An Efficient Computation of Convex Closure on Abstract Events

by

Dwight Samuel Bedasse

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2004

Abstract

The behaviour of distributed applications can be modeled as the occurrence of events and how these events relate to each other. Event data collected according to this event model can be visualized using process-time diagrams that are constructed from a collection of traces and events. One of the main characteristics of a distributed system is the large number of events that are involved, especially in practical situations. This large number of events, and hence large process-time diagrams, make distributed-system observation difficult for the user. However, event-predicate detection, a search mechanism able to detect and locate arbitrary predicates within a process-time diagram or event collection, can help the user to make sense of this large amount of data.

Ping Xie used the convex-abstract event concept, developed by Thomas Kunz, to search for hierarchical event predicates. However, his algorithm for computing convex closure to construct compound events, and especially hierarchical compound events (i.e., compound events that contain other compound events), is inefficient. In one case it took, on average, close to four hours to search the collection of event data for a specific hierarchical event predicate. In another case, it took nearly one hour.

This dissertation discusses an efficient algorithm, an extension of Ping Xie's algorithm, that employs a caching scheme to build compound and hierarchical compound events based on matched sub-patterns. In both cases cited above, the new execution times were reduced by over 94%. They now take, on average, less than four minutes.

Acknowledgements

I would like to thank my advisor Dr. Paul A.S. Ward for his wisdom and limitless patience in helping me to understand my research topic. Furthermore, without his support and guidance, from start to finish, this thesis would not have been a reality.

I'd like to let my colleagues in the Shoshin Research Group, at the University of Waterloo, know how much I appreciate them for always being there whenever I needed technical support.

I am grateful to the International Council for Canadian Studies for providing the funding necessary to complete graduate school.

Finally, I'd like to thank my wife Andrea for all her love, emotional support and encouragement. Without her, my research work would have been much more difficult.

Table of Contents

Chapter 1 Introduction 1

1.1 Motivations and Contribution 1

1.2 Organization 2

Chapter 2 Event Models and Event-Predicate Detection 3

2.1 Understanding Event Models 3

2.1.1 Causality Relations 3

2.1.2 Fidge/Mattern (F/M) Vector Timestamp 4

2.2 Event-Predicate Detection 6

2.2.1 Event Predicates 6

2.2.2 Event-Predicate Language 7

2.2.3 Predicate-Detection Algorithm 8

2.3 Constructing Convex Events 10

2.3.1 Representing Convex Events 10

2.4 Convex Closure Computation 12

2.4.1 The Definitions 12

2.4.2 Ping Xie's Algorithm[7] 13

2.4.3 Algorithm for Dealing with Compound Events 17

2.4.4 Pruning Technique 19

2.4.5 An Overview of Ping Xie's Implementation 21

Chapter 3 Experiments and Results of Two Hierarchical Event-Predicates 26

3.1 Experiment 1 27

3.1.1 First Search Result 27

3.2 Experiment 2 29

3.2.1 First Search Result 29

3.3 Comparison of Results from Experiments 1 and 2 30

Chapter 4 Proposed Algorithm 31

4.1 The Approach 31

4.1.1 The Basic Idea 31

4.1.2 Our Algorithm 34

4.2 Results from Experiments 1 and 2 using the Proposed Algorithm 38

4.2.1 Experiment 1 Analysis 39

4.2.2 Experiment 2 Analysis 39

Chapter 5 Conclusion and Future Work 41

5.1 Conclusion 41
5.2 Future Work 41
Appendix A First Search Result of Ping Xie's Algorithm in Experiment 1 42
Appendix B First Search Result of Ping Xie's Algorithm in Experiment 2 47
Appendix C First Search Result of Proposed Algorithm in Experiment 1 52
Appendix D First Search Result of Proposed Algorithm in Experiment 2 57
Appendix E Cache Performance Results of Proposed Algorithm in Experiment 1 62
Appendix F Cache Performance Results of Proposed Algorithm in Experiment 2 64
BIBLIOGRAPHY 66

List of Figures

Figure 1.1: A potential event pattern that could be useful in debugging a distributed program 2

Figure 2.1: An example of an event collection with F/M timestamp 4

Figure 2.2: The event-predicate language 9

Figure 2.3: ASCII equivalence of operators that are defined in the event-predicate language 9

Figure 2.4: Producer-consumer race problem 10

Figure 2.5: An example of a parse tree representing a hierarchical event predicate 11

Figure 2.6: An example of a convex event, *E* 12

Figure 2.7: An event collection that is used for an example of a convex-closure computation 14

Figure 2.8: Computing the back vector of Figure 2.7, given $E = \{e_{1,3}, e_{3,7}\}$ 15

Figure 2.9: Computing the front vector of Figure 2.7, given $E = \{e_{1,3}, e_{3,7}\}$ 18

Figure 2.10: An example of computing the convex closure of two convex events 19

Figure 2.11: An example of pruning variables' domain for $A \| (a \rightarrow b)$ 20

Figure 2.12: A parse tree representing $A \| (a \rightarrow b)$ 21

Figure 2.13: Step 1 in creating an event database 23

Figure 2.14: Step 2 in creating an event database 23

Figure 2.15: Step 3 in creating an event database 24

Figure 2.16: A process-time diagram in Eclipse 24

Figure 2.17: Pattern selection dialog 25

Figure 2.18: The first matched events for Pattern4 25

Figure 3.1: Test environment: software and hardware 27

Figure 3.2: An example of a process-time diagram for $b \| (c \rightarrow a)$ 28

Figure 4.1: An illustration of our proposed-algorithm approach 32

Figure 4.2: An example of a cached-closure violation 34

List of Tables

Table 3:1: Comparison of Results from Experiments 1 and 2 30
Table 4:1: Cache Performance Results from Experiment 1 38
Table 4:2: Cache Performance Results from Experiment 2 39

Chapter 1
Introduction

1.1 Motivations and Contribution

Understanding the behaviour of a distributed application is often difficult because of the large amount of event data that is collected and the complexity of how these events relate to each other across many concurrently-executing processes. Events are occurrences of interest that help a user trying to understand the behaviour of the application.

One approach to understanding such behaviour is to collect event information according to an event model and visualize this information using a process-time diagram. This diagram shows the partial-order relationship between events. Events could be remote procedure calls (RPCs), alarms, communication with remote processes, etc.

In Figure 1.1, the events surrounded by the dashed lines depict an event pattern that represents events on processes A and C both happening before an event on process B, but with no synchronization between them. If the collection of event data is small, as shown in Figure 1.1, this pattern could be located by inspection. However, in a large collection of event data, locating event patterns by inspection is not feasible. As a result, a search mechanism is needed to help alleviate this problem.

The event pattern previously cited is an example of a simple pattern because it is made up of relationships between primitive events. However, there are complex patterns that are made up of relationships between compound events (sets of primitive events). Apart from locating patterns of interest, the events matching a pattern, especially a complex pattern, can be replaced by an abstract event. An abstract event is a compound event that satisfies a pattern of interest and can be displayed in a process-time diagram. This is especially useful if multiple occurrences of that pattern exist.

Ping Xie [15] developed an algorithm that helps to search for simple and complex patterns. However, his algorithm is inefficient when searching for complex patterns. This dissertation presents an efficient algorithm that, in some cases, reduces the execution time of Ping Xie's algorithm by over 94%: two different cases that searched for complex patterns took, on average, nearly four hours and close to one hour, respectively, to find their first set of matched events. Using our method, the respective pattern matches take less than four minutes to find.

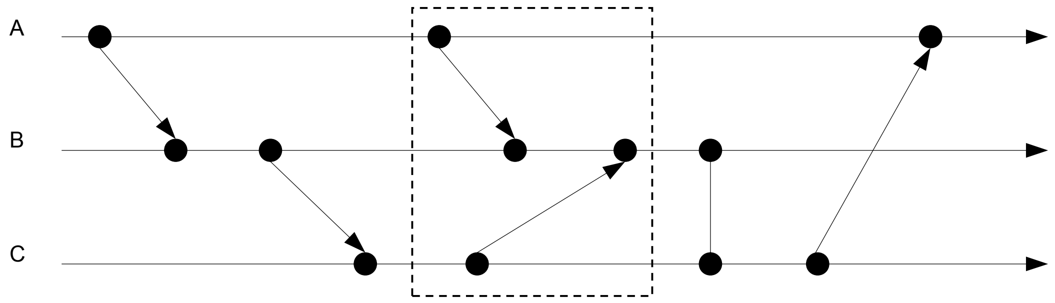


Figure 1.1: A potential event pattern that could be useful in debugging a distributed program

1.2 Organization

Chapter 2 describes the event model of a distributed computation, and details of the search mechanism developed by Ping Xie. Chapter 3 explains Ping Xie's convex-closure algorithm and describes a series of experiments that were performed to analyze the operation of his algorithm. Chapter 4 presents our improved algorithm and results from its implementation, including a performance analysis. Chapter 5 concludes the thesis and discusses possible future work.

Chapter 2
Event Models and Event-Predicate Detection

2.1 Understanding Event Models

An event model is a model of distributed computation in which the computation is decomposed into sequential entities, such as processes and threads, communicating via message passing. Events on those sequential entities are sends, receives, and anything else of interest. Figure 2.1 is an example of a process-time diagram that shows event information collected from the execution of a distributed application. The information is collected in accordance with the event model described. This diagram contains three processes P_1, P_2, and P_3, and five events *b*, *d*, *e*, *f*, and *u*. Time increases from left to right.

In this thesis, only four types of primitive events are considered: unary, synchronous, asynchronous send, and asynchronous receive. A unary event is an event that does not involve any process interaction. An example of a unary event is u in Figure 2.1. A synchronous event is a single event that is imagined to occur simultaneously on two processes. This type of event is used to model synchronous communication between processes. An example of a synchronous event in Figure 2.1 is b, the two halves of which are connected by a vertical, undirected line. Asynchronous send and receive events are used to model asynchronous communication as initiating on one process and terminating on another, respectively. This type of communication is represented by a slanted and directed line connecting both processes. In Figure 2.1, e is an asynchronous-send event and f is the corresponding asynchronous-receive event.

2.1.1 Causality Relations

Understanding causality is fundamental to understanding the behaviour of distributed applications [3]. Given the ordering within processes and the communication between them, the causality relationship between any two primitive events can be determined using the happened-before and concurrent relations introduced by Lamport [10].

Definition 1: ***Happened-Before***

The happened-before relation, denoted $\rightarrow$, on the set of events of a computation is the smallest relation satisfying the following three conditions:

1. if *e* and *f* are events in the same process, and *e* occurs before *f*, then $e \rightarrow f$.
2. if *e* is the sending of a message by one process and *f* is the receipt of that message by another process, then $e \rightarrow f$.
3.

[1]In this thesis, only communication involving exactly two processes is discussed and for purposes of formal description the "one event" view of synchronous communication is adopted.

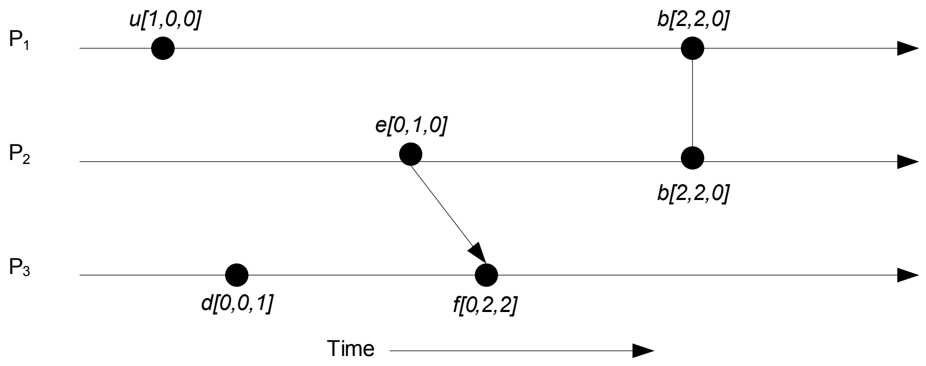


Figure 2.1: An example of an event collection with F/M timestamp

4. if $e \rightarrow f$ and $f \rightarrow g$ then $e \rightarrow g$.

Definition 2: ***Concurrent***

Two distinct events e and f are said to be concurrent if $\neg(e \rightarrow f)$ and $\neg(f \rightarrow e)$. This is denoted as $e \| f$.

2.1.2 Fidge/Mattern (F/M) Vector Timestamp

Based on Lamport's work [10], Fidge and Mattern [6, 7, 11][2] developed the concept of a partially ordered logical clock, and a corresponding algorithm that timestamps events using integer vectors. A comparison of the vector timestamps of any two events allows the causality relation between them to be determined in constant time.

The timestamp algorithm assigns each primitive event e a vector timestamp T_e. T_e is an integer vector of size N, where N is the number of processes in the computation. The algorithm is as follows:

1. Each process P_i in the computation maintains a vector C_i of size N. Each C_i is initialized to a zero vector at the beginning of P_i.
2. Whenever process P_i performs a unary event e:

$$\begin{aligned} C_i[i] &= C_i[i] + 1 \\ T_e &= C_i \end{aligned}$$

3. If e is an asynchronous send event, its timestamp is calculated in a similar way:

$$\begin{aligned} C_i[i] &= C_i[i] + 1 \\ T_e &= C_i \end{aligned}$$

[2]The concept of vector timestamps was developed independently by Fidge and Mattern.

In addition, the outgoing message carries a copy of the updated C_i, denoted as $C_i^{'}$ hereafter.

4. If f is an asynchronous receive event on process P_j corresponding to a send event e on process P_i, C_j and T_f are calculated as follows:

$$\begin{aligned} C_i^{'}[i] &= C_i^{'}[i] + 1 \\ C_j[j] &= C_j[j] + 1 \\ \forall p \in \{1, \ldots, n\}, C_j[p] &= max(C_j[p], C_i^{'}[p]) \\ T_f &= C_j \end{aligned}$$

5. If event e is a synchronous event on P_i and P_j, then e's timestamp is calculated as follows:

$$\begin{aligned} C_i[i] &= C_i[i] + 1 \\ C_j[j] &= C_j[j] + 1 \\ \forall p \in \{1, \ldots, n\}, T_e[p] &= max(C_j[p], C_i[p]) \end{aligned}$$

Furthermore, local vector clocks on P_i and P_j are updated respectively as follows:

$$\begin{aligned} \forall p \in \{1, \ldots, n\}, C_i[p] &= T_e[p] \\ \forall p \in \{1, \ldots, n\}, C_j[p] &= T_e[p] \\ C_i[j] &= C_i[j] + 1 \\ C_j[i] &= C_j[i] + 1 \end{aligned}$$

Applying the algorithm for events u, e, and f (from Figure 2.1) are as follows:

Event d: Applying rule 1, P_3's clock C_3 is initialized to [0,0,0] ;

Applying rule 2 to event d, a **unary event**, results in the following:

$$\begin{aligned} C_3[3] &= C_3[3] + 1 \\ T_d &= C_3 = [0,0,1] \end{aligned}$$

Event e: Applying rule 1, P_2's clock C_2 is initialized to [0,0,0];

Since e is an **asynchronous send event**, we apply rule 3, which is the same as rule 2, to get e's timestamp:

$$T_e = [0,1,0]$$

Event *f*: Since *f* is the **asynchronous receive event** from *e*, we apply rule 4 as follows:

$$\begin{aligned}
C_2'[2] &= C_2'[2] + 1 &&= 2 \\
C_3[3] &= C_3[3] + 1 &&= 2 \\
\forall p \in \{1, \ldots, 3\}, C_3[p] &= max(C_3[p], C_2'[p]) && \\
T_f &= C_3 &&= [0,2,2]
\end{aligned}$$

Cheung [3] proved that the following theorem can be used to deduce the happened-before relation between any two events *e* and *f*:

Theorem 1: ***Precedence Test***

$e \rightarrow f \iff T_e[p] < T_f[p]$, where event *e* occurs on process p[3].

Based on Theorem 1, if $T_d[3] < T_e[3]$, $d \rightarrow e$. However, this condition is false as seen from their timestamps above. Hence, event *d* did not happen-before event *e*. Similarly, event *e* did not happened-before event *d* and thus, based on Definition 2, *d* and *e* are concurrent.

2.2 Event-Predicate Detection

Event-predicate detection is a search mechanism that locates events that satisfy an event pattern of interest to the user. As such it consists of the following three parts:

1. a formalization of event predicates
2. a predicate language
3. a predicate-detection algorithm.

2.2.1 Event Predicates

The user may be interested in a pattern that is made up of a single primitive event *a* or a related pair of primitive events $a \rightarrow b$. The mechanism involved in finding a single primitive event is trivial, i.e., either it exists or it does not and therefore, can be mapped to true/false (T/F). Similarly, any related primitive event pair $a \rightarrow b$ or $a \parallel b$ can also be mapped to T/F based on Theorem 1. However, locating event patterns that are made up of related pairs of compound events (hierarchical event predicates), for example $(a \rightarrow b) \parallel (c \rightarrow d)$, presents a challenge:

$$a \rightarrow b \text{ maps to T/F}$$

therefore,

$$(a \rightarrow b) \parallel (c \rightarrow d) \text{ maps to T/F} \parallel \text{T/F}$$

which is undefined.

[3]If *e* is a synchronous event, either of the two processes in which *e* occurs can be chosen.

Nevertheless, (a→b) $\|$ (c→d) can be evaluated in terms of the logical expression

$$a \rightarrow b \;\wedge\; c \rightarrow d \;\wedge\; a \,\|\, c \;\wedge\; a \,\|\, d \;\wedge\; b \,\|\, c \;\wedge\; b \,\|\, d$$

but this method is cumbersome and does not scale. On the other hand, instead of mapping $a \rightarrow b$ to its usual T/F status, it could be mapped to a compound event. As a result, we could write $(a \rightarrow b) \,\|\, (c \rightarrow d)$ provided there is a definition for the causality relation between compound events.

Two possible definitions, as observed by Kunz [9], for the precedence relation between two compound events E and F are

1. $E \rightarrow F$ if and only if $\forall e \in E$ and $\forall f \in F : e \rightarrow f$,

and

2. $E \rightarrow F$ if and only if $\exists e \in E$ and $\exists f \in F : e \rightarrow f$.

Based on research on event abstraction [3, 8, 9, 12], the first definition is too restrictive because compound events that are clearly related become concurrent with each other.

Cheung [3] chose the second definition and observed that its relation is not a partial-order relation between arbitrary compound events. As a result, he proposed a class of compound events, called contractions, to preserve the partial-order relation between them. However, Summers [12] illustrated a system of contractions for which the transitive property does not hold and proposed a slightly more restrictive definition of contractions to preserve the transitive property.

Kunz [9] proposed another class of compound events called convex events. Contrary to Cheung's and Summers' work on event abstraction [8, 9], Kunz [9] decided not to preserve the partial-order properties of the second definition that deals with the precedence relation between compound events. Instead, he chose to preserve the atomicity property of compound events—convex events do not have outside interference. However, a set of events that is not convex does have outside interference and does not appear, intuitively, to be very useful [16]. Therefore, this kind of event set is replaced by its convex closure. For a pair of event sets, the convex closures have the same precedence relationship as the original sets. A pair of convex events requires only two vector timestamps to correctly and efficiently deduce the precedence relation, based on the second definition, for the convex events, i.e., one vector timestamp per convex event. Ping Xie [15] chose the second definition and used convex events to build hierarchical event predicates.

A convex event is defined as follows:

Definition 3: Convexity of Event Sets

A set of events E is convex if and only if $\forall x, y \in E : x \rightarrow z \wedge z \rightarrow y \Rightarrow z \in E$.

Events x, y, and z are not necessarily primitive events. However, the convexity property must hold for all constituent events of a compound event.

2.2.2 Event-Predicate Language

A predicate language is made up of variables, logical connectives, and predicates. In the context of this thesis, variables are used to represent only primitive events and compound events [16]. Predicates involving the happened-before and concurrent relations play an important role in the expressiveness of the language. These predicates help to support the expression of hierarchical event predicates, such as $(a \rightarrow b) \| (c \rightarrow d)$ that is made up of four primitive-event predicates a, b, c, and d, and the event-predicate operators $\|$ and $\rightarrow$. This example will be revisited in the next section.

This thesis adopts Ping Xie's event-predicate language [15]. However, the syntax of his predicate language was adapted from Jaekl's work [8]. Jaekl showed that his event-predicate language was sufficient for expressing a number of problem classes, such as finding subroutines, identifying bottlenecks, and detecting symmetry in communication. Figure 2.2 lists the production rules used to recognize event predicates.

Ping Xie et al. used the *base_comp* production rule to define a primitive event predicate as a tuple [*trace, type, text*], *trace* is the textual name of the trace on which a matching primitive event must occur. *Type* specifies the event type (for example, thread-end or RPC-call) that the matching event must belong to. *Text* is any extra information that must be attached to the event. In addition, flexibility is added to both string fields, *trace* and *text*, in that, they support regular expression matching.

Expression in the language is strengthened with the addition of the limited operator because there is no negation operator. It represents the following event predicate [16]:

Two events, e and f, are said to satisfy the limited operator $a \xrightarrow{b} c$, if and only if $e \rightarrow f$, e and f satisfy a and c respectively, and there does not exist a third event g such that $e \rightarrow g$, $g \rightarrow f$, and g satisfies b.

The limited operator $a \xrightarrow{b} c$ provides a degree of control that enables the user to detect the presence of simple communication-symmetry problems quite easily [8]. For example[4], Figure 2.4 shows the execution of a producer and a consumer communicating via a bounded buffer of size one. Each message is designated: *sw = send write, rw = receive write, sr = send request, rr = receive request, sd = send data, and rd = receive data.* The extraneous write, surrounded by the dashed line, indicates a synchronization problem and causes the buffer to overflow. This buffer-overflow problem can be detected using the predicate $[B, rw,] \xrightarrow{[B, sd,]} [B, rw,]$ (see Figure 2.4).

2.2.3 Predicate-Detection Algorithm

Algorithms used to locate events that satisfy event patterns can be grouped into two classes [16]: online detection algorithms and offline detection algorithms. An online detection algorithm locates events against predicates as the events are collected during execution. An offline detection algorithm locates events against predicates after they have been collected from an execution of a target application. This thesis discusses only offline predicate detection[5]. Ping Xie's [15] event-predicate detection approach is based on the naive backtracking (BT) algorithm. Figure 2.5 shows the parse tree corresponding to the hierarchical event predicate $(a \rightarrow b) \| (c \rightarrow d)$[6].

[4]Taken from Jaekl's thesis.

