SOFTWARE VERIFICATION AND SPATIOTEMPORAL AGGREGATION IN CONSTRAINT DATABASES

by

Scot Anderson

A DISSERATION

Presented to the Faculty of
The Graduate College at the University of Nebraska
In Partial Fulfillment of Requirements
For the Degree of Doctor of Philosophy
Major: Computer Science

Under the Supervision of Professor Peter Revesz

Lincoln, Nebraska

April, 2007

SOFTWARE VERIFICATION AND SPATIOTEMPORAL AGGREGATION IN CONSTRAINT DATABASES

Scot R. Anderson, Ph.D.
University of Nebraska, 2007

Advisor: Peter Z. Revesz

Computer controlled systems contribute to safety in transportation systems and many other critical life-saving and life-support systems. Part 1 presents the implementation and exploration of arbitrarily precise semantic approximations for software verification using over-approximation and under-approximation techniques in constraint databases. The approximations are implemented in the MLPQ system and several program examples are analyzed in comparison to previous results. Part 2 presents novel MaxCount, ThresholdRange, ThresholdCount, ThresholdSum, ThresholdAverage and CountRange estimation algorithms for moving points in d dimensions and a new spatiotemporal bucketing technique for indices. Each query runs in constant time and is based on skew-aware static size buckets. This bucket technique allows constant time inserts, deletes and updates needed for highly dynamic spatiotemporal databases. The technique is also decomposable to allow partial results to be calculated simultaneously and recombined in linear time. We performed extensive experiments that show these threshold aggregation estimation algorithms run up to 35 times faster than a precise algorithm with accuracy above 95%.

© Copyright by Scot Anderson 2007

All Rights Reserved

To my loving wife Patricia

Acknowledgements

I would like to thank my advisor Professor Peter Revesz, for his guidance and encouragement throughout my work at UNL and for the personally guided tour of Budapest which I will never forget. I would also like to thank Professor Berthe Choueiry for her kind interest in me and hard work that she puts into every student and class. A special thanks goes to Professor Steve Cohn from the Mathematics department for his encouragement in pursuing a computer science degree and for his kind and considerate introduction to my first graduate class at UNL. Also a special thanks to Professor Ashok Samal for serving on my Ph.D. committee.

I would especially like to thank Randy Bell for giving me a copy of Mathematica, which helped immensely in many of the complex calculations required for both the theory and programming.

I would like to thank my parents, Robert and Charel Anderson, for their love, support, and opportunities to live life to the fullest. Without their training and guidance, I would never have made it this far in life. I would also like to thank my brother Eric who always set an example that was worth following. Without his constant companionship growing up, my life would truly have been a misadventure. Finally I would like to thank my wife Patricia Anderson for her love, support and encouragement.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Programs increasingly control many aspects of our daily lives such as air traffic control (ATC) systems. Failures such as the computer controlled Airbus A320 crash on June 26, 1988 (Kilroy 1997) show that programs need thorough debugging and verification before risking lives. We propose a new constraint-database approach to program debugging and verification in Part 1 of this dissertation.

Programs also increasingly need to efficiently deal with moving objects, which are also called spatiotemporal objects, such as airplanes in the case of air traffic control systems. To increase program efficiency, we propose new spatiotemporal aggregation operators in Part 2 of this dissertation.

We outline the main issues about software verification in Section 1.1 and about spatiotemporal aggregation in Section 1.2.

1.1. Software Verification

Verifying the correctness of software is undecidable in general. That is easy to see by looking at the well-known Halting Problem, which is the problem of deciding whether a given program with a given input will terminate. Since the Halting Problem is undecidable in general and termination of programs is usually considered one of the conditions of correctness, it is clear that software verification is also undecidable in general.

However, let us take a deeper look at software verification and identify what can be done. We start with the following definitions.

Definition 1.1 (Program State)**.** *A program state is a pair consisting of the values assigned to the program variables and the specific location of the program code where such an assignment occurs during an execution of the program.*

We call the meaning of the program the *semantics* of the program. In this dissertation we are concerned with the following program semantics, called the collecting semantics[1].

Definition 1.2 (Collecting Semantics)**.** *The collecting semantics is the set of all possible program states that a program may enter during some execution and some input.*

A key idea in software verification is that the program semantics can be approximated using a terminating program that takes as input the program and some approximation parameters and gives either an under-approximation or an over-approximation, which we define as follows.

