let $CS_{D^+} \leftarrow$ all frequent itemsets in $D^+$;
    let $i \leftarrow i + 1$;
(2) for any $X \to Y \in RR$ do
        begin
          let $supp(X \cup Y) \leftarrow w_1 * supp(X \cup Y) + w_2 * supp(X^+ \cup Y^+)$;
          let $conf(X \to Y) \leftarrow w_1 * conf(X \to Y) + w_2 * conf^+(X \to Y)$;
          if $supp \geq minsupp$ and $conf \geq minconf$ then
            begin
              let $R \leftarrow$ rule $X \to Y$;
              output $X \to Y$ as a valid rule of $i$th mining;
            end;
          else let $temp \leftarrow temp \cup \{X, X \cup Y\}$;
        end;
(3) for any $B \in CS$ do
        begin
          let $supp(B) \leftarrow w_1 * supp(B) + w_2 * supp(B^+)$;
          if $supp(B) \geq minsupp$ then
            for any $A \subset B$ do
              begin
                let $supp(A) \leftarrow w_1 * supp(A) + w_2 * supp(A^+)$;
                let $conf(A \to (B - A)) \leftarrow supp(B)/supp(A)$;
                if $conf(A \to (B - A)) \geq minconf$ then
                  begin
                    let $R \Leftarrow$ rule $A \to (B - A)$;
                    output $A \to (B - A)$ as a valid rule of $i$th mining;
                  end;
                else let $temp \leftarrow temp \cup \{B, A\}$;
              end
        end;
(4) call competing;
(5) return;
```

The procedure *Weighting* generates association rules that are weighted. Here the initialization is done in Step (1). Step (2) performs the weighting operations on rules in RR, where RR is the set of valid rules in the last maintenance. In this Step, all valid rules are appended into R and, the itemsets of all invalid rules weighted is temporarily stored in *temp*. Step (3) extracts all rules from competitive set CS and all invalid itemsets weighted in CS is temporarily stored in *temp*. (Note that any itemset in CS' can generally become as a hopeful itemset and may be appended into CS by competition. However, it cannot become a frequent itemset. In other words, CS' can be ignored when rules are mined.) Step (4) calls procedure *competing* to tackle the competing itemsets for CS and CS', which will be described in next section.

5 Competitive Set Method

As has been shown, the proposed weighted model is efficient to mine trend patterns in incremental databases. To capture the novelty, some infrequent itemsets or new itemsets may be changed into frequent itemsets. We refer to this as *the problem of infrequent itemsets.*

To deal with this problem, we use a competitive model to deal with this problem so as to avoid retracing the whole data set. A *competitive set* CS is used to store all hopeful itemsets, which each itemset in CS can become frequent itemset by *competition.* We now define some operations on CS.

Let D be given database, D^+ the incremental data set to D, A be an itemset, *supp*(A) the support of A in D, *supp*(A^+) the relative support of A in D^+. Firstly, all hopeful itemsets in D is appended into CS, which are defined in Theorem 2.

Secondly, an itemset may become invalid after each mining is done. Such an itemset is appended into CS if its weighted support $\geq$ *mincruc.*

Thirdly, some frequent itemsets in D^+ are appended into CS after each mining if their weighted supports $\geq$ *mincruc.* These itemsets are neither in the set of frequent itemsets, nor in CS. But their supports are pretty high in D^+. This means that their supports in D are unknown. For unknown itemsets, a compromise proposal is reasonable. So we can regard their supports in D as *mincruc*/2. For any such itemset X, $supp_w(X) = w_1 * mincruc/2 + w_2 * supp(X^+)$ according to Weight model. And if $supp_w(X) \geq mincruc$, itemset X is appended into CS. In other words, if

$$supp(A^+) \geq \frac{mincruc(2 - w_1)}{2w_2}$$

in D^+, itemset X is appended into CS; else itemset X is appended into CS' if its weighted support *mincruc*/2, which CS' is an extra competitive set. CS' is used to record another kind of hopeful itemsets. The operations on CS' are similar to those on CS. The main use of CS' is to generate a kind of itemsets with middle supports in D^+. For example, let $mincruc = 0.3$ and $minsupp = 0.6$. Assume the support of an itemset A be less than 0.3 in a given database D, and the supports of A in incremental data sets: D_1, D_2, $\cdots$ D_9 be all 0.64. Because the support of A is less than 0.3 in D, A is not kept in system. Let $w_1 = 0.75$ be the weight of the old database and $w_2 = 0.25$ the weight of the new incremental data set.