[5]Ping Xie's [15] event-predicate detection system uses an offline event-predicate algorithm.

[6]Example was taken from Ping Xie's [15] thesis.

$$
\begin{array}{rcl}
\textit{predicates} & \Rightarrow & ((\textit{predicate} \mid \textit{comp\_name})";")^{*} \\
\textit{predicate} & \Rightarrow & \langle ID \rangle ":=" \textit{clause} \\
\textit{clause} & \Rightarrow & \textit{term}(\textit{operator term})^{*} \\
\textit{operator} & \Rightarrow & "-->" \\
 & \mid & "\|" \\
 & \mid & "\xrightarrow{term}" \\
 & \mid & "\wedge" \\
 & \mid & "\vee" \\
\textit{term} & \Rightarrow & \langle ID \rangle \\
 & \mid & \textit{event} \\
 & \mid & "(" \textit{clause} ")" \\
\textit{event} & \Rightarrow & \textit{component}"."\textit{component} \\
 & \mid & \textit{component} \\
\textit{comp\_name} & \Rightarrow & \langle CID \rangle ":=" \textit{component} \\
\textit{component} & \Rightarrow & \textit{Base\_comp} \\
 & \mid & \langle CID \rangle \\
\textit{base\_comp} & \Rightarrow & "[" \textit{process} "," \textit{type} "," \textit{text} "]" \\
\langle ID \rangle & \Rightarrow & \langle ALPHA \rangle (\langle ALNUM \rangle)^{*} \\
\langle CID \rangle & \Rightarrow & \langle ALPHA \rangle (\langle ALNUM \rangle)^{*} \\
\langle ALPHA \rangle & \Rightarrow & ["a"-"z","A"-"Z","\_"] \\
\langle ALNUM \rangle & \Rightarrow & ["a"-"z","A"-"Z","\_","0"-"9"]
\end{array}
$$

Figure 2.2: The event-predicate language

The ASCII equivalence of the operators defined in Figure 2.2 is shown in Figure 2.3.

Formal	ASCII	Meaning
$\rightarrow$	-->	Happened-before operator
$\|$	||	Concurrent operator
$A \xrightarrow{B} C$	A-(B)->C	Limited operator
$\wedge$	AND	Conjunction
$\vee$	OR	Disjunction

Figure 2.3: ASCII equivalence of operators that are defined in the event-predicate language

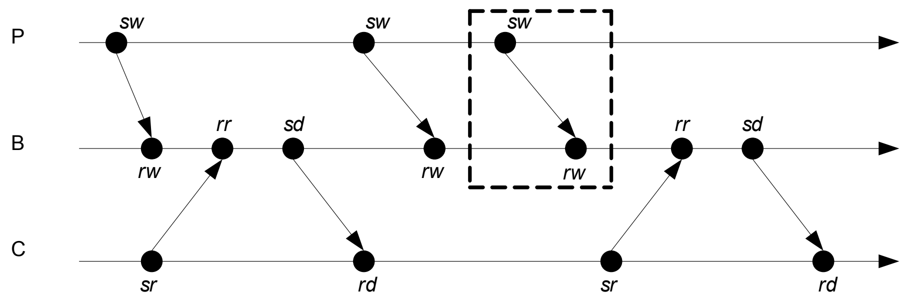


Figure 2.4: Producer-consumer race problem

In Figure 2.5, *a*, *b*, *c*, and *d* are primitive events. *A*, *B*, and *C* are convex events. For example, *A* represents the convex closure of an event set consisting of events *a*, and *b* with *a* preceding *b*. Event pairs that are from either *a*'s and *b*'s domain or *c*'s and *d*'s domain respectively, satisfying $a \rightarrow b$ or $c \rightarrow d$ respectively, may not be convex and thus their convex closures must be computed before any further searching takes place. A domain represents a collection of primitive events that belong to a particular process. Nodes *A* and *B* represent the convex-closures of event sets $a \rightarrow b$ and $c \rightarrow d$, respectively. A precedence test can now be carried out between the convex events, at nodes *A* and *B*, to ascertain whether they are concurrent. If they are concurrent, the algorithm computes another convex closure for them and returns it as a match. Otherwise, the search backtracks to the last leaf node visited, which is *d*'s domain in this example. The next satisfying event in *d*'s domain is selected and predicate $c \rightarrow d$ is evaluated again. If all the events in *d*'s domain have been tested and $c \rightarrow d$ cannot be satisfied, the search further backtracks to *c*'s domain and selects the next satisfying event from it. This new assignment is evaluated against all the satisfying events in *d*'s domain. It is possible that the current convex event at node *A* is not concurrent with any possible convex events at node *B*. If this is the case, the search further backtracks to *b*'s domain and tries to build a new convex event for the events that satisfy $a \rightarrow b$. This backtracking and evaluating process continues until either a match is found or all possible combinations of the events from the domains of predicates *a*, *b*, *c*, and *d* are tested. If the latter case occurs, the hierarchical event predicate is not satisfied.

2.3 Constructing Convex Events

Kunz [9], in an early attempt to construct convex events, developed a limited algorithm that computes the convex closure of a class of compound events that contain only two primitive events that belong to the same process. Ping Xie [15] realized this limitation and proposed a comprehensive algorithm that computes the convex closures for arbitrary compound events and hierarchical compound events. Hierarchical compound events are compound events that contain other compound events.

2.3.1 Representing Convex Events

In Figure 2.6, a convex event is represented as two vectors, a front vector and a back vector, that contain only primitive events. They are defined as follows [9]:

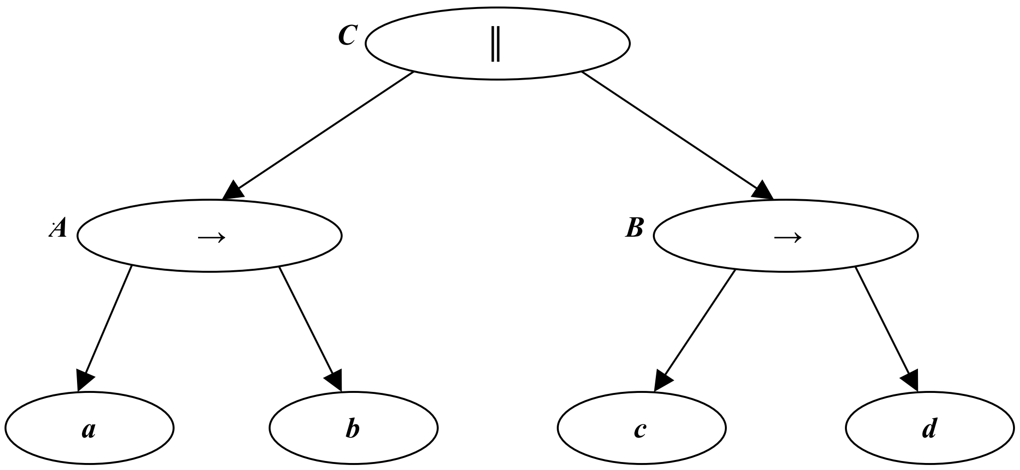


Figure 2.5: An example of a parse tree representing a hierarchical event predicate

Definition 4: ***Front Vector***

Let E be a convex event. The front vector of E is defined as

$$\{e \mid e \in E \wedge \exists p \text{ on which } e \text{ occurs} \wedge \neg\exists f : (f \in E \wedge f \text{ occurs on } p \wedge f \rightarrow e)\}$$

Definition 5: ***Back Vector***

Let E be a convex event. The back vector of E is defined as

$$\{e \mid e \in E \wedge \exists p \text{ on which } e \text{ occurs} \wedge \neg\exists f : (f \in E \wedge f \text{ occurs on } p \wedge e \rightarrow f)\}$$

In addition, we introduce a supplementary definition as follows [9]:

Definition 6: ***Location Set***

The location set of an event set E, $\mathbf{l}E$ is defined as

$\mathbf{l}E = \{p{:}\exists e_p \in E\}$

where p is a process in the computation and the notation e_p indicates event e happened in process p.

Based on Definitions 4 and 5, the front vector and the back vector of a convex event E represent E's boundaries, i.e., the front vector marks E's beginning and the back vector marks E's end.

In Figure 2.6, each primitive event that is a constituent of the front vector is the only event from the process that it occurs on that does not have any predecessors from its process in E. For example, event b is the only event in the front vector that occurs on P_2 and it has no predecessors in E that occur on P_2. The primitive events that make up the back vector have the same characteristics except none of them have any successors from their process in E. For example, event d is the only event in the back vector that occurs on P_1 and it has no successors in E that occur on P_1. Event d has successors e and f, but they do not occur on P_1, and event b has one predecessor, a, that does not occur on P_2.

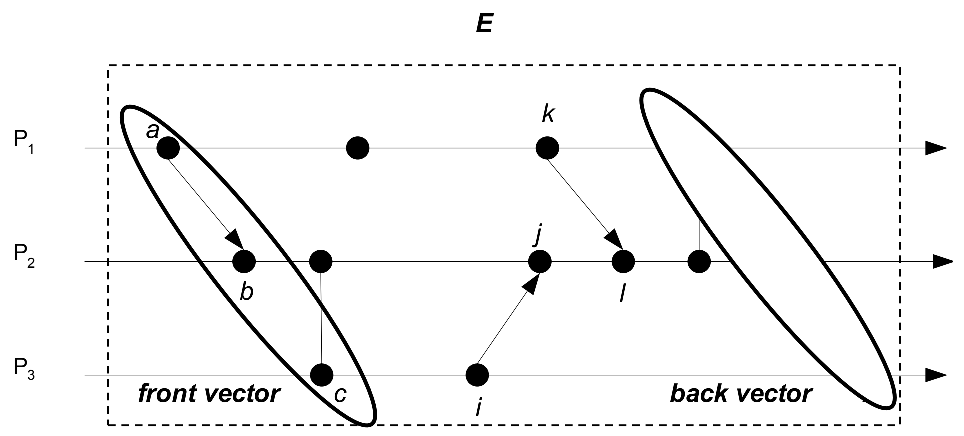


Figure 2.6: An example of a convex event, *E*

2.4 Convex Closure Computation

This section describes Ping Xie's [15] convex-closure computation, including related definitions, computing the back vector and the front vector, complexity analysis of his algorithm, and how his algorithm deals with an event set that includes compound events. In addition, an overview of the implementation of Ping Xie's offline event-predicate detection system is presented.

2.4.1 The Definitions

Several key definitions, involving the relationships between and among primitive events, need to be introduced before the convex-closure algorithm is discussed. The definitions are as follows [15]:

Definition 7: ***Greatest Predecessor***

Let *e* and *f* be two arbitrary primitive events and *P* be a process on which *e* occurs. *e* is *f*'s greatest predecessor on *P*, denoted $gp_P(f) = e$, if and only if $e \rightarrow f$ and there does not exist a third primitive event *g* occurring on *P* such that $e \rightarrow g$ and $g \rightarrow f$.

Definition 8: ***Least Successor***

Let *e* and *f* be two arbitrary primitive events and *P* be a process on which *e* occurs. *e* is *f*'s least successor on *P*, denoted $ls_P(f) = e$, if and only if $f \rightarrow e$ and there does not exist a third primitive event *g* occurring on *P* such that $f \rightarrow g$ and $g \rightarrow e$.

Referring to Figure 2.6, *l's* greatest predecessor on P_1, P_2, and P_3 is *k*, *j*, and *i* respectively. The following theorem [9] makes it possible to locate a primitive event's greatest predecessor on each process in $O(N)$ time, where *N* is the number of processes.

The definition for the Least Predecessor and the Greatest Predecessor is formally defined, for the first time, in this thesis.

Theorem 2: ***Location* of *Greatest Predecessor on each Process***

Let T_e be the F/M timestamp for a primitive event *e*. The index of its greatest predecessor on process P_i is equal to $T_e[P_i]$-1 if $T_e[P_i]$ is greater than 0. Otherwise, *e* has no predecessor on P_i.

Theorem 2 plays an integral role in the convex-closure computation, especially in the construction of the back vector. Unfortunately, a counterpart of Theorem 2 for the least successor does not exist because F/M timestamps do not encode the least-successor information.

Definition 9:

$S_{back}(E) := \{t \in E \mid t \text{ has no successors in } E\}$

where *E* is an arbitrary primitive event set.

Definition 10:

$S_{front}(E) := \{t \in E \mid t \text{ has no predecessors in } E\}$

where *E* is an arbitrary primitive event set.

The next two definitions, especially the latter, define the set of greatest predecessors for an arbitrary primitive event set *E* with the former being a precursor to it.

Definition 11:

Let *E* be an arbitrary primitive event set and *N* be the number of processes in the computation. A primitive event set for *E*, denoted *U*(*E*)*,*is defined as follows:

$$U(E) := S_{back}(E) \cup \{t \mid t = gp_{p_i}, \text{ where } e \in S_{back}(E) \text{ and } i \in [1, N]\}$$

Definition 12:

Let *E* be a primitive event set. The set of *E*'s greatest predecessors on each process, denoted *GP*(*E*), is defined as follows:

$$GP(E) := \begin{Bmatrix} t \in U(E) \mid \text{for at least one process } P \text{ on which } t \text{ occurs,} \\ \neg\exists e \text{ occuring on } P \text{ from } U(E) \text{ with } t \text{ preceding } e \end{Bmatrix}$$

GP(*E*) is obviously a subset of *U*(*E*) and contains only the latest event, if there is more than one event, from the same process in *U*(*E*). In other words, no two events in *GP*(*E*) are from the same process. However, *GP*(*E*) may contain more than one event per process because of synchronous events that necessitate additional events needed in the other process on which they occur.

2.4.2 Ping Xie's Algorithm[7]

In Figure 2.7, $e_{i,j}$ is used to represent the *jth* event on process P_i. The algorithm to compute the convex closure, the front and back vectors, for a primitive event set *E* consists of two steps:

1. locate the back vector of *E*'s convex closure
2. locate the front vector of *E*'s convex closure by checking all the events between the back vector and $S_{front}(E)$

[7]This section was adopted from Ping Xie's thesis [15] except the examples.

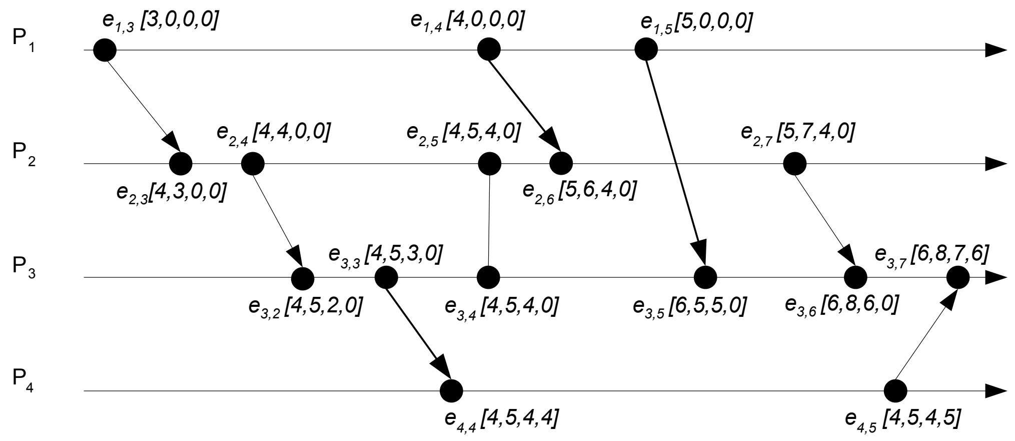


Figure 2.7: An event collection that is used for an example of a convex-closure computation

Algorithm 1 gives the pseudo-code for computing the back vector of the convex-closure of an arbitrary primitive event set *E*.

Algorithm 1

```
1.  function COMPUTE_BACK(E)
2.   $S_{back} := \{t \in E \mid t \text{ has no successors in } E\}$;
3.   $S_{front} := \{t \in E \mid t \text{ has no predecessors in } E\}$;
4.   $Back := S_{back}$;
5.   for each event e in $S_{back}$
6.     $T_e$ := e's F/M vector timestamp;
7.     for $i:=1$ to $N$      /* for N processes */
8.     {
9.       if $T_e[i] = 0$ OR $P_i = P_e$
10.        continue;
11.      $f := e_{i,j}$;          /* $j := T_e[i]-1$ */
12.      if f is not in $S_{front}$ AND $\{\neg\exists g : (g \in S_{front} \wedge g \rightarrow f)\}$
13.        continue;
14.      if there exist an event g from $P_i$ in Back with g preceding f
15.        replace g with f in Back;
16.      else if there does not exist an event from $P_i$ in Back
17.        add f to Back;
18.    }
19.  return Back;
```

Lines 2-4: Line 5:	$S_{back} = \{e_{3,7}\}$, $S_{front} = \{e_{1,3}\}$, and $Back = \{e_{3,7}\}$, respectively. $e = e_{3,7}$			
Lines 5-17:	***i=1***	***i=2***	***i=3***	***i=4***
	$T_e = [6,8,7,6]$ $f=e_{1,5}$ $Back \mathrel{+}= e_{1,5}$	$T_e = [6,8,7,6]$ $f=e_{2,7}$ $Back \mathrel{+}= e_{2,7}$	$T_e = [6,8,7,6]$ *continue*	$T_e = [6,8,7,6]$ $f=e_{4,5}$ $Back \mathrel{+}= e_{4,5}$
Line 19:	$\boldsymbol{Back = \{e_{1,5}, e_{2,7}, e_{3,7}, e_{4,5}\}}$			

Figure 2.8: Computing the back vector of Figure 2.7, given $E = \{e_{1,3}, e_{3,7}\}$

Computing the Back Vector

In Algorithm 1, it is necessary to first locate $S_{back}(E)$ and $S_{front}(E)$ to compute the back vector of E's convex closure. The computation of $S_{back}(E)$ and $S_{front}(E)$ is carried out with the help of F/M vector timestamps. Algorithm 1 does not compute $U(E)$ directly, but computes $GP(E)$ by checking events in $S_{back}(E)$. For each event e in $S_{back}(E)$, its greatest predecessor on each process is located using Theorem 2. If more than one event from the same process is in the set of these greatest predecessors, only the latest event (i.e., one that has no successors from its process in the set) is kept in the set, as required by the definition of $GP(E)$. The others are discarded. $GP(E)$ is obtained after $S_{back}(E)$ is completely checked. The events in GP(E) are then checked against events that are in $S_{front}(E)$. The events in $GP(E)$ that are not part of $S_{front}(E)$ or did not happen after any of the events in $S_{front}(E)$ are removed. Finally, the remaining events in $GP(E)$ are precisely the events of the back vector of E's convex closure.

In Algorithm 1, lines 2 and 3 locate $S_{back}(E)$ and $S_{front}(E)$ respectively. Line 4 initializes the back vector *Back* using $\mathrm{S}_{back}(E)$. Lines 5-17 are a combination of the calculation of $U(E)$, $GP(E)$, and *Back*. Figure 2.8 illustrates how Algorithm 1 works.

Complexity Analysis

Algorithm 1 needs $O(N)$ timestamp computations and $O(N^3)$ timestamp comparisons, where N is the number of processes in the computation.

Line 6 is the only place in Algorithm 1 that a timestamp computation is required and in the worst case $|\mathrm{S}_{back}(E)| = N$, because $\mathrm{S}_{back}(E)$ cannot contain more than one event from a process.

There are four places in Algorithm 1 where timestamp comparisons are required: lines 2, 3, 12, and 14. Lines 2 and 3 require timestamp comparisons to compute S_{back} and S_{front} respectively. The number of timestamp comparisons required to compute S_{back} and S_{front} is $O(E^2)$. The number of timestamp comparisons carried out on lines 12 and 14 is determined by the number of iterations of lines 12 and 14, which is as follows:

$$\mathrm{T} = \left|S_{front}\right| \times \sum_{e \in S_{back}} \left|e'\,s \text{ greatest predecessors}\right|$$

In the worst case where each event e in S_{back} has N greatest predecessors, the equation can be reduced as follows:

$$\mathrm{T} = \left|S_{front}\right| \times \left|S_{back}\right| \times N$$

Note that both S_{front} and S_{back} are of size N in the worst case because neither of them can contain more than one event from a process. Therefore, the worst-case complexity of Algorithm 1 is $O(N^2) + \mathrm{O}(N^3) = O(N^3)$ in terms of timestamp comparisons.

Computing the Front Vector

The computation of the front vector requires the use of locating a primitive event's least successors. Unfortunately, the F/M vector timestamp does not encode least successor information. As a result, the algorithm (Algorithm 2) has to check all events between S_{front} and the back vector, *Back*. This presents an efficiency problem, how to avoid checking an event twice. Ping Xie [15] solved this problem by checking the events in the convex-closure in a predefined order.

Each process in the back vector is selected separately, starting with those that have events in S_{front}. The events from the selected process are checked: starting with the event in S_{front} and ending with the event in *Back*. During the checking, whenever an asynchronous send event or a synchronous event is encountered, if its partner process P' satisfies the following three conditions:

1. P' has not yet been checked,
2. P' has events in the back vector, and
3. there is no event in S_{front} on P',

P' is selected for future checking. The algorithm also maintains a temporary event set, S, for each iteration of the loop over the processes. The partner events are stored in S so that the algorithm knows which event to start with whenever P' is checked. However, before a new process is checked, the algorithm computes S_{front} from S and updates the front vector using S_{front}. Figure 2.9 illustrates how Algorithm 2 works.

Algorithm 2

```
function COMPUTE_FRONT($E$)
 $Back$ := COMPUTE_BACK($E$);
 $S_{front}$ := $\{t \in E \mid t$ has no predecessors in $E\}$;
 $Front$ := $S_{front}$;
 $Scanned$ := {};
 $S$ := {};
 do
   $P$ := $\mathbf{l}S_{front}$;
   for each process $p$ in $P$
   {
     $start$ := the event from $p$ in $S_{front}$;
     $end$ := the event from $p$ in $Back$;
13.     **for** each event $f$ between *start* and *end* inclusive
14.     {
15.       **if** $f$ is neither an asynchronous send event nor a synchronous event
16.         **continue**;
17.       $t := f$'s partner event;
18.       **if** $P_t$ does not have events in *Back* OR $P_t$ is in *Scanned*
19.         **continue**;
20.       **if** there is an event from $P_t$ in $S_{front}$
21.         **continue**;
22.       **if** no event from $P_t$ is in $S$
23.         add $t$ to $S$;
24.       **else if** $t$ precedes the event from $P_t$ in $S$
25.         replace that event with $t$;
26.     }
27.     add $p$ to *Scanned*;
28.   }
29.   add events in $S$ to *Front*;
30.   $S_{front} := \{t \in S \mid t \text{ has no predecessors in } S\}$;
31.   $S := S - S_{front}$;
32. **while** $S_{front}$ is not empty;
33. **return** *Front;*
```

Complexity Analysis

Algorithm 2 uses Algorithm 1, on line 2, and thus must be $O(N^3)$.