Definition 1.3 (Over-Approximation)**.** *Let S be the semantics of a program. We say that any P^l where $S \subseteq P^l$ is an* over-approximation*.*

Definition 1.4 (Under-Approximation)**.** *Let S be the semantics of a program. We say that any P_l where $P_l \subseteq S$ is an* under-approximation*.*

[1]see P. Cousot lecture notes: `http://www.cs.wisc.edu/~cs704-1/LectureNotes/9.AbstractInterpretation.txt`

The approximation is often useful to check certain concerns about the program. These concerns are expressed as some conditions called *error states* that need to be avoided by the program to be considered correct.

If the over-approximation does not contain the error states, then the program is considered correct. However, error states contained in an over-approximation may be spurious. Hence they do not prove the program incorrect.

The spurious error states may be avoided by tightening the over-approximation. If repeated tightening fails to eliminate the error states, then we may suspect that the program is incorrect. By using an under-approximation, we can often prove that the program is incorrect, i.e., falsify it. If an *under-approximation* of the semantics contains some error state, then the program is incorrect. Falsification identifies some of the errors in the program, hence it is a useful aid in debugging the program.

Chapter 2 reviews software verification literature with respect to Abstract Interpretation. Chapter 3 presents the constraint database approach to software verification. The chapter includes the improvements to MLPQ and the framework to translate programs into Datalog rules that represent program semantics. Chapter 4 discusses the experiments, results, conclusions, and future work.

1.2. Spatiotemporal Aggregation

Max, Min, Count, Sum and Average form the set of natural aggregation-operators for relational databases. Spatiotemporal databases containing moving objects can not apply these operators in the same way. However, these operators may still function in interesting ways for moving objects. For example, one can

ask how many moving point objects exist within a rectangular area *at a certain time*, or what is the maximum distance between two moving points *at certain times*. Obviously, when we are interested in a certain time, then the moving point object database can be reduced to a relational database and the above queries can be expressed as simple COUNT or MAX queries.

Moving object databases naturally suggest new aggregate operators that have no equivalents in relational databases. For example, one may ask what is the maximum number of moving-point objects that exist simultaneously within a moving rectangular area at any time instance during a time interval T? We call this the MAXCOUNT query (symmetrically we can also find the MIN-COUNT). One may also ask during what time intervals in T does there exist more than M moving objects within a rectangular area? We call this the THRESHOLDRANGE. We show that a strong relationship exists between MAXCOUNT and THRESHOLDRANGE, and we show that THRESHOLDRANGE forms the bases for a family of threshold operators that include: THRESHOLDCOUNT, THRESHOLDSUM, and THRESHOLDAVERAGE. A related, though less complex, operator answers the question: what is the number of moving objects that exist within or intersect a rectangular area at any time instance during interval T. We call this the COUNTRANGE query.

Threshold aggregation operators are important in many applications such as tracking airplanes or mobile clients of wireless networks.

Example 1.1. Airplanes are commonly modeled as linearly moving objects with preestablished flight plans. Suppose, at any time, at most a constant number M of airplanes is allowed to be in the O'Hare airspace to avoid congestion. Suppose also

a new airplane requests approval of its flight plan for entering the O'Hare airspace between times t_a and t_b. The air traffic controllers can avoid congestion as follows. If after adding a new flight plan, the MaxCount between t_a and t_b is still less than M, then they can approve the flight. Otherwise, they need to find some alternative path, and check it again against the database.

Air traffic controllers try to direct airplanes as linearly moving objects for fuel efficiency, among other reasons. If they recognize a developing congestion too late, then they often must direct the airplane to fly in circles until the congestion has cleared. That solution wastes fuel. On the other hand, if they recognize the developing congestion early, then they can often simply tell the airplane to change its speed, which saves fuel. Therefore, it is important to identify congestions as early as possible. We may identify congestions by using a MaxCount query where a moving box around the airplane and a time interval $[t_a, t_b]$ define the query. If the MaxCount predicts congestion, then the airplane's speed can be adjusted early in the flight.

Example 1.2. Suppose we want to alert pilots if their current flight path takes them through at least one congested region.

Traffic Alert/Collision Avoidance Systems (TCAS) is a system that provides similar functionality. TCASs only provide alerts for current congestion, not predictive congestion. Although TCASs were implemented in 1986, we continue to have mid-air collisions and near misses indicating that the system still needs improvement. ThresholdRange is a modification of MaxCount that returns all

predicted time intervals on the flight path where the COUNT exceeds a given threshold. Hence using THRESHOLDRANGE we can alert a pilot of predicted congestions where more than M other airplanes will be within the space B around the airplane. Predicting and avoiding these areas can significantly reduce the chances of mid-air collisions.