According to the operations on CS, $supp_w(A) = w_1 * mincruc/2 + w_2 * supp(A^+) = 0.75 * 0.15 + 0.25 * 0.64 = 0.2725$. This means that itemset A cannot be appended into CS. But the support is greater than $mincruc/2 = 0.15$. From the novelty of data, it can be generated as a frequent itemsets if there are enough incremental data sets. Accordingly, we use CS' to capture this feature of new data. This kind of itemsets such as A can become frequent as follows.

$$\begin{aligned}
supp(A) < 0.3 &\rightarrow 0.15 * 0.75 + 0.64 * 0.25 = 0.2725 \\
&\rightarrow A \textit{ with } supp(A) = 0.2725 \Rightarrow CS' \\
&\rightarrow 0.2725 * 0.75 + 0.64 * 0.25 = 0.364375 \\
&\rightarrow A \textit{ with } supp(A) = 0.2725 \Rightarrow CS \\
&\rightarrow 0.364375 * 0.75 + 0.64 * 0.25 = 0.43328 \\
&\rightarrow 0.43328 * 0.75 + 0.64 * 0.25 = 0.48496 \\
&\rightarrow 0.48496 * 0.75 + 0.64 * 0.25 = 0.52372 \\
&\rightarrow 0.52372 * 0.75 + 0.64 * 0.25 = 0.55279 \\
&\rightarrow 0.55279 * 0.75 + 0.64 * 0.25 = 0.57459 \\
&\rightarrow 0.57459 * 0.75 + 0.64 * 0.25 = 0.590945 \\
&\rightarrow 0.590945 * 0.75 + 0.64 * 0.25 = 0.60321
\end{aligned}$$

Fourthly, some itemsets in CS' are appended into CS after each mining if their weighted supports $\geq mincruc$

Finally, some itemsets are deleted from CS after each mining of association rules is done. By the weighted model, for any $A \in CS$, $supp_w(A) = w_1 * supp(A) + w_2 * supp(A^+)$. If $supp_w(A) < mincruc$, A is deleted from CS; else A is kept in CS with new support $supp_w(A)$.

We now design the algorithm for pattern competition.

Procedure 2. *Competing*

Input: *mincruc: threshold values; temp, Itemset,* CS_{D^+}, CS': *sets of itemsets;* w_1, w_2: *weights;*

Output: CS, CS': *competitive sets;*

```
(1) let $templ \leftarrow \emptyset$; $temp2 \leftarrow \emptyset$;
(2) for $A \in temp$ do
        if $suupp(A) \geq mincruc$ then
            let $templ \leftarrow A$;
        else if $suupp(A) \geq mincruc/2$ then
            let $temp2 \leftarrow A$;
(3) for $A \in CS'$ do
        begin
            let $supp(A) \leftarrow w_1 * supp(A) + w_2 * supp(A^+)$;
        if $suupp(A) \geq mincruc$ then
          let $temp1 \leftarrow A$;
        else if $suupp(A) \geq mincruc/2$ then
          let $temp2 \leftarrow A$;
      end
(4) for $A \in CS_{D^+}$ do
      begin
        let $supp(A) \leftarrow w_1 * mincruc/2 + w_2 * supp(A^+)$;
        if $suupp(A) \geq mincruc$ then
          let $temp1 \leftarrow A$;
        else if $suupp(A) \geq mincruc/2$ then
          let $temp2 \leftarrow A$;
      end
(5) let $CS \leftarrow temp1$; let $CS' \leftarrow temp2$;
(6) return;
```

The procedure *Competing* generates a competitive set for infrequent itemsets. Here the initialization is done in Step (1). Step (2) handles all itemsets in *temp*. And all itemsets with supports in interval $[mincruc, minsupp)$ is appended into CS and the itemsets with supports in interval $[mincruc/2, mincruc)$ is appended into CS'. Step (3) and (4) are as similar as Step (2) to deal in the itemsets in CS' and CS_{D^+}, respectively.