2.4.3 Algorithm for Dealing with Compound Events

It is incorrect to compute the convex closure of two convex events by simply merging their front and back vectors [15]. Figure 2.10[8] illustrates this point.

In Figure 2.10, the rectangles *E* and *F* represent two convex events and *e* represents a primitive event that is not a part of *E* and *F*. However, it can be seen in the figure that *e* is a member of the convex closure of *E* and *F*, specifically its back vector. As a result, merging the back vectors of both *E* and *F* is not enough to build the correct back vector for *E* and *F*'s convex closure because *e* is not included during the merging [15].

[8]Taken from Ping Xie's thesis [15].

Lines 3-5:	$Back = \{e_{1,3}, e_{2,7}, e_{3,7}, e_{4,6}\}$, $S_{front} = \{e_{1,3}\}$, and $Front = S_{front}$				
Lines 6-8:	$Scanned = \{\}$, $S = \{\}$, and $P = \{1\}$				
Lines 14-27:	$\boldsymbol{f{=}e_{1,3}}$	$\boldsymbol{f{=}e_{1,4}}$	$\boldsymbol{f{=}e_{1,5}}$		
	$t{=}e_{2,3}$ $S{+}{=}\ e_{2,3}$	$t{=}e_{2,6}$	$t{=}e_{3,5}$ $S{+}{=}\ e_{3,5}$		
Lines 28-32:	$Scanned{=}\{1\}$, $S_{front}{=}\{e_{2,3}\}$, $Front{=}\{e_{1,3},e_{2,3}\}$, and $S{=}\{e_{3,5}\}$				
Line 8:	$P = \{2\}$				
Lines 14-27:	$\boldsymbol{f{=}e_{2,3}}$	$\boldsymbol{f{=}e_{2,4}}$	$\boldsymbol{f{=}e_{2,5}}$	$\boldsymbol{f{=}e_{2,6}}$	$\boldsymbol{f{=}e_{2,7}}$
	continue	$t{=}e_{3,2}$ $S{=}\ e_{3,2}$	$t{=}e_{3,4}$	*continue*	$t{=}e_{3,6}$
Lines 28-32:	$Scanned{=}\{1,2\}$, $S_{front}{=}\{e_{3,2}\}$, $Front{=}\{e_{1,3},e_{2,3},e_{3,2}\}$, and $S{=}\{\}$				
Line 8:	$P = \{3\}$				
Lines 14-27:	$\boldsymbol{f{=}e_{3,2}}$	$\boldsymbol{f{=}e_{3,3}}$	$\boldsymbol{f{=}e_{3,4}}$	$\boldsymbol{f{=}e_{3,5}}$ **&** $\boldsymbol{e_{3,6}}$	$\boldsymbol{f{=}e_{3,7}}$
	continue	$t{=}e_{4,4}$ $S{=}\ e_{4,4}$	$t{=}e_{2,5}$ *continue*	*continue*	$t{=}\ e_{4,5}$
Lines 28-32:	$Scanned{=}\{1,2,3\}$, $S_{front}{=}\{e_{4,4}\}$, $Front{=}\{e_{1,3},e_{2,3},e_{3,2},e_{4,4}\}$, and $S{=}\{\}$				
Line 8:	$P = \{4\}$				
Lines 14-27:	$\boldsymbol{f{=}e_{4,4}}$	$\boldsymbol{f{=}e_{4,5}}$			
	continue	$t{=}\ e_{3,7}$ *continue*			
Lines 28-32:	$\boldsymbol{Scanned{=}\{1,2,3,4\}}$**,** $\boldsymbol{S_{front}{=}\{\}}$**,** $\boldsymbol{Front{=}\{e_{1,3},e_{2,3},e_{3,2},e_{4,4}\}}$**, and** $\boldsymbol{S{=}\{\}}$				

Figure 2.9: Computing the front vector of Figure 2.7, given $E = \{e_{1,3}, e_{3,7}\}$

Ping Xie [15] took a simple approach to computing the convex closure between two convex events. He used the primitive events of the front and back vectors of both convex events to build an event set. The resulting convex closure of this set, by applying Algorithm 1 and Algorithm 2, is the convex closure between the two convex events. Algorithms 3, 4, and 5 give the pseudo-code for computing the convex closure between two convex events.

Algorithm 3

```
1.  function EXPAND($E$)
2.     $E'$:= {};
3.     for each event $e$ in $E$
```

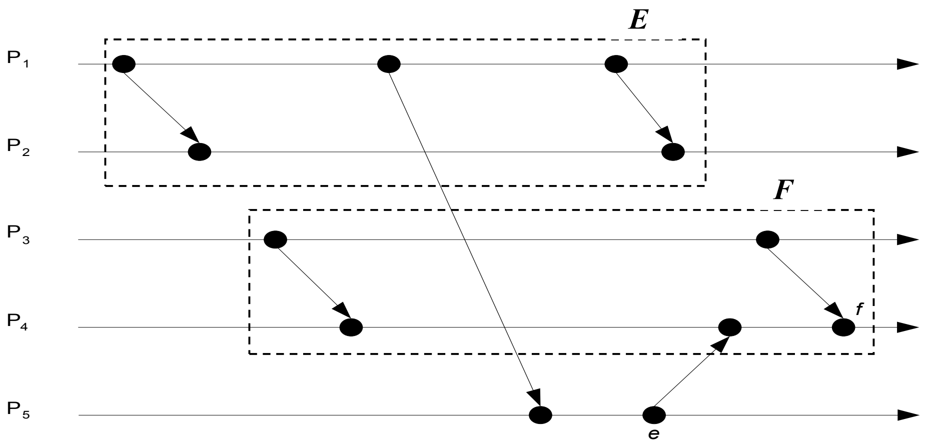


Figure 2.10: An example of computing the convex closure of two convex events

```
4.     if $e$ is a primitive event
5.       Add $e$ to $E'$;
6.     else
7.       Add $e$'s back vector to $E'$;
8.       Add $e$'s front vector to $E'$;
9.   return $E'$;
```

Algorithm 4

```
1. function COMPUTE_BACK_EXT($E$)
2.   return COMPUTE_BACK(EXPAND($E$));
```

Algorithm 5

```
1. function COMPUTE_FRONT_EXT($E$)
2.   return COMPUTE_FRONT(EXPAND($E$));
```

2.4.4 Pruning Technique

Ping Xie [15] employed a pruning technique, known as relevancy-restriction, to improve the performance of the naive BT algorithm. The relevancy-restriction mechanism is based on using a node's convex event that has been computed to remove events from event domains belonging to another node before its sub-tree is evaluated. Figures 2.11 and 2.12 are used to help illustrate this mechanism[9].

[9]Taken from Ping Xie's thesis [15].

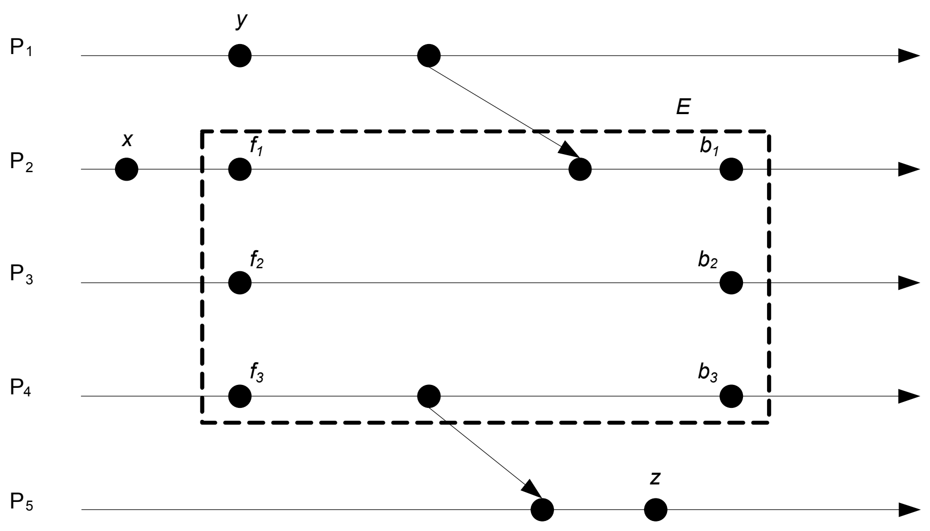


Figure 2.11: An example of pruning variables' domain for $A \| (a \to b)$

Suppose that E, in Figure 2.11 is the convex event that has been computed and is associated with node A in Figure 2.12 E can be used to remove primitive events from a's domain and b's domain before node B's sub-tree, in Figure 2.12, is evaluated. An event e is removed from a's domain and b's domain if it satisfies one of the following two conditions:

1. $P_e \in \mathbf{l}E$,
2. $(e \to f) \wedge (P_e = P_f) \wedge (f \in GP(E))$

These two conditions apply specifically to an event predicate that contains the concurrent relation as seen in Figure 2.12. For example, if event x or y in Figure 2.11 is associated with a's domain or b's domain then any convex event, associated with node B in Figure 2.12, that contains either x or y cannot be concurrent with E because events x, and y satisfy conditions 1 and 2, respectively. In other words both conditions check for events in a's domain or b's domain that happened-before E. Therefore, any convex event containing either x or y will happen-before E according to our precedence relation definition between two compound events. As a result, events x and y would not be included as matched events from a's domain or b's domain for satisfying the event predicate in Figure 2.11, $A \| (a \to b)$.

Ping Xie [15] described a third pruning condition that he did not implement because it requires the least successor of $S_{front}(E)$ in each process $LS(E)$ to be computed which is an expensive exercise due to the fact that F/M timestamp does not encode the least-successor information. The third condition is defined as follows:

Definition 13:

Let N be the number of processes in the computation $LS(E)$ is defined as follows:

$V(E) \quad := \; S_{front}(E) \cup \{t \mid t = ls_{pi}(e), \text{ where } e \in S_{front}(E) \text{ and } i \in [1, N]\}$,

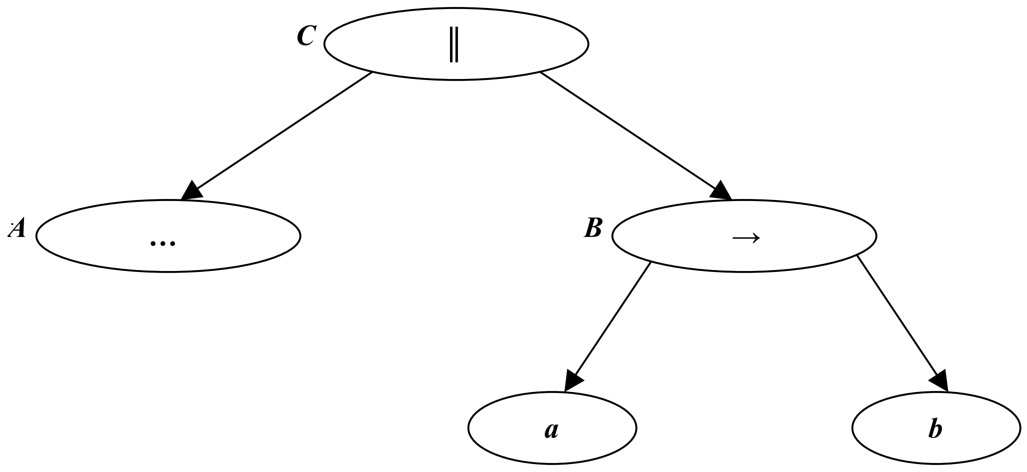


Figure 2.12: A parse tree representing $A \| (a \rightarrow b)$

$$LS(E) := \left\{ \begin{array}{l} t \in V(E) \mid \text{for at least one process } P \text{ on which } t \text{ occurs, there does} \\ \text{not exist a primitive event } e \text{ from } P \text{ in } V(E) \text{ with } e \text{ preceding } t \end{array} \right\}$$

LS(*E*) represents the least successors of $S_{front}(E)$ on each process.

The third pruning condition is then as follows:

$$(f \rightarrow e) \wedge (P_e = P_f) \wedge (f \in LS(E)) .$$

The primitive event *z* in Figure 2.11 satisfies the third condition where *z* is associated with event *e* that is defined in the condition.

In contrast to dealing with concurrent predicates, such as $A \| (a \rightarrow b)$, the pruning strategies described above are ineffective in dealing with the happened-before predicate, for example, $A \rightarrow (a \rightarrow b)$, because of the involvement of an existential quantifier in the precedence definition [15]. As a result, Ping Xie introduced a pruning condition for dealing with the happened-before predicate: only the events contained in *E*, a convex event that is associated with *A*, are excluded because this predicate represents the precedence test for two disjoint convex events that can be determined locally. Similarly, this condition is the only one applied to the limited operator.

The relevance of pruning to our research will be dealt with later in the thesis.

2.4.5 An Overview of Ping Xie's Implementation

Ping Xie's [15] offline event-predicate detection system was built on top of the Eclipse platform. This platform is designed for building integrated development environments (IDEs) that can be used to create applications as diverse as web sites, embedded Java^{TM} programs, C++ programs, and Enterprise JavaBeans^{TM} [4]. Ping Xie chose to build his application in the Java IDE. However, Eclipse does not provide tools for event timestamping and event scrolling. As a result, these key functions were ported from the Partial-Order Event Tracer (POET) [13, 14] by Ping Xie.

POET is a client/server application composed of a number of processes. These processes communicate with each other to provide an event visualization tool that offers a wide range of functions from collecting run-time event data to presenting collected events using a process-time diagram. Several different computing environments have been instrumented for POET including Java, PVM, OSF DCE, and μC++.

Each computing environment has its own target-description file that describes the event types in the environment and encodes information on how to present the events in a process-time diagram [14]. Information about events displayed in POET can be saved to an ASCII file known as a UEF file. This file contains information about events and the relationships between them including event type, partner event, stream data, trace data, etc. Therefore, an event display can be generated from a UEF file and a target-description file that relates to the computing environment from which the events were collected.

Ping Xie implemented a user interface to capture a UEF file and a target-description file to build the process-time diagram in Eclipse. In addition, the collection of events is stored in a relational database. He used the HSQL (Client/Server mode) 1.7.1 database system that is part of the Hyades Project. The Hyades Project is an integrated test, trace and monitoring environment based on Eclipse that provides standards, tools, and tool operability across the test process [5]. An example of importing a UEF file into an event database is shown in Figures 2.13 through 2.15. Figure 2.16 shows an example of a process-time diagram of an imported event database in Eclipse.

Figure 2.13[10] is called an Eclipse *new wizard* window. The user opens this window by selecting menus **File**, **New**, and **Other**, sequentially. "POET" is selected from the list of Java project types that are located in the left panel of the window. This action produces only one selection in the right panel of the window, "Event Database." Therefore, clicking on the **Next** button produces a new window that allows the creation of an empty event database, Figure 2.14.

This window is opened with a default *container* name that includes the name of the project. The user supplies the name of the database and selects the database system. As seen in Figure 2.14, the name of the event database is philos2. The remaining four fields in Figure 2.14 are automatically set with default values whenever the database system is selected. The **Next** button is clicked again. This action produces a new window that imports a UEF file and a target-description file, Figure 2.15.

Figure 2.15 shows that the user selected *philos.uef*. This file contains event data that were collected from a μC++ program called Philosophers. In addition, the target-description file that corresponds to the μC++ computing environment was selected. The user clicks **Finish** to create the *philos* event database. In addition, the name of the database, supplied in Figure 2.14, is placed under the project name that is located in the upper-left panel of the Eclipse parent window. The user double-clicks the database name to produce the process-time diagram, Figure 2.16.

[10]An empty project in Eclipse must be created before Figure 2.13 can be produced. The example assumes that the project was named *test*.

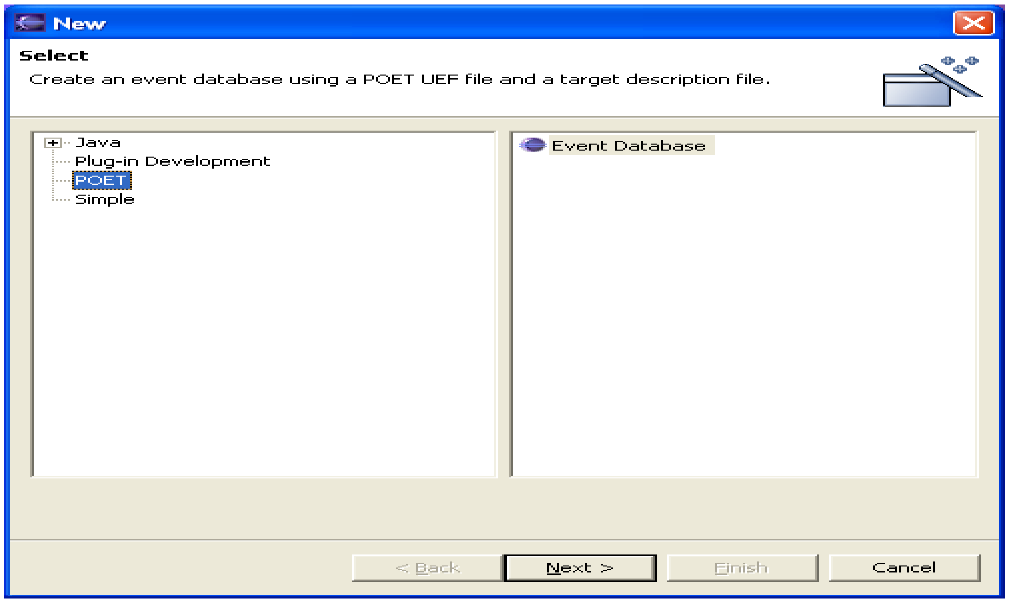
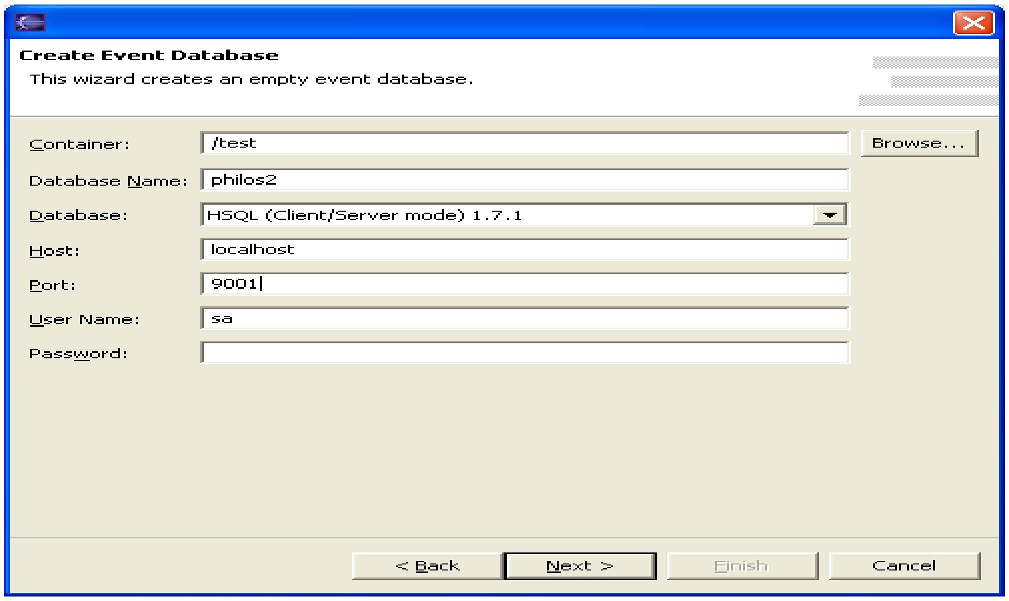


Figure 2.13: Step 1 in creating an event database



Create Event Database

This wizard creates an empty event database.

Container: /test

Database Name: philos2

Database: HSQL (Client/Server mode) 1.7.1

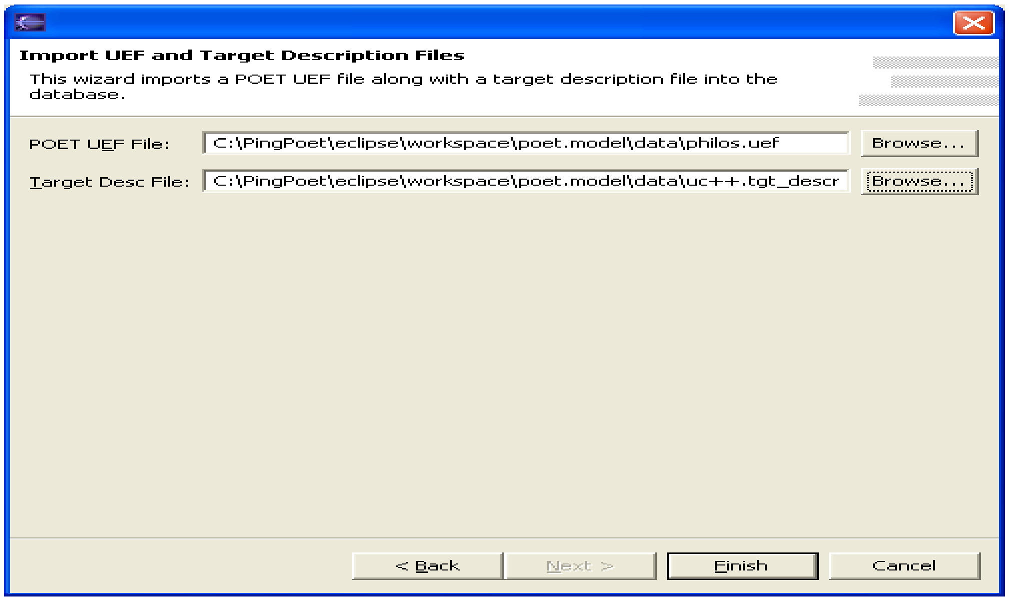
Host: localhost

Port: 9001

User Name: sa

Password:

Figure 2.14: Step 2 in creating an event database


Import UEF and Target Description Files

This wizard imports a POET UEF file along with a target description file into the database.