Example 1.3. Suppose we are especially concerned about a rush-hour period $[t_a, t_b]$ that is particularly stressful to air traffic controllers. Suppose controllers can direct at most M airplanes safely. We can determine the number of controllers needed during the rush-hour time by executing the COUNTRANGE query over the controlled airspace during the rush-hour and dividing by M. By ensuring that a sufficient number of controllers are present, safety is achieved and controllers are not over stressed.

These questions and examples, motivated by research on MAXCOUNT, led us to explore complex threshold aggregations and indexing techniques to support them.

1.3. Overview of Contributions

The detailed contributions of this dissertation can be described chapter-by-chapter as follows.

Chapter 2 describes some basic concepts and previous work related to software verification. Chapter 3 describes the general constraint database approach to software verification and the details of constraint database approximation. Further, we extend the MLPQ constraint database (Revesz et al. 2000) system to include both over-approximation and under-approximation techniques. Chapter 4 gives three

examples of software verification and falsification. We use these examples to compare the constraint database approach to other techniques and conclude that our method provides higher levels of precision than the other methods. This work is based on Anderson & Revesz (2005) and Anderson & Revesz (2007*a*).

Chapter 5 describes some basic concepts and previous work related to spatiotemporal aggregation. Chapter 6 describes the theory of our new bucket structure and how it allows a wide range of indices to perform dynamic updates. Chapter 7 gives the theoretical basis for our spatiotemporal aggregation technique and develops the d-dimensional MaxCount algorithm. We analyze the MaxCount algorithm and show that it runs in constant time, and present motivating examples in 3-dimensional space. Chapter 8 extends the theory for MaxCount to include ThresholdRange, ThresholdCount, ThresholdSum, and ThresholdAverage. We analyze these operators in relation to the MaxCount operator, and show that each runs in constant time. Chapter 9 describes a new estimation algorithm for CountRange, and shows that it runs in constant time. Chapter 10 gives our methods for measuring precision and accuracy of the different aggregation operators. We present some new methods for generating data used in the experiments, and show how it relates to other data generators for spatiotemporal data. We analyze the accuracy and precision of our estimation methods related to the exact methods, and give an empirical result that describes when each method should be used. This work is based on (Anderson 2006) and Anderson & Revesz (2007*b*).

Appendix A gives integral evaluations for each case given in Chapter 7. Appendix B discusses the implementation used in the experiments. Appendix C gives additional results not shown as part of Chapter 10.

Part 1

Software Verification in Constraint Databases

CHAPTER 2

Literature Review

Abstract interpretation techniques provide the framework for extracting abstract semantics of a program (Cousot & Cousot 1977, Cousot & Halbwachs 1978, Cousot & Cousot 1992*a*, Halbwachs 1993, Kerbrat 1995, Cousot 2005).

Definition 2.1 (Abstract Semantics)**.** *We define the abstract semantics of a program by approximating the set of possible program states at a certain location in a program with a finite set of invariants (usually a certain type of constraint).*

The abstract semantics over-approximate the values of the program variables in a convex state space. Hence, the abstract semantics model all possible program states and make verification of program correctness possible. The abstract interpretation framework gathers information about programs in order to build the abstract semantics. These abstract semantics provide sound answers to questions about the program run-time behaviors. Given the type of invariants, these semantics can then be used to design automatic program analyzers (Cousot & Cousot 1992*a*). Abstract interpretation is often understood in terms of abstract-evaluation of the program. Hence, we describe abstract interpretation in terms of the following:

(1) Program P.

(2) Abstract evaluation of P. And,

(3) Abstract semantics.

We elaborate on items 2 and 3 below and in Figure 2.1.