6 Experiments

To evaluate the proposed approach, we have done some experiments using synthetic databases in the Internet. Our experiments shown, this model is efficient and promising. For simplicity, we choose the UCI database BreastCancer to illustrate the effectiveness and efficiency, which contains 699 records. For maintenance, the set of the first 499 records is taken as the initial data set. And the set of each next 50 records is viewed as an incremental new data set. There are four incremental new data sets. And they will be appended into the database one by one. It needs to maintain the association rules once a new data set is appended into. The parameters of experiment databases is summarized as follows.

Table 7: The Experiment Databases

	Record number	Attribute number
Old Database	499	10
New Database 1	50	10
New Database 2	50	10
New Database 3	50	10
New Database 4	50	10

Comparing Running Time of Algorithms

We compare the large item set mining time with the Apriori and *FUP*. Undoubtedly the Apriori will spent the most time cost because it need scan for the candidate items in the old plus new database. The FUP model gets a good improvement. It scan the candidate items in the new database, it need scan in the old database only when the item in old large item set but not in new item set, or in new item set but not in old item set. But our algorithm only need to scan in the new database, so it spent the least time cost. Same conclusion shown in our experiment as in Figure 1.

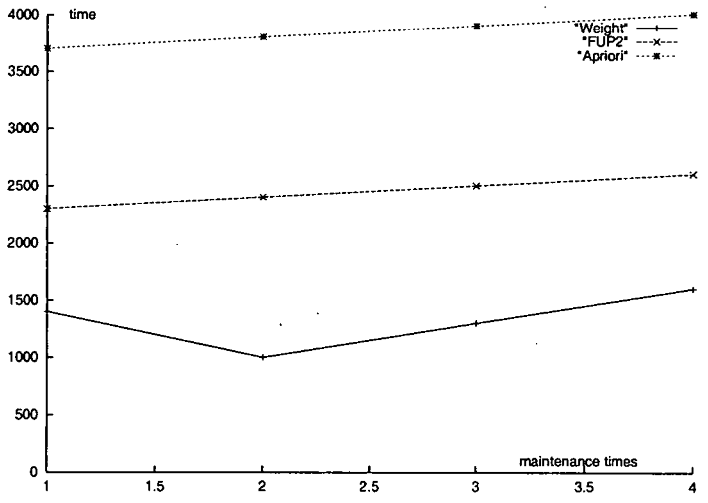

Figure 1: The large item set mining time cost comparison

Our algorithm gets a significant improvement by only scanning the new database. But there are some different rules in our rule set to the rule set made by Apriori which scan all old and new data. But on earth what is the difference, how about its influence? Because Cheung's *FUP* algorithm also scans the old database, which saves some cost by reducing the candidate number in old database, so it generate the same large itemsets as Apriori. Then they will generate same rules according certain confidence. So we only need to compare the result rule set with one of them after generate Large itemset. But our algorithm don't scan the old data but only the new database, then generates the Competitive Rule Set by weighting and choose the winners as result (There is some similarity like genetic algorithm). so it can consist with the new data better than their algorithms. According the result rule sets from large itemset 1 to 4, we compared the difference between our algorithm and Cheung's *FUP* model.

Comparison with FUP

Let $minsupp = 0.2$, $minconf = 0.7$, the confidence threshold $mincruc = 0.25$, the weight of old and new rule confidence is 0.7 and 0.3 respectively, results shown as follows.

Table 8: The rules in the maintenance

	After 1st maintenance	After 2nd maintenance	After 3rd maintenance	After 4th maintenance
FUP_2	1062	1096	1130	1189
Weight algorithm	1095	1185	1204	1397
Both in $A\&W$	1044	1089	1114	1160
Only in Apriori	18	7	16	29
Only in our Weight algorithm	51	96	90	237

All the rules have two supports and two confidences generated relatively by the two algorithms, which $F_{confidence}$ and $F_{support}$ are for FUP algorithm, $W_{confidence}$ and $W_{support}$ are for weight algorithm. If the rule not generated by an algorithm, we let its confidence and support equal zero. We define $error = |W_{support} - F_{support}|$ to measure the difference between two confidence. For those results generated by both FUP and weight algorithm, the comparison shown as follows.