POET UEF File: C:\PingPoet\eclipse\workspace\poet.model\data\philos.uef

Target Desc File: C:\PingPoet\eclipse\workspace\poet.model\data\uc++.tgt_descr

Figure 2.15: Step 3 in creating an event database

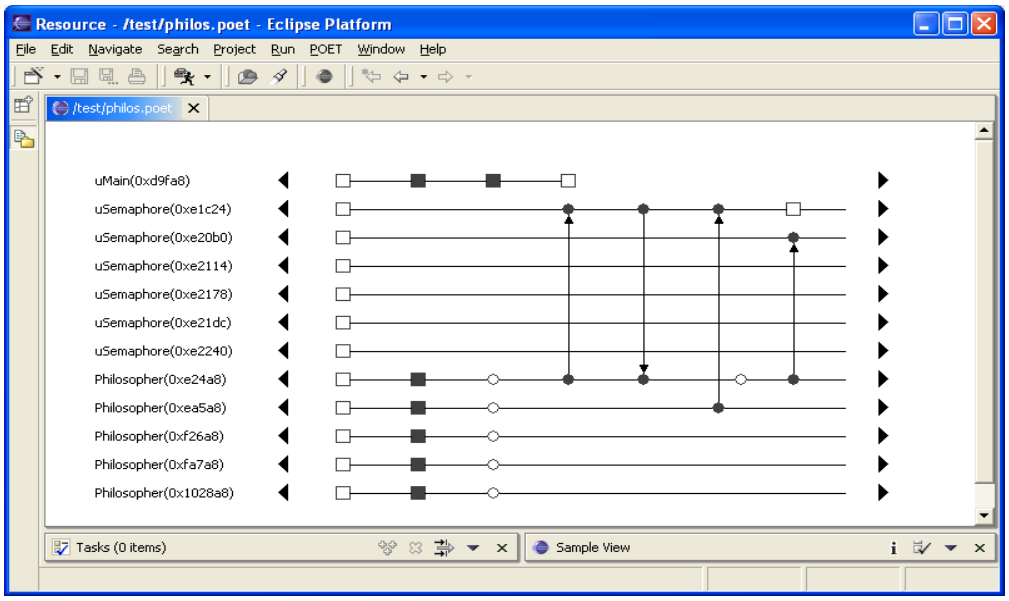


Figure 2.16: A process-time diagram in Eclipse

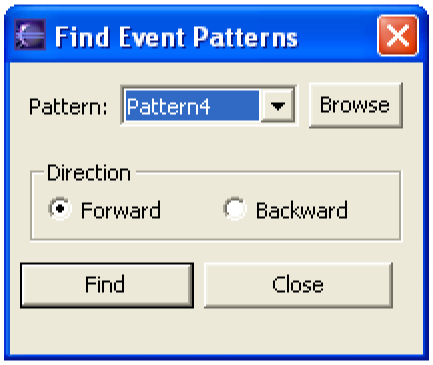


Figure 2.17: Pattern selection dialog

Figure 2.17 was produced from menu option "POET" from Figure 2.16. The user clicks **Browse** to locate files, with a ".pattern" extension, that contain the event predicates. As seen in the figure above, *Pattern4* was selected from a pattern file. The user locates the first matched events of *Pattern4* by clicking **Find**. Other matched events, if any, can be located by subsequent **Find** operations. The first matched events of *Pattern4* are shown below[11].

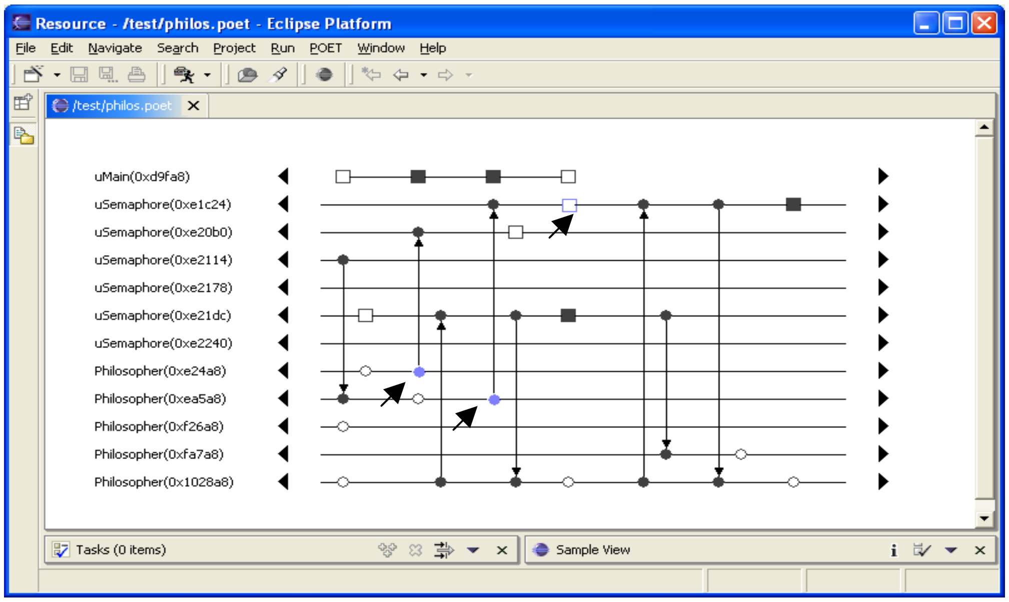


Figure 2.18: The first matched events for Pattern4

Philosopher(0xe24a8) || (Philosopher(0xea5a8) → uSemaphore(0xe1c24))

[11]Each matched event is shown with an arrow pointing at it.

Chapter 3
Experiments and Results of Two Hierarchical Event-Predicates

This chapter presents two hierarchical event-predicate detection experiments. These experiments are not intended to provide a thorough analysis of event-predicate-detection performance. Rather, both are intended to demonstrate the cause of the inefficiency of Ping Xie's algorithm.

Both Experiments 1 and 2 are based on finding the first search result as demonstrated in Section 2.4.4. A set of results was collected from the section of Ping Xie's application that matches events and computes their convex-closures. Ping Xie's application had only two data structures that captured matched events, referred to hereafter as the *left* domain/node and the *right* domain/node. The result set our experiments collected consists of the following information:

1. The number of matched event sets from *left*'*s* and *right*'*s* domains, categorized as follows:
    a) PRIM-PRIM − one event from each domain
    b) CONVEX-PRIM − multiple events from *left*'*s* domain and one event from *right*'*s* domain or PRIM-CONVEX which is the converse of CONVEX-PRIM
    c) CONVEX-CONVEX − multiple events from each domain
2. The number of convex closures of event sets PRIM-PRIM, CONVEX-PRIM, and CONVEX-CONVEX
3. The time taken to compute the closure of each category of event set
4. The front vector and the back vector of each closure of each category of event set

The set of results was written to three text files. Each file contains the information about each category of event set. However, in this document only a subset of the information in each file is presented: Appendices A and B consist of a subset of the set of results from Experiments 1 and 2, respectively.

Each result is written on a single line of six columns. For example:

7 50 {[8,3]} {[1,4]} {[1,3][8,3]} {[1,4][8,3]}.

Each column is separated by spaces. The third and fourth columns are matched events from *left*'s domain and *right*'s domain, respectively. Since, in this example, both columns contain one event from each domain, together they represent the PRIM-PRIM event set {[8,3][1,4]}. The numerical value 7 in the first column indicates that this is the seventh PRIM-PRIM event set. Note that there is no synchronization among the three files. That is, there is no information in the files that relates to the sequence of occurrences among the category of event sets. The second column indicates the time taken, in milliseconds, to compute the closure of only the event set, in this case 50 ms. The last two columns are the front vector and the back vector of the convex closure of the event set represented in the third and fourth columns, respectively.

Hardware	Intel Pentium IV 1.99GHz/480MB
Operating System	Red Hat 9.0/Kernel 2.4.20-8
JDK	SUN JDK 1.4.2_02 for Linux
RDBMS	HSQL (Client/Server mode) 1.7.1 (installed locally)

Figure 3.1: Test environment: software and hardware

The following experiments used an event database *philos* that contains events collected from a µC++ program called Philosophers. This program simulates the classical Dining-Philosophers problem. In addition, the hardware and software used in the experiments can be seen in Figure 3.1

3.1 Experiment 1

The event predicate for experiment 1 is as follows:

SemaphoreBlock1 := ["uSemaphore(0xe1c24)", "", ""];

PhilosopherA := ["Philosopher(0xe24a8)", "", "uP"];

PhilosopherB := ["Philosopher(0xea5a8)", "", "uP"];

PatternA := (PhilosopherB −−> SemaphoreBlock1);

Pattern4 := PhilosopherA | | PatternA;

Predicate *Pattern4* is evaluated against *philos*. In the context of this program, an event associated with a text string "µP" represents a P operation on a semaphore. Predicate *SemaphoreBlock1* represents any type of unary event on semaphore *uSemaphore*(0xe1c24). Predicate *PatternA* searches for pairs of the P operation from *Philosopher*(0xea5a8) that precedes *uSemaphore*(0xe1c24). Therefore, *Pattern4* finds concurrent pairs of P operation from *Philosopher*(0xe24a8) and *PatternA*.

3.1.1 First Search Result

The matched events {[1,112][7,48][8,30]} are the first search result for *Pattern4* and were found in approximately four hours. The search consisted of 47,047 convex closures of PRIM-PRIM event sets and only one convex closure of a PRIM-CONVEX event set.

From the results in Appendix A, the first matched event from *left's* domain and *right's* domain is [8,3] and [1,4], respectively. The search continues with other matched events from *right's* domain in increments of one event up to event [1,571]. This trend is repeated whenever the matched event from *left's* domain changes.

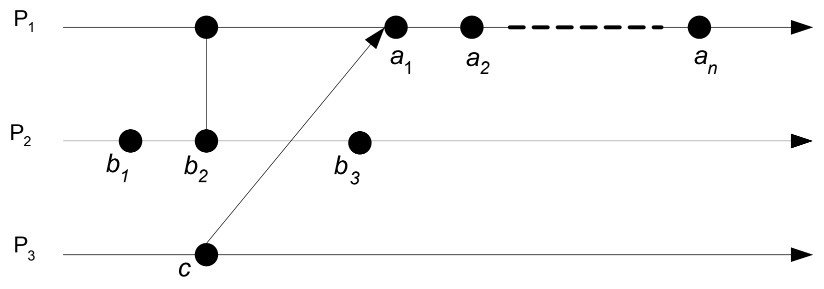


Figure 3.2: An example of a process-time diagram for $b \| (c \rightarrow a)$

The time taken to compute the convex closure of each event set up to the 139th event set was less than 35 milliseconds per closure. However, from the 140th event set, the closure computation time per event set increased to 260 milliseconds and significantly increased to nearly one second at the 542nd event set. This increase in time is due to the gradual increase in the number of events in the closure computation. The expense of the first search result was mainly attributed to Ping Xie's algorithm that computed closures over a similar set of events many times. Based on Algorithms 3, 4, and 5, the closure of each subsequent event set repeats the closure computation for the previous event set. For example, the closure computation for event set {[8,3][1,546]} involved checking events between event set {[8,3][1,545]} that had been checked in the previous closure computation. This trend is seen throughout the entire search result.

Experiment 1 could have taken less time if Ping Xie's third pruning condition was implemented. Figure 3.2 illustrates a process-time diagram that depicts a subset of events for event pattern $b \| (c \rightarrow a)$ that is similar to *Pattern4* with primitive events *b*, *c*, and *a* belonging to processes PhilosopherA, PhilosopherB, and SemaphoreBlock1, respectively.

Suppose in the search operation, event b_1 was matched as an event from PhilosopherA. Ping Xie's search algorithm would match event *c* from PhilosopherB. Pruning conditions one and two would be applied to event *c* with respect to event b_1. Event *c* does not satisfy either condition. As a result, *c* is concurrent with event b_1 and could be a possible matched event for $b \| (c \rightarrow a)$. His search algorithm then matches event a_1 from SemaphoreBlock1 that satisfies the sub-pattern $c \rightarrow a$. Ping Xie computes the convex-closure of event set $\{c, a_1\}$ and evaluates whether the closure is concurrent with event b_1. Obviously, they are not concurrent pairs because a_1 is a successor to b_1. His search algorithm continues to compute closures over event sets $\{c, a_2\}$ to $\{c, a_n\}$.Again, these closures will not be concurrent with event b_1 for the same reason. However, if he had evaluated the events from SemaphoreBlock1 against b_1 as he did with events from PhilosopherB, he would have significantly reduced the number of convex closures that were performed. Events a_1 to a_n would have been pruned from SemaphoreBlock1's domain. This approach is simple and does not require the third pruning condition because b_1 is a primitive event, but if event b_1 was a compound event, the third pruning condition would become necessary.

3.2 Experiment 2

The event predicate for experiment 2 is as follows:

```
SemaphoreReceive  :=  ["uSemaphore(0xe1c24)", "thread received", ""];
  SemaphoreBlock  :=  ["uSemaphore(0xe1c24)", "thread block", ""];
    PhilosopherA  :=  ["Philosopher(0xe24a8)", "", "uP"].SemaphoreReceive;
    PhilosopherB  :=  ["Philosopher(0xea5a8)", "", "uP"].SemaphoreReceive;
       Semaphore  :=  ["uSemaphore(0xe1c24)", "thread received", ""];
         Pattern  :=  (PhilosopherA --> PhilosopherB)
                      -(Semaphore)->SemaphoreBlock
```

Predicate *Pattern* is also evaluated against *philos*. Predicate *SemaphoreBlock* represents unary events of type "thread block" on semaphore *uSemaphore*(0xe1c24). Therefore, predicate *Pattern* finds pairs of synchronized P operations on *uSemaphore*(0xe1c24) from process *Philosopher*(0xe24a8) and Philosopher(0xea5a8) that precedes a "thread block" before a "thread received" on the same semaphore.

3.2.1 First Search Result

The matched events {[1,1][1,3][1,4][7,3][8,3]} are the first search result for *Pattern* and were found after nearly one hour. The search consisted of 148 convex closures of PRIM-PRIM event sets and 6014 convex closures of CONVEX-PRIM event sets.

From the results in Appendix B, the first matched event from *left's* domain and *right's* domain from the PRIM-PRIM event set category is [7,3] and [8,3], respectively. Although not shown in Appendix B, the first matched events, from the CONVEX-PRIM event set category, from *left's* domain are {[1,1][7,3][8,3]} and the first matched event from *right's* domain is {[1,4]}.With regards to the PRIM-PRIM event set category, the search continues with other matched events from *right's* domain in increments of nine events up to event [8,138]. This trend is repeated whenever the matched event from *left's* domain changes. In contrast, for the CONVEX-PRIM event-set category, the search continues with other matched events from *right's* domain in increments of six events up to event [1,560]. This trend is closely repeated whenever the matched events from *left's* domain change.

The time taken to find the first search result for *Pattern* can be explained in a similar way to that of *Pattern4*. However, the first search result for *Pattern* took significantly less time to find than *Pattern4's* first result because *Pattern* computed significantly fewer convex closures than *Pattern4*. *Pattern4* required 47,048 convex-closure computations while *Pattern* required 6,162 convex-closure computations.

	No. of Convex-Closures per Event-Set Category			Total	
Experiments	**PRIM-PRIM**	**CONVEX-PRIM**	**CONVEX-CONVEX**	**Convex-Closures**	**Time(hrs)**
Experiment 1	47, 047	1	0	47,048	4
Experiment 2	148	6,014	0	6,162	1

Table 3:1: Comparison of Results from Experiments 1 and 2

3.3 Comparison of Results from Experiments 1 and 2

From Table 3.1, it can be inferred that the average time taken to compute the closure of a CONVEX-PRIM event set is almost twice the average time taken to compute the closure of a PRIM-PRIM event set, for these patterns and event set. The number of times a unique event set was matched from *left*'s domain was collected in a fourth text file. This file contains information about the sequence of occurrences among the category of event sets during a search operation. A subset of this information for Experiment 1 and 2 is presented in Appendices E and F, respectively. Each line of results consists of four columns of data. Column 1 represents the number of times a unique event set from *left's* domain was matched. Other column data will be dealt with in Chapter 4 because they are not relevant to our analysis in this chapter.

From Experiment 1, the total number of unique PRIM event sets, which is the number of times the backtracking algorithm selects a matched primitive event from *left*'s domain, is 154. This data was taken directly from the fourth text file because there was only one PRIM-CONVEX event set throughout the search operation. However, Column-1 data from Experiment 2 could not be ascertained for both PRIM and CONVEX event sets from *left*'*s* domain because the entire search operation consisted of interleaving event sets.

Hence, our aim to categorize event sets was nullified because we hypothesized that Ping Xie's convex-closure computation was ineffective for PRIM-CONVEX event sets. This hypothesis proved to be false. Experiment 1 consisted of 1,429 unique front vectors. As a result, a unique front vector was computed, on average, for every 33 convex-closures computed. Similarly, Experiment 2 consisted of 185 unique front vectors and therefore, on average, 33 convex-closures were computed for each unique front vector. Therefore, the inefficiency of Ping Xie's offline event-predicate detection system, for Experiments 1 and 2, as described earlier, is caused by the large number of closures computed over similar sets of events.

Chapter 4
Proposed Algorithm

4.1 The Approach

The proposed algorithm is an extension of Ping Xie's algorithm, and is based on caching matched sub-patterns. These sub-patterns are represented by the categories of event sets described in Chapter 3. However, CONVEX-CONVEX event sets are not dealt with in our approach because there was no instance of CONVEX-CONVEX sub-patterns that were matched. Therefore, our algorithm only dealt with PRIM-PRIM and CONVEX-PRIM categories of event sets.

4.1.1 The Basic Idea

As discovered by our experiments, Ping Xie's algorithm computes closures over a similar set of matched events many times. Therefore, our algorithm employs a caching approach to computing the closure over the back vector of the cached convex event and the new matched primitive event. Figure 4.1 illustrates the basic idea of our proposed algorithm.

Suppose that the event predicate we are searching for is $x_4 \rightarrow (y_1 \rightarrow z_3)$ and events x_4, y_1, and z_3 represent primitive events that belong to processes 4, 1, and 3, respectively. Primitive events a, and c_1 are matched events from *left*'s and *right*'s domains, respectively. Ping Xie's approach is to compute the convex-closure of event set $\{a, c_1\}$, $CC(\{a, c_1\})$, which is E_1 in the figure. In addition, the event q is matched from process 4. At this point, the relation $q \rightarrow E_1$ is evaluated. From Figure 4.1, it can be seen that event q does not happen-before E_1. As a result, the search operation backtracks to *right*'s domain and matches another event, c_2. Ping Xie's algorithm then computes $CC(\{a, c_2\})$, shown as E_2 in the figure. This time, event q happen-before E_2 and therefore a match is found for the event predicate $x_4 \rightarrow (y_1 \rightarrow z_3)$. In addition, it can be seen, from Figure 4.1, that E_2 is a superset of E_1. On the other hand, our approach caches E_1 and computes the convex-closure of the event set $\{a, b_2, c_1, c_2\}$ provided the cached matched event(s) from *left*'s domain is equal to the current matched event(s) from *left*'s domain and the cached matched event(s) from *right*'s domain precedes the current matched event(s) from *right's* domain, $a=a$ and $c_1 \rightarrow c_2$. The resulting convex-closure E_2 is the front vector of E_1 (cached front vector) and the back vector of $CC(\{a, b_2, c_1, c_2\})$ if the location set of E_2 is equal to the location set of E_1 and $E_1 \rightarrow c_2$. The following theorems prove the correctness of this cached-closure computation.

Theorem 3

$CC(\{E, e'\}) = CC(\{E, e, e') \Leftrightarrow E \leq e \leq e'$

Proof

For the purpose of these theorems, we use the reflexive form of happened-before:

$$a \leq b \Leftrightarrow a \rightarrow b \text{ or } a = b.$$

Assume $E \leq e \leq e'$ $CC(\{E, e'\}) \neq CC(\{E, e, e')$

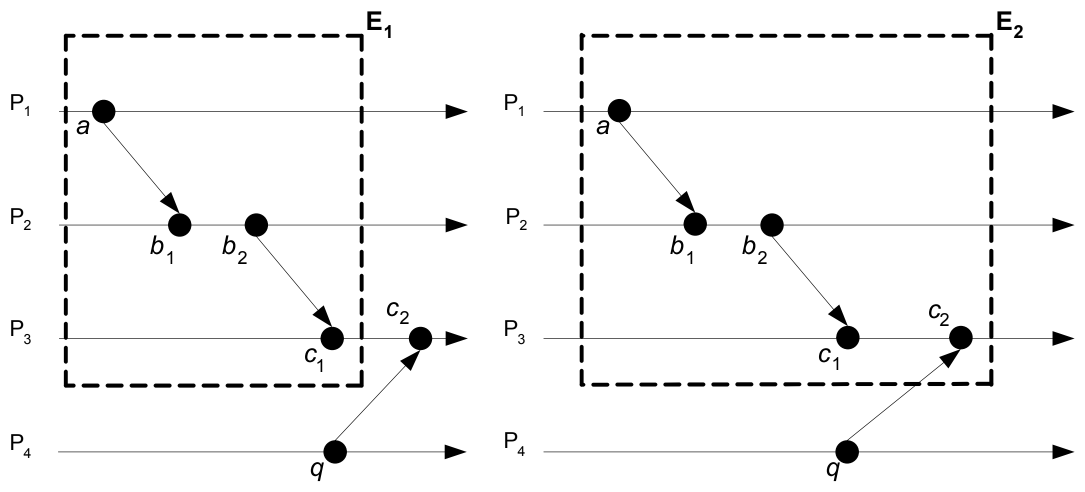


Figure 4.1: An illustration of our proposed-algorithm approach

$\exists x : x \in CC(\{E, e'\}) \wedge x \notin CC(\{E, e, e')$ or $x \notin CC(\{E, e'\}) \wedge x \in CC(\{E, e, e')$

1. $x \in CC(\{E, e'\}) \Leftrightarrow \exists y, z : y, z \in \{E, e'\} : y \leq x \leq z$

   Therefore $\exists y, z : y, z \in \{E, e', e\} : y \leq x \leq z$

   Hence $x \in CC(\{E, e, e')$.

2. $x \notin CC(\{E, e'\}) \wedge x \in CC(\{E, e, e')$

   Therefore $\exists y, z : y, z \in \{E, e, e'\} : y \leq x \leq z$

   Since $E \leq e \leq e'$, $z \leq e'$

   Therefore $x \leq e'$

   Since $y \in \{E, e, e'\}$ and $E \leq e \leq e'$, $\exists w$: $w \in E : w \leq y$

   Therefore $\exists w$: $w \in E : w \leq x$ since $y \leq x$

   Therefore $\exists w$: $w \in \{E, e'\} : w \leq x$

   Hence $x \in CC(\{E, e'\})$.