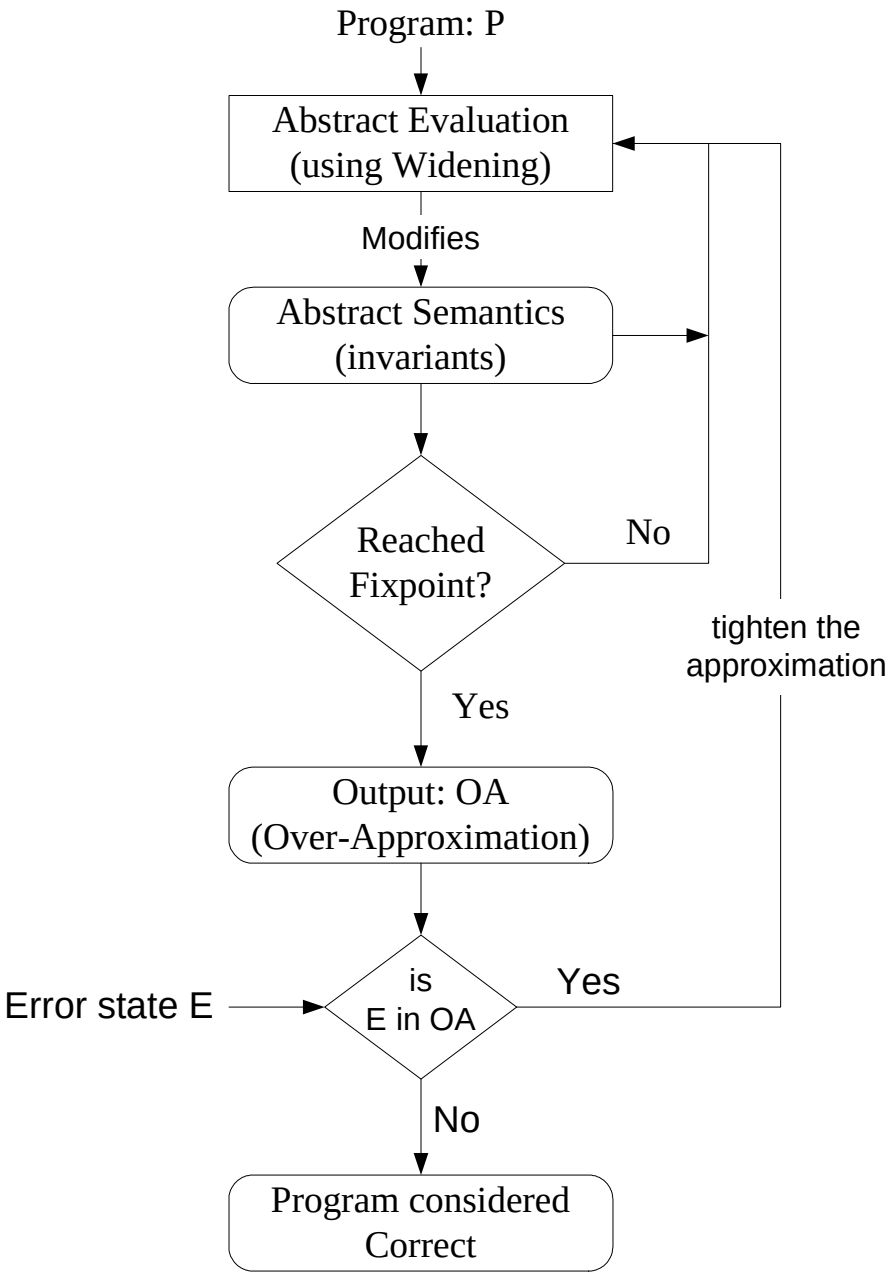


Figure 2.1. Abstract interpretation process.

Definition 2.2 (Abstraction operator)**.** *Given a program P written in a language L and abstract domain $D^\sharp$, define the abstraction operator as:*

$$\alpha : L \to D^\sharp \tag{2.1}$$

Then the abstract semantics of P in $D^\sharp$ represents the over-approximation.

In Step (2) of Figure 2.1 the process finds the abstract semantics by repeatedly performing an abstract evaluation that modifies invariants of the program P. When the process discovers only states already contained in the invariants, then the process has reached a fixpoint and terminates. Figure 2.1 shows the process of reaching an over-approximation.

Definition 2.3 (Counter Automata)**.** *Counter automata $\mathcal{A}$ are defined by tuples (S, X, τ, s_0, A) where S is a finite set of states, X is a finite set of counters $x_1, \ldots, x_k$, which are integer variables, τ is a finite set of transitions from S to S, s_0 is an initial state, and $A \equiv a_1, \ldots, a_k$ is an initial assignment of the state variables. Each transition has two parts, a* guard constraint *over the variables that needs to be satisfied before the transition takes place and a set of assignments to the variables that update their values as the automaton enters the new state.*

An example of counter automata is counter machines which allow only guard constraints that compare variables and constants, and perform assignments that increment and decrement a variable by one or set a variable to a constant. Floyd & Beigel (1994) give an introduction to automata theory that covers counter machines.