Table 9: The comparison with generated results

	After 1st maintenance	After 2nd maintenance	After 3rd maintenance	After 4 th maintenance
Average error	0.015	0.031	0.032	0.02
Max error in a rule	0.296	0.2	0.085	0.252
Rule number with error over 0.1	1	1	0	28

We can see that they are almost equal at the first 3 maintenance, at most one rule with error over 0.1. But at the 4th maintenance, there are 28 rules with error over 0.1, because some rules from new database with significant different confidence begin influence the old rule set by the Competitive Set.

Now we analysis those different rules: (1) To those rules generated by our algorithm but not FUP, they are the new rules can not be found by FUP. They keep more consistency with new database. For example, this is a rule with confidence only 0.4 before maintenance, but in the new database it is a significance rule, always with confidence over 0.9. In FUP, because the new databases are relatively small than the old database, so the significant association in new data still can not be shown in the all data. During maintenance, it got a confidence serials: 0.4, 0.44, 0.46,0.49, 0.55. But according our algorithm, the confidence change in the maintenance should be:

$$\begin{aligned}0.4 &\rightarrow 0.4 * 0.7 + 0.9 * 0.3 = 0.45\\ &\rightarrow 0.45 * 0.7 + 0.9 * 0.3 = 0.585\\ &\rightarrow 0.585 * 0.7 + 0.9 * 0.3 = 0.66\\ &\rightarrow 0.66 * 0.9 + 0.3 = 0.732\end{aligned}$$

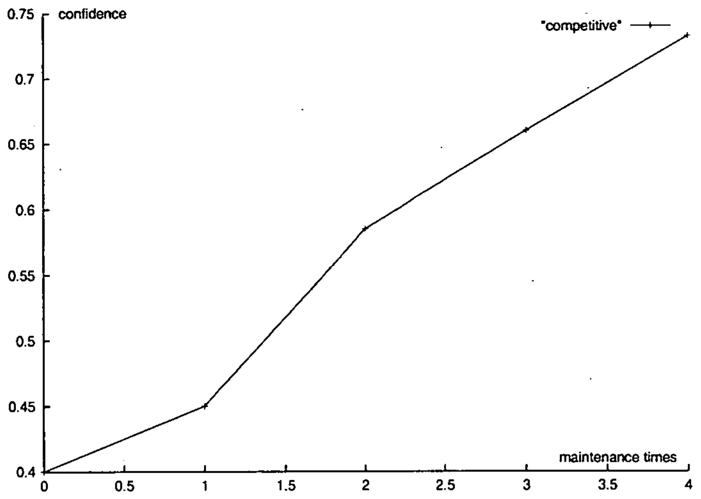

Figure 2: The competitive procedure of a rule

(2) To those rules generated by *FUP* algorithm but not our algorithm, they are the "lost" rules from *FUP*. It needs to know why they lost, how about the influence when lost those rules?

So we analysis all those "lost" rules, and found that, for every rule, at least one of it's $F_{support}$ and $F_{confidence}$ near the relative thresholds. We define the neighbor of *minsupp* and the neighbor of *minconf* as $S_{interval}$ and $C_{interval}$ respectively. There are four classes of lost rules as follows.

Class1: Rules with $F_{confidence}$ in $C_{interval}$ and $F_{support}$ in $S_{interval}$;

Class2: Rules with $F_{confidence}$ in $C_{interval}$ and $F_{support}$ not in $S_{interval}$;

Class3: Rules with $F_{confidence}$ not in $C_{interval}$ and $F_{support}$ in $S_{interval}$;

Class4: Rules with $F_{confidence}$ not in $C_{interval}$ and $F_{support}$ not in $S_{interval}$.

The following figure shown the rule distribution:

Table 10: The rule distribution

	The 1st maintenance	After 2nd maintenance	The 3rd maintenance	The 4th maintenance
Class1	5.5%	0	0	0
Class2	66.5%	0	19%	3%
Class3	27.8%	100%	81%	97%
Class4	0	0	0	0

As we know, those rules with support in $S_{interval}$ or confidence in $C_{intervals}$ usually mean the uncertain and debated knowledge. We often discard those rules in practical decision, so we can say that those "lost" rule only generate some neglectable influence. In order to capture the novelty, this error is certainly reasonable and necessary.