Lemma 1

$\exists a, b : a, b \in CC(E) \cup \{e\}$ and $\exists w : a \leq w \leq b \wedge CC(E) < \{e\}$ then $w \in CC(E \cup \{e\})$

Proof

Proof is performed by contradiction.

Assume $a, b \in CC(E) \cup \{e\}$, $a \leq \mathrm{w} \leq b$ and $\mathrm{w} \notin CC(E \cup \{e\})$

Since $a, b \in CC(E) \cup \{e\}$, 1.($a, b \in CC(E)$) $\vee$ 2.($a, b \in \{e\}$) $\vee$ 3.((a $\in CC(E)$ and $b \in \{e\}$))

1. Since $a \in CC(E)$, $\exists x : x \in E : x \leq a$.

Since $b \in CC(E)$, $\exists y : y \in E : b \le y$.

Therefore $x \le a \le w \le b \le y$.

Since $x \le a \le w \le b \le y$, $\exists x, y : x, y \in E : x \le w \le y$.

Therefore $w \in CC(E)$.

Since $w \in CC(E)$, $w \in CC(E \cup \{e\})$ because $CC(E) \subseteq CC(E \cup \{e\})$.

2. Since $a, b \in \{e\}$, $e \le w \le e$.

   $e \le w \le e \Rightarrow w = e$.

   Therefore $w \in CC(E \cup \{e\})$.

3. $a \in CC(E) \Rightarrow \exists x : x \in E : x \le a$.

   Since $b \in \{e\}$ $\exists x, b : x, b \in E \cup \{e\} : x \le w \le b$.

   Hence $w \in CC(E \cup \{e\})$.

Theorem 4

If $\mathbf{l}CC(E \cup \{e\}) = \mathbf{l}CC(E) \wedge CC(E) < \{e\}$ *then*

$CC(E \cup \{e\}) = CC(E) \cup CC(back(CC(E)) \cup \{e\})$,

where E is an arbitrary event set and e is an arbitrary primitive event.

Proof

First we show $CC(E) \cup CC(back(CC(E)) \cup \{e\}) \subseteq CC(E \cup \{e\})$.

If $x \in CC(E) \cup CC(back(CC(E)) \cup \{e\})$ then $x \in CC(E)$ or $x \in CC(back(CC(E)) \cup \{e\})$

If $x \in CC(E)$ then $x \in CC(E \cup \{e\})$ because $CC(E) \subseteq CC(E \cup \{e\})$.

If $x \in CC(back(CC(E)) \cup \{e\})$ then $a \le x \le b$ for $a, b \in back(CC(E)) \cup \{e\}$.

Since $back(CC(E)) \subseteq CC(E)$ then $a, b \in CC(E) \cup \{e\}$ and $x \in CC(E \cup \{e\})$ by Lemma 1.

Therefore, $CC(E) \cup CC(back(CC(E)) \cup \{e\}) \subseteq CC(E \cup \{e\})$.

Next we show that $CC(E \cup \{e\}) \subseteq CC(E) \cup CC(back(CC(E)) \cup \{e\})$. In other words, we'll show that $\forall z : z \in CC(E \cup \{e\}) \Rightarrow z \in CC(E) \vee (back(CC(E)) \le z \le \{e\})$.

Proof is performed by contradiction.

1. $\exists z : z \in CC(E \cup \{e\}) \wedge z \notin CC(E) \wedge z \neg(\le) \{e\}$.

   $z \in CC(E \cup \{e\}) \Rightarrow \exists y : y \in E \cup \{e\} : z \le y$

   Since $z \notin CC(E) \wedge CC(E) < e$, then $y \notin E$.

   Thus $y = \{e\}$ and $z \le e$.

2. $\exists z : z \in CC(E \cup \{e\}) \wedge z \notin CC(E) \wedge z \parallel back(CC(E))$.

   If $z \parallel back(CC(E))$ then $P_z \notin \mathbf{l}back(CC(E))$, where P_z is the process in which z occurs.

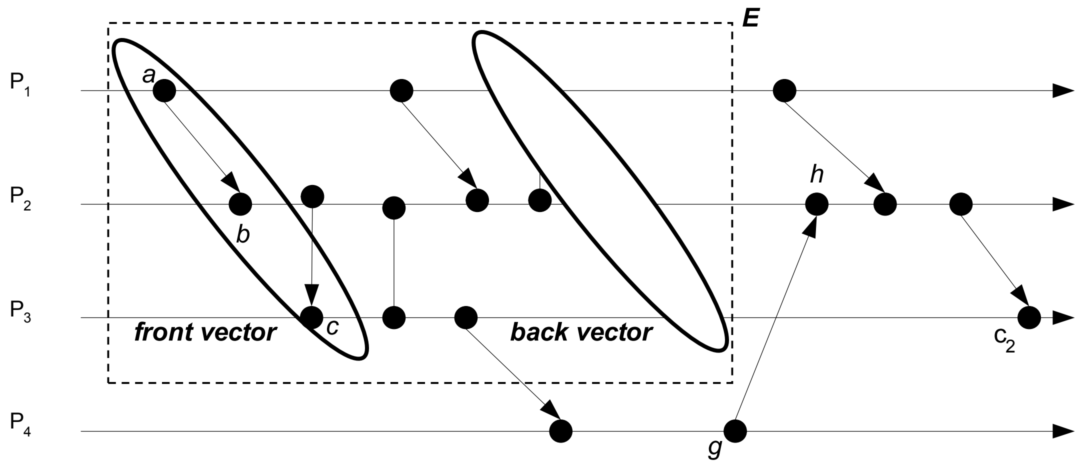


Figure 4.2: An example of a cached-closure violation

However, $\mathbf{l}back(CC(E)) = \mathbf{l}CC(E)$ (by definition of *back*, definition 12) $= \mathbf{l}CC(E \cup \{e\})$.

Therefore, $P_z \notin \mathbf{l}CC(E \cup \{e\})$.

However, $z \in CC(E \cup \{e\})$ and therefore, $P_z \in \mathbf{l}CC(E \cup \{e\})$

3. $\exists z : z \in CC(E \cup \{e\}) \wedge z \notin CC(E) \wedge z < back(CC(E))$.

   Since $z \in CC(E \cup \{e\})$, $\exists x : x \in E \cup \{e\} : x \leq z$

   Since $z < back(CC(E))$ then $x \in E$.

   Since $CC(E) < \{e\}$ thus $z \in CC(E)$.

Figure 4.2 shows an example that does not satisfy Theorem 4 because event *h* is a receive event that lies between the back vector of *E* and c_2, and its partner event *g* does not occur in the location set of *E.* Furthermore, event g is preceded by the front vector of *E*. Therefore, the front vector of the convex-closure of $\{a, c_2\}$ is different from the cached front vector $\{a, b, c\}$. Hence, Theorem 4 would not apply.

4.1.2 Our Algorithm

Based on Theorems 3 and 4, our proposed algorithm is performed as follows:

```
If ((cached matched-event(s) from left node = current matched event(s) from left node) AND
    (cached matched-event(s) from right node PRECEDES
     current matched-event(s) from right node))
Then
    /* Cached-closure computation */
```

Compute convex-closure of input event set containing the back vector of the cached convex event and current matched event from right's *domain*

***If** Theorem 4 prerequisites are violated **Then** apply Ping Xie's algorithm*

Else

Compute convex-closure using Ping Xie's algorithm

Cache newly computed convex closure.

Algorithms 6, 7, 8, and 9 implement the cached-closure computation. Algorithm 6 builds the input event set E′, using the events in the back vector of the cached convex event E ($Back_{cache}$) and the current matched event from *right*'s domain r′. The back vector is computed using Algorithm 7[12]. The pseudo code that checks for the condition that violates the prerequisites for Theorem 4 is shown in Algorithm 8[13]. Algorithm 9 implements Algorithm 8.

Algorithm 6

```
1. function DBMERGE(E)
2.   E′ := {};
3.   for each event e in E
4.     if e is a primitive event
5.       Add e to E′;                /* r′ */
6.     else
7.       Add e's back vector to E′;  /* $Back_{cache}$ */
8.   return E′;
```

Algorithm 7

```
1. function DB_BACK(E)
2.    return COMPUTE_BACK(DBMERGE(E));
```

Algorithm 8

```
1. function DB_FRONT(E, $Front_{cache}$, $Back_{cache}$)
2.   Back := COMPUTE_BACK(DBMERGE(E));
3.   $S_{front}$ := $\{t \in E \mid t$ has no predecessors in $E\}$;
4.   Front := $Back_{cache}$;
5.   Scanned := {};
6.   S := {};
```

[12]Algorithm 7 employs Ping Xie's algorithm, Algorithm 1.

[13]The changes that were made to Algorithm 2 can be identified by their bold-faced line numbers.

```
7.   do
8.     $P$ := l$Back$;
9.     if $P \neq$ l$Back_{cache}$
10.      return (COMPUTE_FRONT_EXT($E$));
11.    for each process $p$ in $P$
12.    {
13.      $start$ := the event from $p$ in $Front$;
14.      $end$ := the event from $p$ in $Back$;
15.      for each event $f$ between $start$ and $end$ inclusive
16.      {
17.        if $f$ is an asynchronous send event or a unary event
18.          continue;
19.        $t$ := $f$'s partner event;
20.        if ($P_t$ is not in l$Back_{cache}$ AND an event $g$ in $Front_{cache}$ precedes $t$)
21.          return (COMPUTE_FRONT_EXT($E$));
22.        if $P_t$ does not have events in $Back$ OR $P_t$ is in $Scanned$
23.          continue;
24.        if there is an event from $P_t$ in $S_{front}$
25.          continue;
26.        if no event from $P_t$ is in $S$
27.          add $t$ to $S$;
28.        else if $t$ precedes the event from $P_t$ in $S$
29.          replace that event with $t$;
30.      }
31.      add $p$ to $Scanned$;
32.    }
33.    add events in $S$ to $Front$;
34.    $S_{front} := \{t \in S \mid t$ has no predecessors in $S\}$;
35.    $S := S - S_{front}$;
36.  while $S_{front}$ is not empty;
37.  return $Front$;
```

Algorithm 9

```
1. function DB_FRONT_EXT($E$)
2.   for each event $e$ in $E$
```

```
3.        if $e$ is convex event
4.            $Front_{cache}$ = $e$'s front vector;
5.            $Back_{cache}$ = $e$'s back vector;
6         else  $Front_{cache} = Back_{cache} = e$;
7.    return DB_FRONT($E$, $Front_{cache}$, $Back_{cache}$);
```

4.1.2.1 Implementing Theorem 4 Checks in Algorithm 8

Only two lines in Algorithm 8 check for Theorem 4 prerequisite violations:

Line 9 checks the condition that if the location set of the newly computed back vector (*Back*) is not equal to the location set of the cached back vector, Theorem 4 is not applicable. In other words, $Front_{cache}$ and *Back* will have two different location sets. Therefore, the resulting cached-closure computation would be invalid because the front vector and the back vector of a convex event always have the same location set.

Line 20 implements the conditions illustrated in Figure 4.2. Although the location sets of both $Back_{cache}$ and *Back* are equal, Theorem 4 can still be inapplicable based on the following two conditions:

1. A partner event g of any event, between $Back_{cache}$ and r', belongs to a process that is not in the location set of $Back_{cache}$.
2. g is preceded by an event in $Front_{cache}$

Line 17 implies that event g must be the partner event of an asynchronous-receive event or a synchronous event.

4.1.2.2 Complexity Analysis

Algorithms 6, 7, 8, and 9 were used to implement our cached-closure computation. The complexity analysis for each algorithm is as follows:

Algorithm 6

The cost of Algorithm 6 is determined by the number of iterations of line 4 because r′ is only one event. The number of iterations of line 3 depends on N_c, the number of processes that were present in $Back_{cache}$. In the worst case $N_c = N$, and thus the complexity is $O(N)$.

Algorithm 7

Algorithm 7 requires invocation of Algorithm 6, at cost $O(N)$. It then invokes Algorithm 1 at cost $O(N^3)$. It thus has a cost of $O(N^3)$.

Algorithm 8

The cost differences between Agorithm 2 and 8 can be seen in the different lines that Agorithm 8 has, over Algorithm 2. First, in lines 8 and 9, it must determine if there is a cache hit. The cost of this is the cost of the number of elements in the back cache, which is N in the worst case. This is thus $O(N)$ cost.

		No. of Convex Closures	No. of Unique Fronts	Average No. of Closures per Unique Fronts	No. of successful Cached-Closures
Category of Event Sets	PRIM-PRIM	47,047	1,429	32.9	45,617
	PRIM-CONVEX	1	1	1.0	-
	CONVEX-CONVEX	0	0		
Overall		47,048	1,430	33.9	45,617

Table 4:1: Cache Performance Results from Experiment 1

The second difference is then that the algorithm must determine if there location sets match. This is determined by lines 15 to 21. In those lines, the events in the convex closure between back and "e" are checked to see if they are receives or synchronous, with partners that are outside the location set of the original closure, and those partners are successors to the cached value of Front. If they are, then Ping Xie's algorithm is employed (line 21). The worst case cost of this is $O(N_pN_e)$ where N_p is the number of processes and N_e is the number of events, since it is possible that all but a small number of events are part of the closure increment, and all but one process is in the location set. While this is bad in theory, in practice, the number of events is much smaller than the total number of events in the system.

Algorithm 9

The cost of Algorithm 9 is determined by the number of iterations of line 2. The number of iterations of lines 4 and 5 depends on N_c. In the worst case $N_c = N$, and thus the complexity is $O(N)$.

4.2 Results from Experiments 1 and 2 using the Proposed Algorithm

The data in Tables 4.1 and 4.2 came from an entire data set collected during the search operation in Experiments 1 and 2 using our algorithm, respectively. A subset of the data collected in Experiment 1 can be seen in Appendices C and E. In addition, a subset of the data collected in Experiment 2 can be seen in Appendices D and F.

The data in the last Column of Table 4.1 refers to the number of successful cached-closure computations: the number of times our algorithm was able to use Theorem 4. This data was collected in a fourth text file, described in Section 3.3. Each line of results consists of four columns of data: Column 1 was defined in Section 3.3; Column 2 is self-explanatory; Column 3 shows the number of times a unique event set from *left's* domain was matched against an event set from *right's* domain; and Column 4 refers to the data in the last Column of Table 4.1.

		No. of Convex Closures	No. of Unique Fronts	Average No. of Closures per Unique Fronts	No. of successful Cached-Closures
Category of Event Sets	PRIM-PRIM	148	44	3.4	-
	CONVEX-PRIM	6,014	141	42.7	-
	CONVEX-CONVEX	0	0		
Overall		6,162	185	46.1	5,754

Table 4:2: Cache Performance Results from Experiment 2

4.2.1 Experiment 1 Analysis

The overall cache hit rate (OCHR) in Experiment 1 is calculated from the following expression:

$$\text{OCHR} = \frac{\text{No. of Successful Cached - Closures}}{\text{Overall No. of Convex - Closures}}$$

$$\text{OCHR} = \frac{45{,}617}{47{,}048} = 0.970$$

Therefore, OCHR in Experiment 1 is 0.970. In other words, for every 1000 cached closure computations, 970 were satisfied from the cache. This result indicates that Theorem 4 was always applicable when a result was found in cache. Hence the number of unique fronts equals the number of times Ping Xie's algorithm was invoked instead of cache, 1,430 (47,048 − 45,617) times. As a result, the time taken to perform the first search result in Experiment 1 was significantly reduced, from nearly four hours to less than three minutes. A comparison of data in Appendix A and in Appendix C confirms the improved performance of the Proposed Algorithm over Ping Xie's algorithm. The 140th closure computation took on average less than 1 millisecond instead of 260 milliseconds. The 542nd down to 568th closure computations, each took on average less than 10 milliseconds instead of as much as 1,230 milliseconds, and so on.

4.2.2 Experiment 2 Analysis

The overall cache hit rate (OCHR) in Experiment 2 is calculated as follows:

$$\text{OCHR} = \frac{5{,}754}{6{,}162} = 0.934.$$

Therefore, for every 1000 cached closure computations in Experiment 2, 934 were satisfied from the cache.

In Table 4.2, the entire first search result in Experiment 2 consisted of 148 closures of PRIM-PRIM event sets and 6,014 closures of CONVEX-PRIM event sets. A total of 185 unique front vectors were computed out of 6,162 closure computations, 44 of which were computed from PRIM-PRIM event sets and 141 of which were computed from CONVEX-PRIM event sets.

As a result, a unique front vector was computed, on average, for every 3 convex-closures computed from a PRIM-PRIM event set and for every 43 convex-closures from a CONVEX-PRIM event set. From this result, caching was less effective for PRIM-PRIM event sets than it was for CONVEX-PRIM event sets for this pattern because of the interleaving of both event sets and the fact that our cache size was one, throughout the search operation. Therefore, the execution times for the PRIM-PRIM event-set category obtained from Ping Xie's algorithm and our algorithm are similar (see Appendices B and D).

Chapter 5
Conclusion and Future Work

5.1 Conclusion

This thesis has presented an efficient algorithm that reduces the execution time of Ping Xie's algorithm by over 94% for experiments 1 and 2. This was done by building hierarchical event predicates (complex patterns) in an incremental manner and employing a simple caching scheme that uses the most recently stored cache based on matched sub-patterns. The cache hit rate for the two hierarchical event predicates in experiments 1 and 2 were 0.970 and 0.934, respectively. As a result, both experiments took less than four minutes instead of experiment 1 that took almost four hours and experiment 2 that took close to one hour, to find the first search result.

The major cost of a convex-closure computation is the expense incurred when computing the front vector. This is caused by the fact that the F/M timestamp does not encode *least-successor* information and therefore the closure computation has to iterate over all constituent events in the resulting convex closure to compute the front vector [15]. By applying an incremental closure algorithm with the use of a suitable caching scheme, we were able to achieve significant improved performance over Ping Xie's algorithm.

5.2 Future Work

A possible future study is to perform a wider range of experiments, considering alternative event-data sets, on different hierarchical event predicates. This approach should produce a more comprehensive understanding of our algorithm.

A second future study that needs to be done, which is a spin-off from the above approach, is to study various caching schemes and see how well they perform. This may lead to an approach that favours a particular caching scheme based on certain criteria: the make-up of hierarchical event predicates, the size of the event collection, etc.

Finally, search operations of various hierarchical event predicates may consist of a reasonable number of Theorem-4 violations and/or closure computations involving CONVEX-CONVEX event sets. Therefore, this outcome presents two more possible future tasks: the development of a suitable patch and an approach to efficiently computing the closures of CONVEX-CONVEX event sets.

Appendix A
First Search Result of Ping Xie’s Algorithm in Experiment 1

1	0	{[8,3]}	{[1,4]}	{[1,3][8,3]}	{[1,4][8,3]}
2	10	{[8,3]}	{[1,5]}	{[1,3][7,9][8,3]}	{[1,5][7,9][8,3]}
3	0	{[8,3]}	{[1,6]}	{[1,3][7,9][8,3]}	{[1,6][7,10][8,3]}
4	0	{[8,3]}	{[1,7]}	{[1,3][7,9][8,3]}	{[1,7][7,10][8,3]}
5	0	{[8,3]}	{[1,8]}	{[1,3][7,9][8,3]}	{[1,8][7,10][8,4]}
6	10	{[8,3]}	{[1,9]}	{[1,3][7,9][8,3][9,3]}	{[1,9][7,10][8,4][9,3]}
7	0	{[8,3]}	{[1,10]}	{[1,3][7,9][8,3][9,3]}	{[1,10][7,10][8,4][9,3]}
8	10	{[8,3]}	{[1,11]}	{[1,3][7,9][8,3][9,3]}	{[1,11][7,10][8,6][9,3]}
9	0	{[8,3]}	{[1,12]}	{[1,3][7,9][8,3][9,3]}	{[1,12][7,10][8,7][9,3]}
10	0	{[8,3]}	{[1,13]}	{[1,3][7,9][8,3][9,3]}	{[1,13][7,10][8,7][9,3]}
11	0	{[8,3]}	{[1,14]}	{[1,3][7,9][8,3][9,3]}	{[1,14][7,10][8,7][9,4]}
12	10	{[8,3]}	{[1,15]}	{[1,3][7,9][8,3][9,3][10,3]}	{[1,15][7,10][8,7][9,4][10,3]}
13	0	{[8,3]}	{[1,16]}	{[1,3][7,9][8,3][9,3][10,3]}	{[1,16][7,10][8,7][9,4][10,3]}
14	0	{[8,3]}	{[1,17]}	{[1,3][4,1][7,9][8,3][9,3][10,3]}	{[1,17][4,2][7,10][8,7][9,9][10,3]}
15	0	{[8,3]}	{[1,18]}	{[1,3][4,1][7,9][8,3][9,3][10,3]}	{[1,18][4,2][7,10][8,7][9,10][10,3]}
16	10	{[8,3]}	{[1,19]}	{[1,3][4,1][7,9][8,3][9,3][10,3]}	{[1,19][4,2][7,10][8,7][9,10][10,3]}
17	0	{[8,3]}	{[1,20]}	{[1,3][4,1][7,9][8,3][9,3][10,3]}	{[1,20][4,2][7,10][8,7][9,10][10,4]}
18	10	{[8,3]}	{[1,21]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,21][4,2][7,10][8,7][9,10][10,4][11,3]}
19	30	{[8,3]}	{[1,22]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,22][4,2][7,10][8,7][9,10][10,4][11,3]}
20	0	{[8,3]}	{[1,23]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,23][4,2][7,10][8,7][9,10][10,6][11,3]}
21	0	{[8,3]}	{[1,24]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,24][4,2][7,10][8,7][9,10][10,7][11,3]}
22	10	{[8,3]}	{[1,25]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,25][4,2][7,10][8,7][9,10][10,7][11,3]}
23	0	{[8,3]}	{[1,26]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,26][4,2][7,10][8,7][9,10][10,7][11,4]}
24	0	{[8,3]}	{[1,27]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,27][2,4][4,2][7,15][8,7][9,10][10,7][11,4]}
25	0	{[8,3]}	{[1,28]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,28][2,4][4,2][7,15][8,7][9,10][10,7][11,4]}
26	10	{[8,3]}	{[1,29]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,29][2,4][4,2][7,15][8,7][9,10][10,7][11,6]}
27	0	{[8,3]}	{[1,30]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,30][2,4][4,2][7,15][8,7][9,10][10,7][11,7]}
28	0	{[8,3]}	{[1,31]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,31][2,4][4,2][7,15][8,7][9,10][10,7][11,7]}
29	0	{[8,3]}	{[1,32]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,32][2,4][4,2][7,16][8,7][9,10][10,7][11,7]}
30	0	{[8,3]}	{[1,33]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,33][2,4][4,4][7,16][8,7][9,15][10,7][11,7]}