Boigelot & Wolper (1994), Fribourg & Olsén (1997), Fribourg & Richardson (1996), Halbwachs (1993), and Kerbrat (1995) study extensions of counter machines.

The counter automata restricts the kind of operations needed to perform abstract evaluation and build the abstract semantics. The transitions in the automata (i.e. the arrows and constraints on the arrows) can represent different things depending on the language of L used in the constraints.

As an example of counter automata, consider the subway control program shown in Figure 2.2.

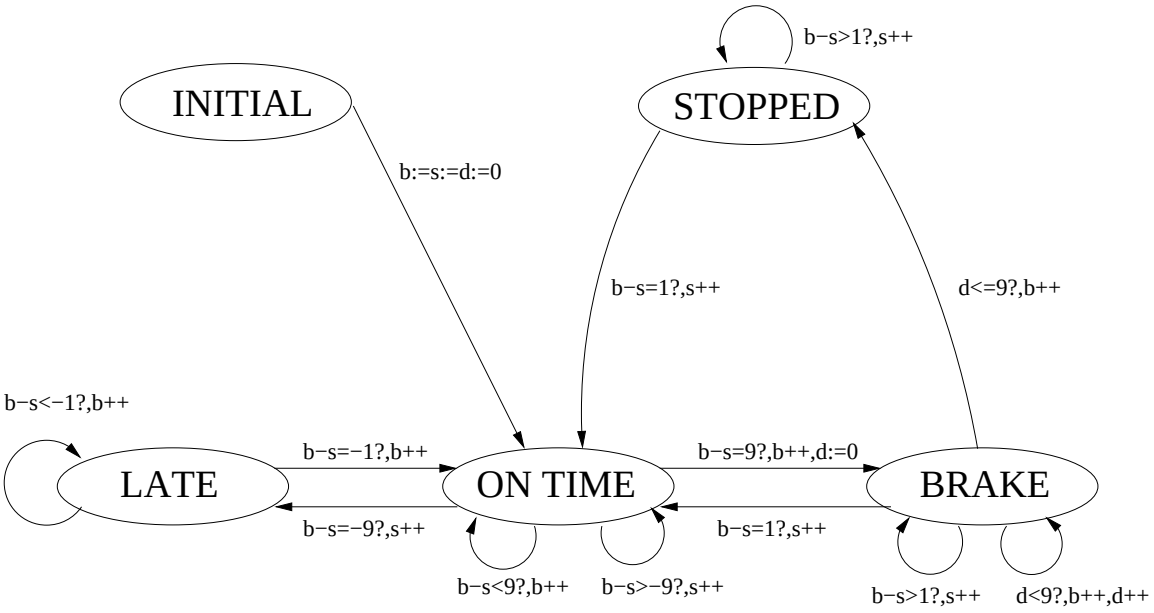


Figure 2.2. The subway automaton.

Example 2.1 (Subway Counter Automata)**.** Figure 2.2 shows an example counter automaton A that represents the control program of a subway system. Although A

has a finite set of states as shown, the values of b, d, and s increase with each transition.Therefore, there exists an infinite number of states. We analyze A in detail in Chapter 4.

The abstraction operator α transforms A into a set of invariants in the abstract domain that hold between the different variables. We can express the invariants of this automaton using difference constraints. Other common abstractions represent invariants using intervals, linear inequalities, or convex polyhedra (Cousot & Halbwachs 1978).

Example 2.2 (Interval Domain)**.** Suppose that an interval abstract domain represents the states of the program variables and that at point after repeatedly applying the abstract evaluation to P we have

$$0 \leq x_1 \leq 10 \quad \wedge \quad 0 \leq x_2 \leq 10 \tag{2.2}$$

Further, suppose in the next application of abstract evaluation of P, we discover the state $x_1 = 5, x_2 = 20$ exists. This addition *widens* the abstract semantics to

$$0 \leq x_1 \leq 10 \quad \wedge \quad 0 \leq x_2 \leq 20 \tag{2.3}$$

This widening doubles the representation size for a single state as shown in Figure 2.3 (a) and (b).

This simple example demonstrates the problems of any type of convex abstract domain. The example above uses the interval abstract domain and expresses the idea of *widening*. The widening operator invoked by abstract evaluation takes the

states found during the abstract evaluation and expands the abstract semantics to include any new states discovered.