We have seen, our algorithm can save much time cost in association rule maintenance. It keeps more association with the new data than old data, especially in a time serial of continually maintenance. It can generate many new rules to describe the new association in new data. At same time, it discards some old rules unfitting the new data. Their confidence or support near the threshold, so deleting them will only generate a slight influence to the final decision. At a word, our algorithm is efficient in association maintenance, especially suit the practice company decisions which pay more attention to the new market trend, need to a time serial support analysis.

7 Conclusions

Database mining generally presupposes that the goal pattern to be learned is stable over time. This means that its pattern description does not change while learning proceeds. In real-world applications, however, pattern drift is a natural phenomenon which must be accounted for by the mining model. To capture more properties of new data, we advocated a new model of mining association rules in incremental databases in this paper. Our concept of mining association rules in this paper is different from previously proposed ones. It is based entirely on the idea of weighted methods; the main feature of our model is that it reflects the novelty of dynamic data and the size of the given database. Actually, previous frequency-based models are the special cases of our method working without respect to the novelty. Our experiments shown, the proposed model is efficient and promising.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, Mining association rules between sets of items in large databases. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1993: 207-216.

[2] S. Brin, R. Motwani and C. Silverstein, Beyond Market Baskets: Generalizing Association Rules to Correlations. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997: 265-276.

[3] M. Chen, J. Han and P. Yu, Data Mining: An Overview from a Database Perspective, *IEEE Trans. Knowledge and Data Eng.*, vol. 8, 6(1996): 866-881.

[4] D. Cheung, J. Han, V. Ng and C. Wong, Maintenance of discovered association rules in large databases: An incremental updating technique, *Proceedings of 12nd International Conference on Data Engineering,* New Orleans, Louisiana, *1996: 106-114.*

[5] D. Cheung, S. Lee and B. Kao, A gerneral incremental technique for maintaining discovered association rules, *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications,* Melbourne, Australia, 1997.4: 185-194.

[6] R. Godin and R. Missaoui, An incremental concept formation approach for learning from databases. *Theoretical Computer Science*, 133(1994): 387-419.

[7] J. Han, J. Pei, and Y. Yin, Mining frequent patterns without candidate generation. In: *Proceedings of ACM SIGMOD*, 2000: 1-12.

[8] C. Hidber, Online Association rule mining, In: *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1999.

[9] H. Liu and H. Motoda, *Instance Selection and Construction for Data Mining.* Kluwer Academic Publishers, Feburary 2001.

[10] G. Piatetsky-Shapiro, Discovery, analysis, and presentation of strong rules. In: *Knowledge discovery in Databases*, G. Piatetsky-Shapiro and W. Frawley (Eds.), AAAI Press/MIT Press, 1991: 229-248.

[11] T. Shintani and M. Kitsuregawa, Parallel mining algorithms for generalized association rules with classification hierarchy. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1998: 25-36.

[12] R. Srikant and R. Agrawal, Mining generalized association rules. *Future Generation Computer Systems*, Vol. 13, 1997: 161-180.

[13] P. Utgoff, Incremental induction of desion trees. *Machine Learning*, 4(1989): 161-186.

[14] D. Tsur, J. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov and A. Rosenthal, Query flocks: A generalization of association-rule mining. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1998: 1-12.

[15] G. Webb, Efficient search for association rules. In: *Proceedings of ACM SIGKDD*, 2000: 99-107.

[16] X. Wu, Building Intelligent Learning Database Systems, *AI Magazine*, 21(2000), 3: 59-65.

[17] S. Zhang and C. Zhang, Estimating Itemsets of Interest by Sampling. In: *Proceedings of the 10th IEEE International Conference on Fuzzy Systems*, Dec. 2001.

[18] S. Zhang and C. Zhang, Pattern discovery in probabilistic databases. In: *Proceedings of AI'01*, Dec. 2001.

[19] Chengqi Zhang and Shichao Zhang, *Association Rules Mining: Models and Algorithms*. Springer-Verlag Publishers in Lecture Notes on Computer Science, Volume 2307, p. 243, 2002.

Received December, 2001

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János