…					
140	260	{[8,3]}	{[1,143]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,143][2,16][3,12][4,14][5,12][6,12][7,51][8,42][9,48][10,37][11,43]}
141	230	{[8,3]}	{[1,144]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,144][2,16][3,12][4,14][5,12][6,12][7,51][8,43][9,48][10,37][11,43]}
142	261	{[8,3]}	{[1,145]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,145][2,16][3,12][4,14][5,12][6,12][7,51][8,43][9,48][10,37][11,43]}
…					
542	951	{[8,3]}	{[1,545]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,545][2,56][3,48][4,52][5,54][6,58][7,186][8,145][9,163][10,162][11,171]}
543	1252	{[8,3]}	{[1,546]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,546][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,162][11,171]}
544	982	{[8,3]}	{[1,547]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,547][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,162][11,171]}
545	971	{[8,3]}	{[1,548]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,548][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,163][11,171]}
546	971	{[8,3]}	{[1,549]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,549][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
547	972	{[8,3]}	{[1,550]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,550][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
548	981	{[8,3]}	{[1,551]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,551][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,165][11,174]}
549	982	{[8,3]}	{[1,552]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,552][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,174]}
550	981	{[8,3]}	{[1,553]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,553][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,174]}
551	1002	{[8,3]}	{[1,554]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,554][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,175]}
552	961	{[8,3]}	{[1,555]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,555][2,56][3,48][4,52][5,58][6,60][7,187][8,145][9,163][10,168][11,180]}
553	981	{[8,3]}	{[1,556]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,556][2,56][3,48][4,52][5,58][6,60][7,187][8,145][9,163][10,168][11,181]}
554	982	{[8,3]}	{[1,557]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,557][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,171][11,181]}
555	991	{[8,3]}	{[1,558]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,558][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,181]}
556	992	{[8,3]}	{[1,559]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,559][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
557	1011	{[8,3]}	{[1,560]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
558	982	{[8,3]}	{[1,561]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,561][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,174][11,183]}
559	1021	{[8,3]}	{[1,562]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,562][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
560	992	{[8,3]}	{[1,563]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,563][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
561	1011	{[8,3]}	{[1,564]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,564][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,184]}
562	981	{[8,3]}	{[1,565]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,565][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,189]}
563	1001	{[8,3]}	{[1,566]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,566][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,190]}
564	1011	{[8,3]}	{[1,567]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,567][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,195]}
565	1042	{[8,3]}	{[1,568]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,568][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,196]}
566	1262	{[8,3]}	{[1,569]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,569][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,198]}
567	1021	{[8,3]}	{[1,570]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,570][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
568	1022	{[8,3]}	{[1,571]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,571][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}

…

16551	792	{[8,21]}	{[1,541]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,541][2,56][3,48][4,52][5,54][6,54][7,180][8,145][9,163][10,160][11,169]}
16552	791	{[8,21]}	{[1,542]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,542][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,160][11,169]}
16553	811	{[8,21]}	{[1,543]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,543][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
16554	801	{[8,21]}	{[1,544]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,544][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
16555	831	{[8,21]}	{[1,545]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,545][2,56][3,48][4,52][5,54][6,58][7,186][8,145][9,163][10,162][11,171]}
16556	831	{[8,21]}	{[1,546]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,546][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,162][11,171]}
16557	812	{[8,21]}	{[1,547]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,547][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,162][11,171]}
16558	831	{[8,21]}	{[1,548]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,548][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,163][11,171]}
16559	821	{[8,21]}	{[1,549]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,549][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
16560	811	{[8,21]}	{[1,550]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,550][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
16561	821	{[8,21]}	{[1,551]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,551][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,165][11,174]}
16562	832	{[8,21]}	{[1,552]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,552][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,174]}
16563	811	{[8,21]}	{[1,553]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,553][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,174]}
16564	831	{[8,21]}	{[1,554]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,554][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,175]}
16565	861	{[8,21]}	{[1,555]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,555][2,56][3,48][4,52][5,58][6,60][7,187][8,145][9,163][10,168][11,180]}
16566	851	{[8,21]}	{[1,556]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,556][2,56][3,48][4,52][5,58][6,60][7,187][8,145][9,163][10,168][11,181]}
16567	882	{[8,21]}	{[1,557]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,557][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,171][11,181]}
16568	861	{[8,21]}	{[1,558]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,558][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,181]}
16569	851	{[8,21]}	{[1,559]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,559][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
16570	1102	{[8,21]}	{[1,560]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
16571	861	{[8,21]}	{[1,561]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,561][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,174][11,183]}
16572	851	{[8,21]}	{[1,562]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,562][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
16573	851	{[8,21]}	{[1,563]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,563][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
16574	872	{[8,21]}	{[1,564]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,564][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,184]}
16575	861	{[8,21]}	{[1,565]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,565][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,189]}
16576	871	{[8,21]}	{[1,566]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,566][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,190]}
16577	871	{[8,21]}	{[1,567]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,567][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,195]}
16578	862	{[8,21]}	{[1,568]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,568][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,196]}
16579	881	{[8,21]}	{[1,569]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,569][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,198]}
16580	881	{[8,21]}	{[1,570]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,570][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
16581	861	{[8,21]}	{[1,571]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,571][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}

…					
32373	631	{[8,63]}	{[1,568]}	{[1,215][2,23][3,19]...[9,72][10,58][11,66]}	{[1,568][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,196]}
32374	621	{[8,63]}	{[1,569]}	{[1,215][2,23][3,19]...[9,72][10,58][11,66]}	{[1,569][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,198]}
32375	621	{[8,63]}	{[1,570]}	{[1,215][2,23][3,19]...[9,72][10,58][11,66]}	{[1,570][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
32376	611	{[8,63]}	{[1,571]}	{[1,215][2,23][3,19]...[9,72][10,58][11,66]}	{[1,571][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
…					
36574	802	{[8,30]}	{[1,558]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,558][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,181]}
36575	811	{[8,30]}	{[1,559]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,559][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
36576	811	{[8,30]}	{[1,560]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
36577	801	{[8,30]}	{[1,561]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,561][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,174][11,183]}
36578	811	{[8,30]}	{[1,562]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,562][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
36579	791	{[8,30]}	{[1,563]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,563][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
36580	802	{[8,30]}	{[1,564]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,564][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,184]}
36581	801	{[8,30]}	{[1,565]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,565][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,189]}
36582	811	{[8,30]}	{[1,566]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,566][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,190]}
36583	821	{[8,30]}	{[1,567]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,567][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,195]}
36584	801	{[8,30]}	{[1,568]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,568][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,196]}
36585	811	{[8,30]}	{[1,569]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,569][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,198]}
36586	802	{[8,30]}	{[1,570]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,570][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
36587	801	{[8,30]}	{[1,571]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,571][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
…					
47037	0	{[8,138]}	{[1,562]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,562][2,56][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
47038	10	{[8,138]}	{[1,563]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,563][2,56][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
47039	0	{[8,138]}	{[1,564]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,564][2,56][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,184]}
47040	0	{[8,138]}	{[1,565]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,565][2,56][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,189]}
47041	0	{[8,138]}	{[1,566]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,566][2,56][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,190]}
47042	0	{[8,138]}	{[1,567]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,567][2,56][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,195]}
47043	10	{[8,138]}	{[1,568]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,568][2,56][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,196]}
47044	0	{[8,138]}	{[1,569]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,569][2,56][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,198]}
47045	0	{[8,138]}	{[1,570]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,570][2,56][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
47046	0	{[8,138]}	{[1,571]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,571][2,56][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
47047	120	{[8,30]}	{[1,112]}	{[1,111][8,30]}	{[1,112][8,30]}

Appendix B

First Search Result of Ping Xie's Algorithm in Experiment 2

1	10	{[7,3]}	{[8,3]}	{[1,1][7,3][8,3]}	{[1,3][7,4][8,3]}
2	150	{[7,3]}	{[8,12]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,51][2,4][3,6][4,4][5,4][6,6][7,25][8,12][9,22][10,9][11,19]}
3	190	{[7,3]}	{[8,21]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,81][2,10][3,6][4,8][5,6][6,6][7,34][8,21][9,30][10,19][11,25]}
4	240	{[7,3]}	{[8,30]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,111][2,10][3,12][4,10][5,6][6,12][7,46][8,30][9,40][10,25][11,31]}
5	321	{[7,3]}	{[8,39]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,135][2,14][3,12][4,14][5,12][6,12][7,48][8,39][9,48][10,37][11,40]}
6	371	{[7,3]}	{[8,48]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,177][2,16][3,18][4,16][5,16][6,18][7,64][8,48][9,61][10,45][11,55]}
7	420	{[7,3]}	{[8,57]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,201][2,22][3,18][4,20][5,18][6,18][7,70][8,57][9,66][10,55][11,61]}
8	571	{[7,3]}	{[8,66]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,237][2,22][3,24][4,22][5,22][6,24][7,82][8,66][9,79][10,63][11,73]}
9	601	{[7,3]}	{[8,75]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,255][2,26][3,24][4,26][5,24][6,24][7,84][8,75][9,84][10,73][11,76]}
10	671	{[7,3]}	{[8,84]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,297][2,28][3,30][4,28][5,28][6,30][7,100][8,84][9,97][10,81][11,91]}
11	661	{[7,3]}	{[8,93]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,315][2,32][3,30][4,28][5,30][6,30][7,102][8,93][9,100][10,85][11,97]}
12	731	{[7,3]}	{[8,102]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,351][2,34][3,36][4,34][5,34][6,36][7,115][8,102][9,112][10,99][11,109]}
13	762	{[7,3]}	{[8,111]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,381][2,40][3,36][4,38][5,36][6,36][7,124][8,111][9,120][10,109][11,115]}
14	791	{[7,3]}	{[8,120]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,411][2,40][3,42][4,40][5,40][6,42][7,133][8,120][9,130][10,117][11,127]}
15	831	{[7,3]}	{[8,129]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,441][2,46][3,42][4,44][5,42][6,42][7,142][8,129][9,138][10,127][11,133]}
16	941	{[7,3]}	{[8,138]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,477][2,46][3,48][4,46][5,46][6,48][7,154][8,138][9,151][10,135][11,145]}
17	130	{[7,15]}	{[8,138]}	{[1,27][3,3][5,3][6,1][7,15][8,10][9,15][11,6]}	{[1,51][3,6][5,4][6,6][7,25][8,12][9,22][11,19]}
18	150	{[7,15]}	{[8,12]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,81][2,10][3,6][4,8][5,6][6,6][7,34][8,21][9,30][10,19][11,25]}
19	180	{[7,15]}	{[8,21]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,111][2,10][3,12][4,10][5,6][6,12][7,46][8,30][9,40][10,25][11,31]}
20	271	{[7,15]}	{[8,30]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,135][2,14][3,12][4,14][5,12][6,12][7,48][8,39][9,48][10,37][11,40]}
21	340	{[7,15]}	{[8,39]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,177][2,16][3,18][4,16][5,16][6,18][7,64][8,48][9,61][10,45][11,55]}
22	391	{[7,15]}	{[8,48]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,201][2,22][3,18][4,20][5,18][6,18][7,70][8,57][9,66][10,55][11,61]}
23	771	{[7,15]}	{[8,57]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,237][2,22][3,24][4,22][5,22][6,24][7,82][8,66][9,79][10,63][11,73]}
24	551	{[7,15]}	{[8,66]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,255][2,26][3,24][4,26][5,24][6,24][7,84][8,75][9,84][10,73][11,76]}
25	631	{[7,15]}	{[8,75]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,297][2,28][3,30][4,28][5,28][6,30][7,100][8,84][9,97][10,81][11,91]}
26	631	{[7,15]}	{[8,84]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,315][2,32][3,30][4,28][5,30][6,30][7,102][8,93][9,100][10,85][11,97]}
27	691	{[7,15]}	{[8,93]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,351][2,34][3,36][4,34][5,34][6,36][7,115][8,102][9,112][10,99][11,109]}
28	731	{[7,15]}	{[8,102]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,381][2,40][3,36][4,38][5,36][6,36][7,124][8,111][9,120][10,109][11,115]}
29	771	{[7,15]}	{[8,111]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,411][2,40][3,42][4,40][5,40][6,42][7,133][8,120][9,130][10,117][11,127]}
30	801	{[7,15]}	{[8,120]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,441][2,46][3,42][4,44][5,42][6,42][7,142][8,129][9,138][10,127][11,133]}

...					
71	701	{[1,1][7,3][8,3]}	{[1,424]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,424][2,44][3,42][4,40][5,40][6,42][7,138][8,127][9,133][10,117][11,129]}
72	731	{[1,1][7,3][8,3]}	{[1,430]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,430][2,44][3,42][4,40][5,42][6,42][7,138][8,127][9,136][10,120][11,130]}
73	741	{[1,1][7,3][8,3]}	{[1,436]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,436][2,46][3,42][4,40][5,42][6,42][7,141][8,127][9,136][10,121][11,133]}
74	771	{[1,1][7,3][8,3]}	{[1,442]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,442][2,46][3,42][4,44][5,42][6,42][7,142][8,129][9,138][10,127][11,133]}
75	792	{[1,1][7,3][8,3]}	{[1,448]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,448][2,46][3,42][4,46][5,42][6,46][7,148][8,130][9,141][10,127][11,135]}
76	821	{[1,1][7,3][8,3]}	{[1,454]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,454][2,46][3,42][4,46][5,42][6,46][7,148][8,133][9,142][10,129][11,135]}
77	831	{[1,1][7,3][8,3]}	{[1,460]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,460][2,46][3,46][4,46][5,42][6,46][7,150][8,135][9,148][10,130][11,135]}
78	842	{[1,1][7,3][8,3]}	{[1,466]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,466][2,46][3,46][4,46][5,42][6,48][7,151][8,135][9,148][10,133][11,138]}
79	841	{[1,1][7,3][8,3]}	{[1,472]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,472][2,46][3,46][4,46][5,42][6,48][7,154][8,135][9,150][10,133][11,139]}
80	861	{[1,1][7,3][8,3]}	{[1,478]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,478][2,46][3,48][4,46][5,46][6,48][7,154][8,138][9,151][10,135][11,145]}
81	871	{[1,1][7,3][8,3]}	{[1,484]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,484][2,46][3,48][4,46][5,46][6,48][7,154][8,139][9,154][10,135][11,147]}
82	851	{[1,1][7,3][8,3]}	{[1,490]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,490][2,50][3,48][4,46][5,48][6,48][7,156][8,145][9,154][10,138][11,148]}
83	1152	{[1,1][7,3][8,3]}	{[1,496]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,496][2,52][3,48][4,46][5,48][6,48][7,159][8,145][9,154][10,139][11,151]}
84	891	{[1,1][7,3][8,3]}	{[1,502]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,502][2,52][3,48][4,52][5,48][6,48][7,160][8,145][9,159][10,145][11,151]}
85	912	{[1,1][7,3][8,3]}	{[1,508]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,508][2,52][3,48][4,52][5,48][6,52][7,166][8,145][9,160][10,147][11,153]}
86	901	{[1,1][7,3][8,3]}	{[1,514]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,514][2,52][3,48][4,52][5,48][6,54][7,166][8,145][9,163][10,148][11,156]}
87	912	{[1,1][7,3][8,3]}	{[1,520]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,520][2,52][3,48][4,52][5,48][6,54][7,168][8,145][9,163][10,151][11,157]}
88	921	{[1,1][7,3][8,3]}	{[1,526]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,526][2,52][3,48][4,52][5,54][6,54][7,169][8,145][9,163][10,156][11,163]}
89	952	{[1,1][7,3][8,3]}	{[1,532]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,532][2,54][3,48][4,52][5,54][6,54][7,175][8,145][9,163][10,157][11,165]}
90	941	{[1,1][7,3][8,3]}	{[1,538]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,538][2,56][3,48][4,52][5,54][6,54][7,180][8,145][9,163][10,160][11,166]}
91	951	{[1,1][7,3][8,3]}	{[1,544]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,544][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
92	971	{[1,1][7,3][8,3]}	{[1,550]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,550][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
93	982	{[1,1][7,3][8,3]}	{[1,560]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
...					
155	711	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,418]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,418][2,40][3,42][4,40][5,40][6,42][7,136][8,121][9,132][10,117][11,127]}
156	1021	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,424]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,424][2,44][3,42][4,40][5,40][6,42][7,138][8,127][9,133][10,117][11,129]}
157	751	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,430]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,430][2,44][3,42][4,40][5,42][6,42][7,138][8,127][9,136][10,120][11,130]}
158	751	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,436]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,436][2,46][3,42][4,40][5,42][6,42][7,141][8,127][9,136][10,121][11,133]}
159	792	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,442]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,442][2,46][3,42][4,44][5,42][6,42][7,142][8,129][9,138][10,127][11,133]}
160	821	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,448]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,448][2,46][3,42][4,46][5,42][6,46][7,148][8,130][9,141][10,127][11,135]}
161	821	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,454]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,454][2,46][3,42][4,46][5,42][6,46][7,148][8,133][9,142][10,129][11,135]}
162	841	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,460]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,460][2,46][3,46][4,46][5,42][6,46][7,150][8,135][9,148][10,130][11,135]}

…					
163	852	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,466]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,466][2,46][3,46][4,46][5,42][6,48][7,151][8,135][9,148][10,133][11,138]}
164	861	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,472]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,472][2,46][3,46][4,46][5,42][6,48][7,154][8,135][9,150][10,133][11,139]}
165	891	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,478]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,478][2,46][3,48][4,46][5,46][6,48][7,154][8,138][9,151][10,135][11,145]}
166	892	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,484]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,484][2,46][3,48][4,46][5,46][6,48][7,154][8,139][9,154][10,135][11,147]}
167	871	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,490]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,490][2,50][3,48][4,46][5,48][6,48][7,156][8,145][9,154][10,138][11,148]}
168	921	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,496]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,496][2,52][3,48][4,46][5,48][6,48][7,159][8,145][9,154][10,139][11,151]}
169	921	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,502]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,502][2,52][3,48][4,52][5,48][6,48][7,160][8,145][9,159][10,145][11,151]}
170	911	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,508]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,508][2,52][3,48][4,52][5,48][6,52][7,166][8,145][9,160][10,147][11,153]}
171	912	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,514]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,514][2,52][3,48][4,52][5,48][6,54][7,166][8,145][9,163][10,148][11,156]}
172	941	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,520]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,520][2,52][3,48][4,52][5,48][6,54][7,168][8,145][9,163][10,151][11,157]}
173	941	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,526]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,526][2,52][3,48][4,52][5,54][6,54][7,169][8,145][9,163][10,156][11,163]}
174	951	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,532]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,532][2,54][3,48][4,52][5,54][6,54][7,175][8,145][9,163][10,157][11,165]}
175	951	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,538]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,538][2,56][3,48][4,52][5,54][6,54][7,180][8,145][9,163][10,160][11,166]}
176	952	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,544]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,544][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
177	981	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,550]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,550][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
178	1242	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,560]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
...					
2149	651	{[1,45][2,5]...[10,12][11,18]}	{[1,400]}	{[1,45][2,5]...[10,12][11,18]}	{[1,400][2,40][3,40][4,40][5,36][6,42][7,130][8,117][9,130][10,112][11,120]}
2150	651	{[1,45][2,5]...[10,12][11,18]}	{[1,406]}	{[1,45][2,5]...[10,12][11,18]}	{[1,406][2,40][3,40][4,40][5,36][6,42][7,132][8,117][9,130][10,115][11,121]}
2151	691	{[1,45][2,5]...[10,12][11,18]}	{[1,412]}	{[1,45][2,5]...[10,12][11,18]}	{[1,412][2,40][3,42][4,40][5,40][6,42][7,133][8,120][9,130][10,117][11,127]}
2152	681	{[1,45][2,5]...[10,12][11,18]}	{[1,418]}	{[1,45][2,5]...[10,12][11,18]}	{[1,418][2,40][3,42][4,40][5,40][6,42][7,136][8,121][9,132][10,117][11,127]}
2153	711	{[1,45][2,5]...[10,12][11,18]}	{[1,424]}	{[1,45][2,5]...[10,12][11,18]}	{[1,424][2,44][3,42][4,40][5,40][6,42][7,138][8,127][9,133][10,117][11,129]}
2154	691	{[1,45][2,5]...[10,12][11,18]}	{[1,430]}	{[1,45][2,5]...[10,12][11,18]}	{[1,430][2,44][3,42][4,40][5,42][6,42][7,138][8,127][9,136][10,120][11,130]}
2155	701	{[1,45][2,5]...[10,12][11,18]}	{[1,436]}	{[1,45][2,5]...[10,12][11,18]}	{[1,436][2,46][3,42][4,40][5,42][6,42][7,141][8,127][9,136][10,121][11,133]}
2156	691	{[1,45][2,5]...[10,12][11,18]}	{[1,442]}	{[1,45][2,5]...[10,12][11,18]}	{[1,442][2,46][3,42][4,44][5,42][6,42][7,142][8,129][9,138][10,127][11,133]}
2157	711	{[1,45][2,5]...[10,12][11,18]}	{[1,448]}	{[1,45][2,5]...[10,12][11,18]}	{[1,448][2,46][3,42][4,46][5,42][6,46][7,148][8,130][9,141][10,127][11,135]}
2158	701	{[1,45][2,5]...[10,12][11,18]}	{[1,454]}	{[1,45][2,5]...[10,12][11,18]}	{[1,454][2,46][3,42][4,46][5,42][6,46][7,148][8,133][9,142][10,129][11,135]}
2159	711	{[1,45][2,5]...[10,12][11,18]}	{[1,460]}	{[1,45][2,5]...[10,12][11,18]}	{[1,460][2,46][3,46][4,46][5,42][6,46][7,150][8,135][9,148][10,130][11,135]}
2160	721	{[1,45][2,5]...[10,12][11,18]}	{[1,466]}	{[1,45][2,5]...[10,12][11,18]}	{[1,466][2,46][3,46][4,46][5,42][6,48][7,151][8,135][9,148][10,133][11,138]}
2161	761	{[1,45][2,5]...[10,12][11,18]}	{[1,472]}	{[1,45][2,5]...[10,12][11,18]}	{[1,472][2,46][3,46][4,46][5,42][6,48][7,154][8,135][9,150][10,133][11,139]}
2162	771	{[1,45][2,5]...[10,12][11,18]}	{[1,478]}	{[1,45][2,5]...[10,12][11,18]}	{[1,478][2,46][3,48][4,46][5,46][6,48][7,154][8,138][9,151][10,135][11,145]}