Hence, when abstract evaluation reaches the fixpoint, the widening operator can no longer expand the representation of the state. While the interval abstract domain from Example 2.2 allows fast calculations, the results are usually too imprecise for most applications. Clearly different abstract domains and widening operators must be used.

Definition 2.4 (Widening Operator (Cousot & Cousot 1976, 1992*b*))**.** *Let L be a lattice and $\sqsubseteq$ be the ordering operator. The expression $a \sqsubseteq b$ reads "a precedes b" where $a, b \in s$ and s is an increasing sequence in L. A widening operator is defined as a mapping $\nabla : L \times L \to L$ such that*

$$\begin{aligned} \forall x, y \in L : x \sqsubseteq x \nabla y \\ \forall x, y \in L : y \sqsubseteq x \nabla y \end{aligned} \tag{2.4}$$

and for all increasing chains in L, $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$, the increasing chain defined by $y^0 = x^0, \ldots, y^{i+1} = y^i \nabla x^{i+1}, \ldots$ is not an infinite, strictly increasing chain.

Widening operators of many varieties (Cousot & Cousot 1976, 1992*b*, Bagnara et al. 2005, Revesz 2007) provide the mechanism for expanding abstract semantics to find an over-approximation.

Widening operators serve three main purposes (Cousot & Cousot 1992*a*):

(1) Force the stabilization of approximated iteration sequences after a finite number of iterations.

(2) Speed the convergence of iteration sequences. And,

(3) Select a (possibly infinite) set of approximations of concrete elements when considering abstract domains that are algebraically weak.

The convex-polyhedra domain with various associated widening techniques introduced by Cousot & Halbwachs (1978) and expanded by Halbwachs (1979) is the most widely used abstract domain (Bagnara et al. 2005).

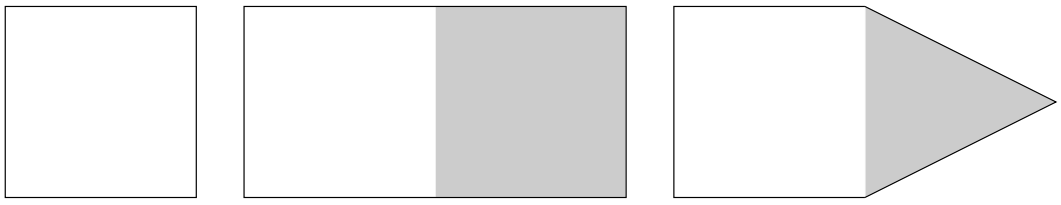


Figure 2.3. (s) Original state space. (b) Widened using intervals. (c) Convex polyhedra.

Example 2.3 (Convex Polyhedra Domain)**.** Suppose instead of using intervals as in Example 2.2, a convex polyhedra represented by linear constraints is used to bound the state space. The representation for the state space in the abstract domain of complex polyhedra in Figure 2.3 (a) gives:

$$x_1 \geq 0 \quad \wedge \quad -x_1 \geq -10 \quad \wedge \quad x_2 \geq 0 \quad \wedge \quad -x_2 \geq -10 \tag{2.5}$$

However, widening increases the state space by half as much as in Example 2.2 as follows.

$$\begin{aligned} &x_1 \geq 0 \quad \wedge \quad -x_1 \geq -10 \quad \wedge \quad x_2 \geq 0 \quad \wedge \\ &2x_1 - x_2 \geq -10 \quad \wedge \quad -2x_1 - x_2 \geq -30 \end{aligned} \tag{2.6}$$

Figure 2.3 shows the original state space in (a) and the results of widening using intervals in (b) versus widening using convex polyhedra in (c). In each case the gray areas show the widened section.

The proliferation of tools for software verification gives a hint to the broad spectrum of approaches used to solve this important problem. The YAHODA verification tools database (Crhová et al. 2002) lists 57 tools. The most relevant approach to our constraint database approach is Abstract Interpretation. However, constraint database approximation is different from abstract interpretation methods. The main difference is that, at least in theory (Revesz 1999), both an under-approximation and over-approximation of the least fixpoint can be arbitrarily close to the actual least fixpoint. Second, constraint databases use a natural under-approximation and over-approximation that does not use a widening operator. Third, our method allows disjoint, non-convex regions to represent invariants. Naturally, increasing precision in our method increases the running time. More details about constraint logic programming and constraint databases can be found in the surveys by Jaffar & Maher (1994), and Revesz (1998), and the books by Kuper et al. (2000), Marriott & Stuckey (1998), and Revesz (2002).

CHAPTER 3

The Constraint Database Approach

In the constraint database area, researchers have found methods for finding over-approximations and under-approximations of the least fixpoint semantics of Datalog programs. MLPQ Revesz et al. (2000) is a constraint database system that provides a high degree of precision by allowing non-convex and disjoint regions to represent collecting semantics. The level of approximation is controlled by a single parameter l.

Figure 3.1 provides the general constraint database approach to the verification of programs (Revesz 2007).

Section 3.3 describes the first two steps in detail. These steps form the framework that translates a program into Datalog. The next step calculates an over-approximation given the bounding parameter l. The results from the over-approximation (or under-approximation) often contain a large set of data due to the disjunctive representation of variable values. The constraint database approach simplifies interpretation of results by providing native facilities to query the results for error states using Datalog or SQL. Not finding the error state in the over-approximation verifies program correctness. If we suspect that the error state is present, we perform the under-approximation. Finding the error state in the under-approximation falsifies the program.

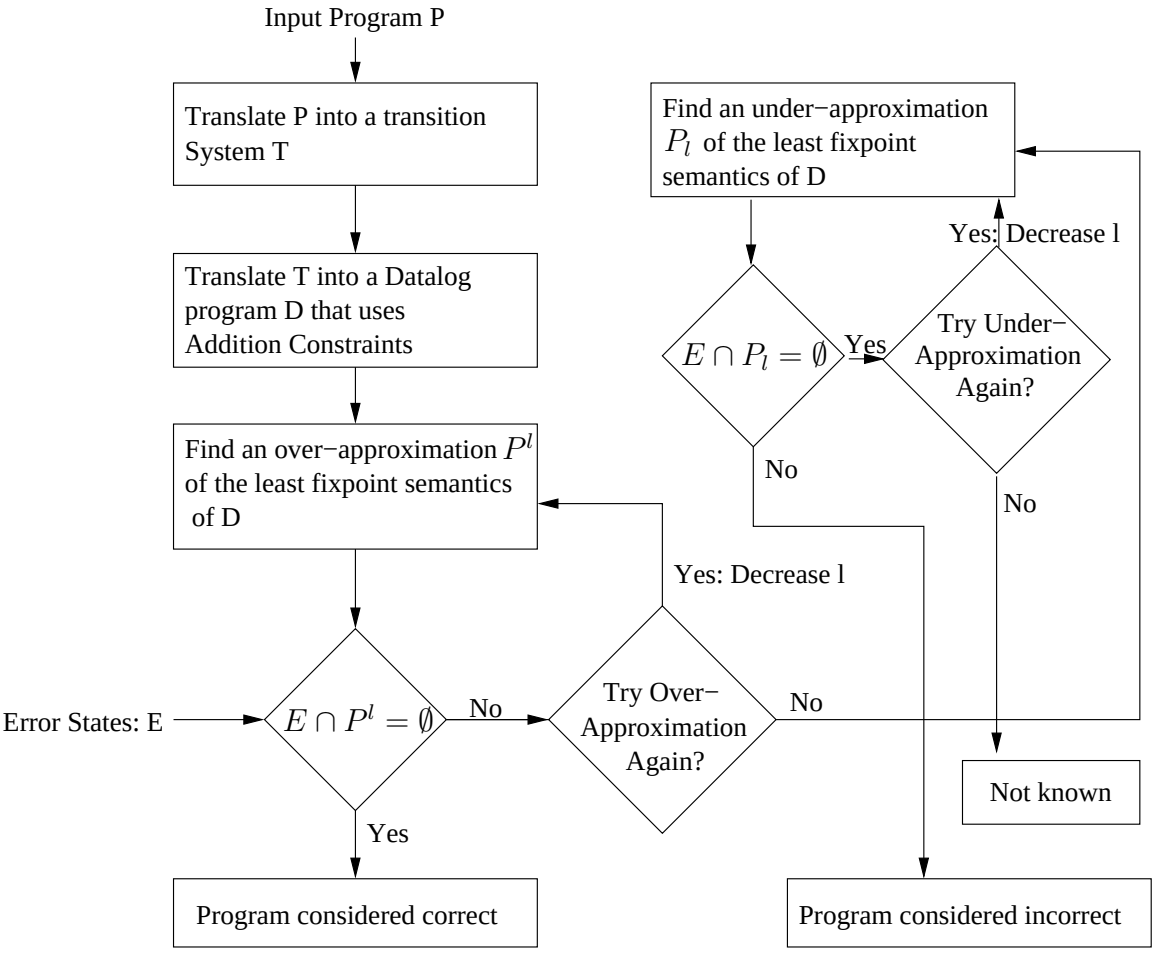


Figure 3.1. Constraint database approach.