…					
2163	782	{[1,45][2,5]...[10,12][11,18]}	{[1,484]}	{[1,45][2,5]...[10,12][11,18]}	{[1,484][2,46][3,48][4,46][5,46][6,48][7,154][8,139][9,154][10,135][11,147]}
2164	801	{[1,45][2,5]...[10,12][11,18]}	{[1,490]}	{[1,45][2,5]...[10,12][11,18]}	{[1,490][2,50][3,48][4,46][5,48][6,48][7,156][8,145][9,154][10,138][11,148]}
2165	801	{[1,45][2,5]...[10,12][11,18]}	{[1,496]}	{[1,45][2,5]...[10,12][11,18]}	{[1,496][2,52][3,48][4,46][5,48][6,48][7,159][8,145][9,154][10,139][11,151]}
2166	1082	{[1,45][2,5]...[10,12][11,18]}	{[1,502]}	{[1,45][2,5]...[10,12][11,18]}	{[1,502][2,52][3,48][4,52][5,48][6,48][7,160][8,145][9,159][10,145][11,151]}
...					
5139	490	{[1,225][2,23]...[10,66][11,72]}	{[1,478]}	{[1,225][2,23]...[10,66][11,72]}	{[1,478][2,46][3,48][4,46][5,46][6,48][7,154][8,138][9,151][10,135][11,145]}
5140	501	{[1,225][2,23]...[10,66][11,72]}	{[1,484]}	{[1,225][2,23]...[10,66][11,72]}	{[1,484][2,46][3,48][4,46][5,46][6,48][7,154][8,139][9,154][10,135][11,147]}
5141	531	{[1,225][2,23]...[10,66][11,72]}	{[1,490]}	{[1,225][2,23]...[10,66][11,72]}	{[1,490][2,50][3,48][4,46][5,48][6,48][7,156][8,145][9,154][10,138][11,148]}
5142	521	{[1,225][2,23]...[10,66][11,72]}	{[1,496]}	{[1,225][2,23]...[10,66][11,72]}	{[1,496][2,52][3,48][4,46][5,48][6,48][7,159][8,145][9,154][10,139][11,151]}
5143	511	{[1,225][2,23]...[10,66][11,72]}	{[1,502]}	{[1,225][2,23]...[10,66][11,72]}	{[1,502][2,52][3,48][4,52][5,48][6,48][7,160][8,145][9,159][10,145][11,151]}
5144	531	{[1,225][2,23]...[10,66][11,72]}	{[1,508]}	{[1,225][2,23]...[10,66][11,72]}	{[1,508][2,52][3,48][4,52][5,48][6,52][7,166][8,145][9,160][10,147][11,153]}
5145	560	{[1,225][2,23]...[10,66][11,72]}	{[1,514]}	{[1,225][2,23]...[10,66][11,72]}	{[1,514][2,52][3,48][4,52][5,48][6,54][7,166][8,145][9,163][10,148][11,156]}
5146	580	{[1,225][2,23]...[10,66][11,72]}	{[1,520]}	{[1,225][2,23]...[10,66][11,72]}	{[1,520][2,52][3,48][4,52][5,48][6,54][7,168][8,145][9,163][10,151][11,157]}
5147	581	{[1,225][2,23]...[10,66][11,72]}	{[1,526]}	{[1,225][2,23]...[10,66][11,72]}	{[1,526][2,52][3,48][4,52][5,54][6,54][7,169][8,145][9,163][10,156][11,163]}
5148	571	{[1,225][2,23]...[10,66][11,72]}	{[1,532]}	{[1,225][2,23]...[10,66][11,72]}	{[1,532][2,54][3,48][4,52][5,54][6,54][7,175][8,145][9,163][10,157][11,165]}
5149	591	{[1,225][2,23]...[10,66][11,72]}	{[1,538]}	{[1,225][2,23]...[10,66][11,72]}	{[1,538][2,56][3,48][4,52][5,54][6,54][7,180][8,145][9,163][10,160][11,166]}
5150	591	{[1,225][2,23]...[10,66][11,72]}	{[1,544]}	{[1,225][2,23]...[10,66][11,72]}	{[1,544][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
5151	601	{[1,225][2,23]...[10,66][11,72]}	{[1,550]}	{[1,225][2,23]...[10,66][11,72]}	{[1,550][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
5152	881	{[1,225][2,23]...[10,66][11,72]}	{[1,560]}	{[1,225][2,23]...[10,66][11,72]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
...					
6004	10	{[1,459][5,43]...[10,132][11,138]}	{[1,496]}	{[1,459][2,47]….[10,132][11,138]}	{[1,496][2,52][5,48][7,159][8,145][9,154][10,139][11,151]}
6005	0	{[1,459][5,43]...[10,132][11,138]}	{[1,502]}	{[1,459][2,47]…..[10,132][11,138]}	{[1,502][2,52][4,52][5,48][7,160][8,145][9,159][10,145][11,151]}
6006	0	{[1,459][5,43]...[10,132][11,138]}	{[1,508]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,508][2,52][4,52][5,48][6,52][7,166][8,145][9,160][10,147][11,153]}
6007	0	{[1,459][5,43]...[10,132][11,138]}	{[1,514]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,514][2,52][4,52][5,48][6,54][7,166][8,145][9,163][10,148][11,156]}
6008	0	{[1,459][5,43]...[10,132][11,138]}	{[1,520]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,520][2,52][4,52][5,48][6,54][7,168][8,145][9,163][10,151][11,157]}
6009	10	{[1,459][5,43]...[10,132][11,138]}	{[1,526]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,526][2,52][4,52][5,54][6,54][7,169][8,145][9,163][10,156][11,163]}
6010	0	{[1,459][5,43]...[10,132][11,138]}	{[1,532]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,532][2,54][4,52][5,54][6,54][7,175][8,145][9,163][10,157][11,165]}
6011	0	{[1,459][5,43]...[10,132][11,138]}	{[1,538]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,538][2,56][4,52][5,54][6,54][7,180][8,145][9,163][10,160][11,166]}
6012	0	{[1,459][5,43]...[10,132][11,138]}	{[1,544]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,544][2,56][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
6013	0	{[1,459][5,43]...[10,132][11,138]}	{[1,550]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,550][2,56][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
6014	0	{[1,459][5,43]...[10,132][11,138]}	{[1,560]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,560][2,56][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}

Appendix C

First Search Result of Proposed Algorithm in Experiment 1

1	0	{[8,3]}	{[1,4]}	{[1,3][8,3]}	{[1,4][8,3]}
2	10	{[8,3]}	{[1,5]}	{[1,3][7,9][8,3]}	{[1,5][7,9][8,3]}
3	0	{[8,3]}	{[1,6]}	{[1,3][7,9][8,3]}	{[1,6][7,10][8,3]}
4	0	{[8,3]}	{[1,7]}	{[1,3][7,9][8,3]}	{[1,7][7,10][8,3]}
5	0	{[8,3]}	{[1,8]}	{[1,3][7,9][8,3]}	{[1,8][7,10][8,4]}
6	0	{[8,3]}	{[1,9]}	{[1,3][7,9][8,3][9,3]}	{[1,9][7,10][8,4][9,3]}
7	0	{[8,3]}	{[1,10]}	{[1,3][7,9][8,3][9,3]}	{[1,10][7,10][8,4][9,3]}
8	10	{[8,3]}	{[1,11]}	{[1,3][7,9][8,3][9,3]}	{[1,11][7,10][8,6][9,3]}
9	0	{[8,3]}	{[1,12]}	{[1,3][7,9][8,3][9,3]}	{[1,12][7,10][8,7][9,3]}
10	0	{[8,3]}	{[1,13]}	{[1,3][7,9][8,3][9,3]}	{[1,13][7,10][8,7][9,3]}
11	0	{[8,3]}	{[1,14]}	{[1,3][7,9][8,3][9,3]}	{[1,14][7,10][8,7][9,4]}
12	0	{[8,3]}	{[1,15]}	{[1,3][7,9][8,3][9,3][10,3]}	{[1,15][7,10][8,7][9,4][10,3]}
13	0	{[8,3]}	{[1,16]}	{[1,3][7,9][8,3][9,3][10,3]}	{[1,16][7,10][8,7][9,4][10,3]}
14	0	{[8,3]}	{[1,17]}	{[1,3][4,1][7,9][8,3][9,3][10,3]}	{[1,17][4,2][7,10][8,7][9,9][10,3]}
15	0	{[8,3]}	{[1,18]}	{[1,3][4,1][7,9][8,3][9,3][10,3]}	{[1,18][4,2][7,10][8,7][9,10][10,3]}
16	20	{[8,3]}	{[1,19]}	{[1,3][4,1][7,9][8,3][9,3][10,3]}	{[1,19][4,2][7,10][8,7][9,10][10,3]}
17	0	{[8,3]}	{[1,20]}	{[1,3][4,1][7,9][8,3][9,3][10,3]}	{[1,20][4,2][7,10][8,7][9,10][10,4]}
18	0	{[8,3]}	{[1,21]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,21][4,2][7,10][8,7][9,10][10,4][11,3]}
19	0	{[8,3]}	{[1,22]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,22][4,2][7,10][8,7][9,10][10,4][11,3]}
20	0	{[8,3]}	{[1,23]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,23][4,2][7,10][8,7][9,10][10,6][11,3]}
21	0	{[8,3]}	{[1,24]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,24][4,2][7,10][8,7][9,10][10,7][11,3]}
22	0	{[8,3]}	{[1,25]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,25][4,2][7,10][8,7][9,10][10,7][11,3]}
23	0	{[8,3]}	{[1,26]}	{[1,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,26][4,2][7,10][8,7][9,10][10,7][11,4]}
24	0	{[8,3]}	{[1,27]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,27][2,4][4,2][7,15][8,7][9,10][10,7][11,4]}
25	0	{[8,3]}	{[1,28]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,28][2,4][4,2][7,15][8,7][9,10][10,7][11,4]}
26	0	{[8,3]}	{[1,29]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,29][2,4][4,2][7,15][8,7][9,10][10,7][11,6]}
27	0	{[8,3]}	{[1,30]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,30][2,4][4,2][7,15][8,7][9,10][10,7][11,7]}
28	10	{[8,3]}	{[1,31]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,31][2,4][4,2][7,15][8,7][9,10][10,7][11,7]}
29	0	{[8,3]}	{[1,32]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,32][2,4][4,2][7,16][8,7][9,10][10,7][11,7]}
30	0	{[8,3]}	{[1,33]}	{[1,3][2,3][4,1][7,9][8,3][9,3][10,3][11,3]}	{[1,33][2,4][4,4][7,16][8,7][9,15][10,7][11,7]}

…					
140	0	{[8,3]}	{[1,143]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,143][2,16][3,12][4,14][5,12][6,12][7,51][8,42][9,48][10,37][11,43]}
141	0	{[8,3]}	{[1,144]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,144][2,16][3,12][4,14][5,12][6,12][7,51][8,43][9,48][10,37][11,43]}
142	0	{[8,3]}	{[1,145]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,145][2,16][3,12][4,14][5,12][6,12][7,51][8,43][9,48][10,37][11,43]}
…					
542	0	{[8,3]}	{[1,545]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,545][2,56][3,48][4,52][5,54][6,58][7,186][8,145][9,163][10,162][11,171]}
543	0	{[8,3]}	{[1,546]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,546][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,162][11,171]}
544	0	{[8,3]}	{[1,547]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,547][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,162][11,171]}
545	0	{[8,3]}	{[1,548]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,548][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,163][11,171]}
546	0	{[8,3]}	{[1,549]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,549][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
547	0	{[8,3]}	{[1,550]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,550][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
548	0	{[8,3]}	{[1,551]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,551][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,165][11,174]}
549	10	{[8,3]}	{[1,552]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,552][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,174]}
550	0	{[8,3]}	{[1,553]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,553][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,174]}
551	0	{[8,3]}	{[1,554]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,554][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,175]}
552	10	{[8,3]}	{[1,555]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,555][2,56][3,48][4,52][5,58][6,60][7,187][8,145][9,163][10,168][11,180]}
553	0	{[8,3]}	{[1,556]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,556][2,56][3,48][4,52][5,58][6,60][7,187][8,145][9,163][10,168][11,181]}
554	0	{[8,3]}	{[1,557]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,557][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,171][11,181]}
555	10	{[8,3]}	{[1,558]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,558][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,181]}
556	0	{[8,3]}	{[1,559]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,559][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
557	0	{[8,3]}	{[1,560]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
558	0	{[8,3]}	{[1,561]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,561][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,174][11,183]}
559	10	{[8,3]}	{[1,562]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,562][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
560	0	{[8,3]}	{[1,563]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,563][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
561	0	{[8,3]}	{[1,564]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,564][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,184]}
562	10	{[8,3]}	{[1,565]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,565][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,189]}
563	0	{[8,3]}	{[1,566]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,566][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,190]}
564	0	{[8,3]}	{[1,567]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,567][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,195]}
565	10	{[8,3]}	{[1,568]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,568][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,196]}
566	0	{[8,3]}	{[1,569]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,569][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,198]}
567	0	{[8,3]}	{[1,570]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,570][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
568	0	{[8,3]}	{[1,571]}	{[1,3][2,3][3,1][4,1][5,1][6,1][7,9][8,3][9,3][10,3][11,3]}	{[1,571][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}

…

16551	0	{[8,21]}	{[1,541]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,541][2,56][3,48][4,52][5,54][6,54][7,180][8,145][9,163][10,160][11,169]}
16552	0	{[8,21]}	{[1,542]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,542][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,160][11,169]}
16553	20	{[8,21]}	{[1,543]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,543][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
16554	0	{[8,21]}	{[1,544]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,544][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
16555	10	{[8,21]}	{[1,545]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,545][2,56][3,48][4,52][5,54][6,58][7,186][8,145][9,163][10,162][11,171]}
16556	0	{[8,21]}	{[1,546]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,546][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,162][11,171]}
16557	10	{[8,21]}	{[1,547]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,547][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,162][11,171]}
16558	0	{[8,21]}	{[1,548]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,548][2,56][3,48][4,52][5,54][6,58][7,187][8,145][9,163][10,163][11,171]}
16559	0	{[8,21]}	{[1,549]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,549][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
16560	0	{[8,21]}	{[1,550]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,550][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
16561	0	{[8,21]}	{[1,551]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,551][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,165][11,174]}
16562	0	{[8,21]}	{[1,552]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,552][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,174]}
16563	0	{[8,21]}	{[1,553]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,553][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,174]}
16564	0	{[8,21]}	{[1,554]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,554][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,166][11,175]}
16565	20	{[8,21]}	{[1,555]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,555][2,56][3,48][4,52][5,58][6,60][7,187][8,145][9,163][10,168][11,180]}
16566	0	{[8,21]}	{[1,556]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,556][2,56][3,48][4,52][5,58][6,60][7,187][8,145][9,163][10,168][11,181]}
16567	20	{[8,21]}	{[1,557]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,557][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,171][11,181]}
16568	10	{[8,21]}	{[1,558]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,558][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,181]}
16569	0	{[8,21]}	{[1,559]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,559][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
16570	0	{[8,21]}	{[1,560]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
16571	0	{[8,21]}	{[1,561]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,561][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,174][11,183]}
16572	0	{[8,21]}	{[1,562]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,562][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
16573	10	{[8,21]}	{[1,563]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,563][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
16574	10	{[8,21]}	{[1,564]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,564][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,184]}
16575	0	{[8,21]}	{[1,565]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,565][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,189]}
16576	0	{[8,21]}	{[1,566]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,566][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,190]}
16577	0	{[8,21]}	{[1,567]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,567][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,195]}
16578	0	{[8,21]}	{[1,568]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,568][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,196]}
16579	0	{[8,21]}	{[1,569]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,569][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,198]}
16580	11	{[8,21]}	{[1,570]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,570][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
16581	0	{[8,21]}	{[1,571]}	{[1,81][2,11][3,7][4,11]...[8,21][9,33][10,21][11,30]}	{[1,571][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}

…					
32373	0	{[8,63]}	{[1,568]}	{[1,215][2,23][3,19]...[9,72][10,58][11,66]}	{[1,568][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,196]}
32374	0	{[8,63]}	{[1,569]}	{[1,215][2,23][3,19]...[9,72][10,58][11,66]}	{[1,569][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,198]}
32375	0	{[8,63]}	{[1,570]}	{[1,215][2,23][3,19]...[9,72][10,58][11,66]}	{[1,570][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
32376	10	{[8,63]}	{[1,571]}	{[1,215][2,23][3,19]...[9,72][10,58][11,66]}	{[1,571][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
…					
36574	0	{[8,30]}	{[1,558]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,558][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,181]}
36575	20	{[8,30]}	{[1,559]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,559][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
36576	0	{[8,30]}	{[1,560]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
36577	0	{[8,30]}	{[1,561]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,561][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,174][11,183]}
36578	0	{[8,30]}	{[1,562]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,562][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
36579	10	{[8,30]}	{[1,563]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,563][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
36580	0	{[8,30]}	{[1,564]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,564][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,184]}
36581	0	{[8,30]}	{[1,565]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,565][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,189]}
36582	0	{[8,30]}	{[1,566]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,566][2,56][3,48][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,190]}
36583	0	{[8,30]}	{[1,567]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,567][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,195]}
36584	0	{[8,30]}	{[1,568]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,568][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,196]}
36585	0	{[8,30]}	{[1,569]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,569][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,198]}
36586	0	{[8,30]}	{[1,570]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,570][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
36587	0	{[8,30]}	{[1,571]}	{[1,111][2,13][3,13]...[9,42][10,30][11,36]}	{[1,571][2,56][3,48][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
…					
47037	0	{[8,138]}	{[1,562]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,562][2,56][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
47038	10	{[8,138]}	{[1,563]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,563][2,56][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,183]}
47039	0	{[8,138]}	{[1,564]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,564][2,56][4,52][5,60][6,60][7,187][8,145][9,163][10,175][11,184]}
47040	0	{[8,138]}	{[1,565]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,565][2,56][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,189]}
47041	0	{[8,138]}	{[1,566]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,566][2,56][4,52][5,60][6,62][7,187][8,145][9,163][10,175][11,190]}
47042	10	{[8,138]}	{[1,567]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,567][2,56][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,195]}
47043	0	{[8,138]}	{[1,568]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,568][2,56][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,196]}
47044	0	{[8,138]}	{[1,569]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,569][2,56][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,198]}
47045	0	{[8,138]}	{[1,570]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,570][2,56][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
47046	10	{[8,138]}	{[1,571]}	{[1,477][2,49][4,47]...[9,153][10,138][11,147]}	{[1,571][2,56][4,52][5,60][6,64][7,187][8,145][9,163][10,175][11,199]}
47047	120	{[8,30]}	{[1,112]}	{[1,111][8,30]}	{[1,112][8,30]}

Appendix D
First Search Result of Proposed Algorithm in Experiment 2

1	10	{[7,3]}	{[8,3]}	{[1,1][7,3][8,3]}	{[1,3][7,4][8,3]}
2	170	{[7,3]}	{[8,12]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,51][2,4][3,6][4,4][5,4][6,6][7,25][8,12][9,22][10,9][11,19]}
3	200	{[7,3]}	{[8,21]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,81][2,10][3,6][4,8][5,6][6,6][7,34][8,21][9,30][10,19][11,25]}
4	411	{[7,3]}	{[8,30]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,111][2,10][3,12][4,10][5,6][6,12][7,46][8,30][9,40][10,25][11,31]}
5	401	{[7,3]}	{[8,39]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,135][2,14][3,12][4,14][5,12][6,12][7,48][8,39][9,48][10,37][11,40]}
6	410	{[7,3]}	{[8,48]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,177][2,16][3,18][4,16][5,16][6,18][7,64][8,48][9,61][10,45][11,55]}
7	451	{[7,3]}	{[8,57]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,201][2,22][3,18][4,20][5,18][6,18][7,70][8,57][9,66][10,55][11,61]}
8	571	{[7,3]}	{[8,66]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,237][2,22][3,24][4,22][5,22][6,24][7,82][8,66][9,79][10,63][11,73]}
9	601	{[7,3]}	{[8,75]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,255][2,26][3,24][4,26][5,24][6,24][7,84][8,75][9,84][10,73][11,76]}
10	601	{[7,3]}	{[8,84]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,297][2,28][3,30][4,28][5,28][6,30][7,100][8,84][9,97][10,81][11,91]}
11	611	{[7,3]}	{[8,93]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,315][2,32][3,30][4,28][5,30][6,30][7,102][8,93][9,100][10,85][11,97]}
12	681	{[7,3]}	{[8,102]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,351][2,34][3,36][4,34][5,34][6,36][7,115][8,102][9,112][10,99][11,109]}
13	711	{[7,3]}	{[8,111]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,381][2,40][3,36][4,38][5,36][6,36][7,124][8,111][9,120][10,109][11,115]}
14	741	{[7,3]}	{[8,120]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,411][2,40][3,42][4,40][5,40][6,42][7,133][8,120][9,130][10,117][11,127]}
15	731	{[7,3]}	{[8,129]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,441][2,46][3,42][4,44][5,42][6,42][7,142][8,129][9,138][10,127][11,133]}
16	801	{[7,3]}	{[8,138]}	{[1,1][2,1][3,1]...[9,3][10,3][11,3]}	{[1,477][2,46][3,48][4,46][5,46][6,48][7,154][8,138][9,151][10,135][11,145]}
17	131	{[7,15]}	{[8,138]}	{[1,27][3,3][5,3][6,1][7,15][8,10][9,15][11,6]}	{[1,51][3,6][5,4][6,6][7,25][8,12][9,22][11,19]}
18	140	{[7,15]}	{[8,12]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,81][2,10][3,6][4,8][5,6][6,6][7,34][8,21][9,30][10,19][11,25]}
19	210	{[7,15]}	{[8,21]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,111][2,10][3,12][4,10][5,6][6,12][7,46][8,30][9,40][10,25][11,31]}
20	281	{[7,15]}	{[8,30]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,135][2,14][3,12][4,14][5,12][6,12][7,48][8,39][9,48][10,37][11,40]}
21	361	{[7,15]}	{[8,39]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,177][2,16][3,18][4,16][5,16][6,18][7,64][8,48][9,61][10,45][11,55]}
22	390	{[7,15]}	{[8,48]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,201][2,22][3,18][4,20][5,18][6,18][7,70][8,57][9,66][10,55][11,61]}
23	471	{[7,15]}	{[8,57]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,237][2,22][3,24][4,22][5,22][6,24][7,82][8,66][9,79][10,63][11,73]}
24	530	{[7,15]}	{[8,66]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,255][2,26][3,24][4,26][5,24][6,24][7,84][8,75][9,84][10,73][11,76]}
25	590	{[7,15]}	{[8,75]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,297][2,28][3,30][4,28][5,28][6,30][7,100][8,84][9,97][10,81][11,91]}
26	631	{[7,15]}	{[8,84]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,315][2,32][3,30][4,28][5,30][6,30][7,102][8,93][9,100][10,85][11,97]}
27	641	{[7,15]}	{[8,93]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,351][2,34][3,36][4,34][5,34][6,36][7,115][8,102][9,112][10,99][11,109]}
28	651	{[7,15]}	{[8,102]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,381][2,40][3,36][4,38][5,36][6,36][7,124][8,111][9,120][10,109][11,115]}
29	691	{[7,15]}	{[8,111]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,411][2,40][3,42][4,40][5,40][6,42][7,133][8,120][9,130][10,117][11,127]}
30	691	{[7,15]}	{[8,120]}	{[1,27][2,5][3,3]...[9,15][10,10][11,6]}	{[1,441][2,46][3,42][4,44][5,42][6,42][7,142][8,129][9,138][10,127][11,133]}

...					
71	600	{[1,1][7,3][8,3]}	{[1,424]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,424][2,44][3,42][4,40][5,40][6,42][7,138][8,127][9,133][10,117][11,129]}
72	600	{[1,1][7,3][8,3]}	{[1,430]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,430][2,44][3,42][4,40][5,42][6,42][7,138][8,127][9,136][10,120][11,130]}
73	621	{[1,1][7,3][8,3]}	{[1,436]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,436][2,46][3,42][4,40][5,42][6,42][7,141][8,127][9,136][10,121][11,133]}
74	140	{[1,1][7,3][8,3]}	{[1,442]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,442][2,46][3,42][4,44][5,42][6,42][7,142][8,129][9,138][10,127][11,133]}
75	151	{[1,1][7,3][8,3]}	{[1,448]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,448][2,46][3,42][4,46][5,42][6,46][7,148][8,130][9,141][10,127][11,135]}
76	260	{[1,1][7,3][8,3]}	{[1,454]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,454][2,46][3,42][4,46][5,42][6,46][7,148][8,133][9,142][10,129][11,135]}
77	270	{[1,1][7,3][8,3]}	{[1,460]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,460][2,46][3,46][4,46][5,42][6,46][7,150][8,135][9,148][10,130][11,135]}
78	371	{[1,1][7,3][8,3]}	{[1,466]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,466][2,46][3,46][4,46][5,42][6,48][7,151][8,135][9,148][10,133][11,138]}
79	391	{[1,1][7,3][8,3]}	{[1,472]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,472][2,46][3,46][4,46][5,42][6,48][7,154][8,135][9,150][10,133][11,139]}
80	461	{[1,1][7,3][8,3]}	{[1,478]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,478][2,46][3,48][4,46][5,46][6,48][7,154][8,138][9,151][10,135][11,145]}
81	510	{[1,1][7,3][8,3]}	{[1,484]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,484][2,46][3,48][4,46][5,46][6,48][7,154][8,139][9,154][10,135][11,147]}
82	531	{[1,1][7,3][8,3]}	{[1,490]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,490][2,50][3,48][4,46][5,48][6,48][7,156][8,145][9,154][10,138][11,148]}
83	531	{[1,1][7,3][8,3]}	{[1,496]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,496][2,52][3,48][4,46][5,48][6,48][7,159][8,145][9,154][10,139][11,151]}
84	550	{[1,1][7,3][8,3]}	{[1,502]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,502][2,52][3,48][4,52][5,48][6,48][7,160][8,145][9,159][10,145][11,151]}
85	110	{[1,1][7,3][8,3]}	{[1,508]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,508][2,52][3,48][4,52][5,48][6,52][7,166][8,145][9,160][10,147][11,153]}
86	131	{[1,1][7,3][8,3]}	{[1,514]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,514][2,52][3,48][4,52][5,48][6,54][7,166][8,145][9,163][10,148][11,156]}
87	200	{[1,1][7,3][8,3]}	{[1,520]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,520][2,52][3,48][4,52][5,48][6,54][7,168][8,145][9,163][10,151][11,157]}
88	270	{[1,1][7,3][8,3]}	{[1,526]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,526][2,52][3,48][4,52][5,54][6,54][7,169][8,145][9,163][10,156][11,163]}
89	310	{[1,1][7,3][8,3]}	{[1,532]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,532][2,54][3,48][4,52][5,54][6,54][7,175][8,145][9,163][10,157][11,165]}
90	320	{[1,1][7,3][8,3]}	{[1,538]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,538][2,56][3,48][4,52][5,54][6,54][7,180][8,145][9,163][10,160][11,166]}
91	380	{[1,1][7,3][8,3]}	{[1,544]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,544][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
92	431	{[1,1][7,3][8,3]}	{[1,550]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,550][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
93	521	{[1,1][7,3][8,3]}	{[1,560]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
...					
155	0	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,418]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,418][2,40][3,42][4,40][5,40][6,42][7,136][8,121][9,132][10,117][11,127]}
156	10	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,424]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,424][2,44][3,42][4,40][5,40][6,42][7,138][8,127][9,133][10,117][11,129]}
157	20	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,430]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,430][2,44][3,42][4,40][5,42][6,42][7,138][8,127][9,136][10,120][11,130]}
158	0	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,436]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,436][2,46][3,42][4,40][5,42][6,42][7,141][8,127][9,136][10,121][11,133]}
159	10	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,442]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,442][2,46][3,42][4,44][5,42][6,42][7,142][8,129][9,138][10,127][11,133]}
160	40	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,448]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,448][2,46][3,42][4,46][5,42][6,46][7,148][8,130][9,141][10,127][11,135]}
161	20	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,454]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,454][2,46][3,42][4,46][5,42][6,46][7,148][8,133][9,142][10,129][11,135]}
162	0	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,460]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,460][2,46][3,46][4,46][5,42][6,46][7,150][8,135][9,148][10,130][11,135]}

…					
163	0	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,466]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,466][2,46][3,46][4,46][5,42][6,48][7,151][8,135][9,148][10,133][11,138]}
164	10	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,472]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,472][2,46][3,46][4,46][5,42][6,48][7,154][8,135][9,150][10,133][11,139]}
165	50	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,478]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,478][2,46][3,48][4,46][5,46][6,48][7,154][8,138][9,151][10,135][11,145]}
166	10	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,484]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,484][2,46][3,48][4,46][5,46][6,48][7,154][8,139][9,154][10,135][11,147]}
167	10	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,490]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,490][2,50][3,48][4,46][5,48][6,48][7,156][8,145][9,154][10,138][11,148]}
168	20	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,496]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,496][2,52][3,48][4,46][5,48][6,48][7,159][8,145][9,154][10,139][11,151]}
169	20	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,502]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,502][2,52][3,48][4,52][5,48][6,48][7,160][8,145][9,159][10,145][11,151]}
170	0	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,508]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,508][2,52][3,48][4,52][5,48][6,52][7,166][8,145][9,160][10,147][11,153]}
171	30	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,514]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,514][2,52][3,48][4,52][5,48][6,54][7,166][8,145][9,163][10,148][11,156]}
172	30	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,520]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,520][2,52][3,48][4,52][5,48][6,54][7,168][8,145][9,163][10,151][11,157]}
173	20	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,526]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,526][2,52][3,48][4,52][5,54][6,54][7,169][8,145][9,163][10,156][11,163]}
174	0	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,532]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,532][2,54][3,48][4,52][5,54][6,54][7,175][8,145][9,163][10,157][11,165]}
175	20	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,538]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,538][2,56][3,48][4,52][5,54][6,54][7,180][8,145][9,163][10,160][11,166]}
176	0	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,544]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,544][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
177	0	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,550]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,550][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
178	30	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,560]}	{[1,1][2,1][3,1]...[10,3][11,3]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
...					
2149	10	{[1,45][2,5]...[10,12][11,18]}	{[1,400]}	{[1,45][2,5]...[10,12][11,18]}	{[1,400][2,40][3,40][4,40][5,36][6,42][7,130][8,117][9,130][10,112][11,120]}
2150	10	{[1,45][2,5]...[10,12][11,18]}	{[1,406]}	{[1,45][2,5]...[10,12][11,18]}	{[1,406][2,40][3,40][4,40][5,36][6,42][7,132][8,117][9,130][10,115][11,121]}
2151	50	{[1,45][2,5]...[10,12][11,18]}	{[1,412]}	{[1,45][2,5]...[10,12][11,18]}	{[1,412][2,40][3,42][4,40][5,40][6,42][7,133][8,120][9,130][10,117][11,127]}
2152	20	{[1,45][2,5]...[10,12][11,18]}	{[1,418]}	{[1,45][2,5]...[10,12][11,18]}	{[1,418][2,40][3,42][4,40][5,40][6,42][7,136][8,121][9,132][10,117][11,127]}
2153	10	{[1,45][2,5]...[10,12][11,18]}	{[1,424]}	{[1,45][2,5]...[10,12][11,18]}	{[1,424][2,44][3,42][4,40][5,40][6,42][7,138][8,127][9,133][10,117][11,129]}
2154	20	{[1,45][2,5]...[10,12][11,18]}	{[1,430]}	{[1,45][2,5]...[10,12][11,18]}	{[1,430][2,44][3,42][4,40][5,42][6,42][7,138][8,127][9,136][10,120][11,130]}
2155	0	{[1,45][2,5]...[10,12][11,18]}	{[1,436]}	{[1,45][2,5]...[10,12][11,18]}	{[1,436][2,46][3,42][4,40][5,42][6,42][7,141][8,127][9,136][10,121][11,133]}
2156	10	{[1,45][2,5]...[10,12][11,18]}	{[1,442]}	{[1,45][2,5]...[10,12][11,18]}	{[1,442][2,46][3,42][4,44][5,42][6,42][7,142][8,129][9,138][10,127][11,133]}
2157	10	{[1,45][2,5]...[10,12][11,18]}	{[1,448]}	{[1,45][2,5]...[10,12][11,18]}	{[1,448][2,46][3,42][4,46][5,42][6,46][7,148][8,130][9,141][10,127][11,135]}
2158	10	{[1,45][2,5]...[10,12][11,18]}	{[1,454]}	{[1,45][2,5]...[10,12][11,18]}	{[1,454][2,46][3,42][4,46][5,42][6,46][7,148][8,133][9,142][10,129][11,135]}
2159	10	{[1,45][2,5]...[10,12][11,18]}	{[1,460]}	{[1,45][2,5]...[10,12][11,18]}	{[1,460][2,46][3,46][4,46][5,42][6,46][7,150][8,135][9,148][10,130][11,135]}
2160	0	{[1,45][2,5]...[10,12][11,18]}	{[1,466]}	{[1,45][2,5]...[10,12][11,18]}	{[1,466][2,46][3,46][4,46][5,42][6,48][7,151][8,135][9,148][10,133][11,138]}
2161	20	{[1,45][2,5]...[10,12][11,18]}	{[1,472]}	{[1,45][2,5]...[10,12][11,18]}	{[1,472][2,46][3,46][4,46][5,42][6,48][7,154][8,135][9,150][10,133][11,139]}
2162	30	{[1,45][2,5]...[10,12][11,18]}	{[1,478]}	{[1,45][2,5]...[10,12][11,18]}	{[1,478][2,46][3,48][4,46][5,46][6,48][7,154][8,138][9,151][10,135][11,145]}

…					
2163	10	{[1,45][2,5]...[10,12][11,18]}	{[1,484]}	{[1,45][2,5]...[10,12][11,18]}	{[1,484][2,46][3,48][4,46][5,46][6,48][7,154][8,139][9,154][10,135][11,147]}
2164	10	{[1,45][2,5]...[10,12][11,18]}	{[1,490]}	{[1,45][2,5]...[10,12][11,18]}	{[1,490][2,50][3,48][4,46][5,48][6,48][7,156][8,145][9,154][10,138][11,148]}
2165	10	{[1,45][2,5]...[10,12][11,18]}	{[1,496]}	{[1,45][2,5]...[10,12][11,18]}	{[1,496][2,52][3,48][4,46][5,48][6,48][7,159][8,145][9,154][10,139][11,151]}
2166	20	{[1,45][2,5]...[10,12][11,18]}	{[1,502]}	{[1,45][2,5]...[10,12][11,18]}	{[1,502][2,52][3,48][4,52][5,48][6,48][7,160][8,145][9,159][10,145][11,151]}
...					
5139	491	{[1,225][2,23]...[10,66][11,72]}	{[1,478]}	{[1,225][2,23]...[10,66][11,72]}	{[1,478][2,46][3,48][4,46][5,46][6,48][7,154][8,138][9,151][10,135][11,145]}
5140	80	{[1,225][2,23]...[10,66][11,72]}	{[1,484]}	{[1,225][2,23]...[10,66][11,72]}	{[1,484][2,46][3,48][4,46][5,46][6,48][7,154][8,139][9,154][10,135][11,147]}
5141	10	{[1,225][2,23]...[10,66][11,72]}	{[1,490]}	{[1,225][2,23]...[10,66][11,72]}	{[1,490][2,50][3,48][4,46][5,48][6,48][7,156][8,145][9,154][10,138][11,148]}
5142	0	{[1,225][2,23]...[10,66][11,72]}	{[1,496]}	{[1,225][2,23]...[10,66][11,72]}	{[1,496][2,52][3,48][4,46][5,48][6,48][7,159][8,145][9,154][10,139][11,151]}
5143	30	{[1,225][2,23]...[10,66][11,72]}	{[1,502]}	{[1,225][2,23]...[10,66][11,72]}	{[1,502][2,52][3,48][4,52][5,48][6,48][7,160][8,145][9,159][10,145][11,151]}
5144	0	{[1,225][2,23]...[10,66][11,72]}	{[1,508]}	{[1,225][2,23]...[10,66][11,72]}	{[1,508][2,52][3,48][4,52][5,48][6,52][7,166][8,145][9,160][10,147][11,153]}
5145	0	{[1,225][2,23]...[10,66][11,72]}	{[1,514]}	{[1,225][2,23]...[10,66][11,72]}	{[1,514][2,52][3,48][4,52][5,48][6,54][7,166][8,145][9,163][10,148][11,156]}
5146	20	{[1,225][2,23]...[10,66][11,72]}	{[1,520]}	{[1,225][2,23]...[10,66][11,72]}	{[1,520][2,52][3,48][4,52][5,48][6,54][7,168][8,145][9,163][10,151][11,157]}
5147	0	{[1,225][2,23]...[10,66][11,72]}	{[1,526]}	{[1,225][2,23]...[10,66][11,72]}	{[1,526][2,52][3,48][4,52][5,54][6,54][7,169][8,145][9,163][10,156][11,163]}
5148	10	{[1,225][2,23]...[10,66][11,72]}	{[1,532]}	{[1,225][2,23]...[10,66][11,72]}	{[1,532][2,54][3,48][4,52][5,54][6,54][7,175][8,145][9,163][10,157][11,165]}
5149	10	{[1,225][2,23]...[10,66][11,72]}	{[1,538]}	{[1,225][2,23]...[10,66][11,72]}	{[1,538][2,56][3,48][4,52][5,54][6,54][7,180][8,145][9,163][10,160][11,166]}
5150	40	{[1,225][2,23]...[10,66][11,72]}	{[1,544]}	{[1,225][2,23]...[10,66][11,72]}	{[1,544][2,56][3,48][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
5151	10	{[1,225][2,23]...[10,66][11,72]}	{[1,550]}	{[1,225][2,23]...[10,66][11,72]}	{[1,550][2,56][3,48][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
5152	10	{[1,225][2,23]...[10,66][11,72]}	{[1,560]}	{[1,225][2,23]...[10,66][11,72]}	{[1,560][2,56][3,48][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}
...					
6004	0	{[1,459][5,43]...[10,132][11,138]}	{[1,496]}	{[1,459][2,47]….[10,132][11,138]}	{[1,496][2,52][5,48][7,159][8,145][9,154][10,139][11,151]}
6005	10	{[1,459][5,43]...[10,132][11,138]}	{[1,502]}	{[1,459][2,47]…..[10,132][11,138]}	{[1,502][2,52][4,52][5,48][7,160][8,145][9,159][10,145][11,151]}
6006	0	{[1,459][5,43]...[10,132][11,138]}	{[1,508]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,508][2,52][4,52][5,48][6,52][7,166][8,145][9,160][10,147][11,153]}
6007	0	{[1,459][5,43]...[10,132][11,138]}	{[1,514]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,514][2,52][4,52][5,48][6,54][7,166][8,145][9,163][10,148][11,156]}
6008	0	{[1,459][5,43]...[10,132][11,138]}	{[1,520]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,520][2,52][4,52][5,48][6,54][7,168][8,145][9,163][10,151][11,157]}
6009	0	{[1,459][5,43]...[10,132][11,138]}	{[1,526]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,526][2,52][4,52][5,54][6,54][7,169][8,145][9,163][10,156][11,163]}
6010	0	{[1,459][5,43]...[10,132][11,138]}	{[1,532]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,532][2,54][4,52][5,54][6,54][7,175][8,145][9,163][10,157][11,165]}
6011	0	{[1,459][5,43]...[10,132][11,138]}	{[1,538]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,538][2,56][4,52][5,54][6,54][7,180][8,145][9,163][10,160][11,166]}
6012	0	{[1,459][5,43]...[10,132][11,138]}	{[1,544]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,544][2,56][4,52][5,54][6,54][7,181][8,145][9,163][10,162][11,169]}
6013	0	{[1,459][5,43]...[10,132][11,138]}	{[1,550]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,550][2,56][4,52][5,54][6,60][7,187][8,145][9,163][10,163][11,174]}
6014	0	{[1,459][5,43]...[10,132][11,138]}	{[1,560]}	{[1,459][2,47]…...[10,132][11,138]}	{[1,560][2,56][4,52][5,60][6,60][7,187][8,145][9,163][10,172][11,183]}

Appendix E

Cache Performance Results of Proposed Algorithm in Experiment 1

No. of time *left* event set changes	**No. of events in *left* event set**	**No.of iterations**	**No. of successful cached computation**
0	1	1	0
0	1	2	0
0	1	3	1
0	1	4	2
0	1	5	3
0	1	6	3
0	1	7	4
0	1	8	5
0	1	9	6
0	1	10	7
0	1	11	8
0	1	12	8
0	1	13	9
0	1	14	9
0	1	15	10
…			
50	1	161	16270
50	1	162	16271
50	1	163	16272
50	1	164	16273
50	1	165	16274
50	1	166	16275
50	1	167	16276
50	1	168	16277
50	1	169	16278
50	1	170	16279
50	1	171	16280
50	1	172	16281
50	1	173	16282
50	1	174	16283
50	1	175	16284
…			
153	1	84	45607
153	1	85	45608
153	1	86	45609
153	1	87	45610
153	1	88	45611
153	1	89	45612
153	1	90	45613
153	1	91	45614
153	1	92	45615
153	1	93	45616
153	1	94	45617
154	1	1	45617
155	1	1	45617

Appendix F
Cache Performance Results of Proposed Algorithm in Experiment 2

No. of time *left* event set changes	No. of events in *left* event set	No.of iterations	No. of successful cached computation
0	1	1	0
1	3	2	0
1	3	3	0
1	3	4	0
1	3	5	1
1	3	6	2
1	3	7	2
1	3	8	2
1	3	9	2
1	3	10	3
1	3	11	4
1	3	12	5
1	3	13	6
1	3	14	7
…			
3	11	82	167
3	11	83	168
3	11	84	169
3	11	85	170
4	1	1	170
5	11	1	170
5	11	2	171
5	11	3	172
5	11	4	173
5	11	5	174
5	11	6	175
…			
293	9	12	5742
293	9	13	5743
293	9	14	5744
294	1	1	5744
295	7	1	5744
295	7	2	5745
295	7	3	5745
295	7	4	5746
295	7	5	5746
295	7	6	5746
295	7	7	5747
295	7	8	5748
295	7	9	5749
295	7	10	5750
295	7	11	5751
295	7	12	5752
295	7	13	5753
295	7	14	5754

BIBLIOGRAPHY

[1] A.A. Basten. *Hierarchical Event-Based Behavioral Abstraction in Interactive Distributed Debugging: A Theoretical Approach*. Master's thesis, Eindhoven University of Technology, August 1993.

[2] T. Basten, T. Kunz, J. P. Black, M. H. Coffin, and D. J. Taylor. Vector time and causality among abstract events in distributed computations. *Distributed Computing*, vol. 11, pages 21-39, 1997.

[3] W. H. Cheung. *Process and Event Abstraction for Debugging Distributed Programs.* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, September 1989.

[4] Eclipse.org. The Eclispe Project. http://www.eclipse.org.

[5] Eclipse.org. The Hyades Project. http://www.eclipse.org/hyades.

[6] C.J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *Proceedings of the 11th Australian Computer Science Conference*, Brisbane, pages 56-66, February, 1988. Appeared as *ACM SIGPLAN Notices*, 24(1), January 1989.

[7] C.J. Fidge. Logical time in distributed computing systems. *IEEE Computer*, 24(8):28-33, August 1991.

[8] C.E. Jaekl. *Event-Predicate Detection in the Debugging of Distributed Applications*. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 1996.

[9] T. Kunz. *Abstract Behaviour of Distributed Executions with Applications to Visualization*. PhD thesis, Technical University of Darmstadt, Federal Republic of Germany, February 1994.

[10] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM* 21(7):558-565, July 1978.

[11] F. Mattern. On the relativistic structure of logical time in distributed systems. In M. Cosnard et al., editors, *Parallel and Distributed Algorithms*, pages 215-226. Elsevier Science Publishers B.V., Amsterdam, North-Holland, The Netherlands, 1989.

[12] J.A. Summers. *Precedence-Preserving Abstraction for Distributed Debugging*. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 1992.

[13] D.J. Taylor. Event displays for debugging and managing distributed systems. In *International Workshop on Network and Systems Management*, pages 112-124, Kyongju, Korea, August 1995.

[14] D.J. Taylor, T. Kunz and J.P. Black. Achieving target-system independence in event visualization. In *CD-ROM Proceedings of the CAS Conference*, 1995.

[15] Ping Xie. *Convex-Event Based Offline Event-Predicate Detection*. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2003.

[16] Ping Xie and D.J. Taylor. Specifying and locating hierarchical patterns in event data. In *CD-ROM Proceedings of the CAS Conference*, 2004.