In theory, the constraint database approach can reach arbitrary precision by calculating the under-approximation and over-approximation repeatedly as $l \to -\infty$. Hence, this method approaches a precise evaluation. While this approach may not be possible for every constraint, it may be reasonable to lower l to a point where the constraint used to query for an error state converges. This method provides a way to find constraints in the program that converge in conjunction with constraints that do not. Chapter 4 gives an example of convergence in the subway automaton.

3.1. Approximation Background

It is easy to express in Datalog with addition constraints a program that will not terminate using a standard bottom-up evaluation (Revesz 2002). Consider the following Datalog program:

$$\begin{aligned} D(x, y, z) \quad &:\!- \quad x - y = 0,\ z = 0. \\ D(x', y, z') \quad &:\!- \quad D(x, y, z),\ x' - x = 1,\ z' - z = 1. \end{aligned} \tag{3.1}$$

This query expresses that the DIFFERENCE of x and y is z. Further, based on Equation (3.1) we can also express a MULTIPLICATION relation as follows:

$$\begin{aligned} M(x, y, z) \quad &:\!- \quad x = 0,\ y = 0,\ z = 0. \\ M(x', y, z') \quad &:\!- \quad M(x, y, z),\ D(z', z, y),\ x' - x = 1. \\ M(x, y', z') \quad &:\!- \quad M(x, y, z),\ D(z', z, x),\ y' - y = 1. \end{aligned} \tag{3.2}$$

Using Equations (3.1) and (3.2) we can express any Diophantine equation (Anderson 2003), which by Matiyasevich (1993), is Turing complete.

Definition 3.1 (Addition Constraints)**.** *Addition constraints (Revesz 2002) have the form:*

$$\pm\, x \pm y\ \theta\ b \ \textit{ or } \ \pm\, x\ \theta\ b \tag{3.3}$$

where x and y are integer variables, and b is an integer constant called a bound*, and θ is either $\geq$ or $>$.*

Addition constraints can be represented using Addition Bound Matrices (ABM) defined as follows.

Definition 3.2 (Addition Bound Matrices (ABM) (Miné 2001))**.** *Given a conjunction of addition constraints with variables $x_1, \ldots, x_n$, represent each variable in two cases:*

$$+ x_i \rightarrow x_i^+ \tag{3.4}$$

$$-x_i \rightarrow x_i^- \tag{3.5}$$

Hence we will represent the variables using $2n$ symbols in an Addition Bound Matrix as:

	x_1^+	x_1^-	$\ldots$	x_n^+	x_n^-
x_1^+	$e_{1,1}$	$e_{1,2}$	$\ldots$		$e_{1,2n}$
x_1^-	$e_{2,1}$				
$\ldots$	$\vdots$		$\ddots$		$\vdots$
x_n^+					
x_n^-	$e_{2n,1}$		$\ldots$		$e_{2n,2n}$

Each entry $e_{i,j}$ represents the bound in the constraint $X - Y \geq B$ where X is the row variable and Y is the column variable. If there is no constraint between a row and column variable the bound is given as $-\infty$.

Example 3.1 (ABM (Revesz 2007))**.** Consider the following upper bound, lower bound and addition constraints:

$$-x \geq -25, \quad y \geq 3, \quad x - y \geq 4, \quad x + y \geq 10, \quad -x - y \geq -40. \tag{3.6}$$

Translating these constraints into difference constraints we have:

$$\begin{aligned} x^- - x^+ &\geq -50 \\ y^+ - y^- &\geq 6 \\ x^+ - y^+ &\geq 4 \\ x^+ - y^- &\geq 10 \\ x^- - y^+ &\geq -40 \end{aligned} \tag{3.7}$$

For constraint databases, Revesz Revesz (1999) introduced two methods for approximating the least fixpoint evaluation of addition constraints by modifying the standard bottom-up evaluation.

Definition 3.3 (Lower-Bound Modification)**.** *Let $l < 0$ be any fixed integer constant. We change in the constraint tuples the value of any bound b to be $max(b, l)$. Given a Datalog program P, the result of a bottom-up evaluation of P using this modification is denoted P_l.*

Definition 3.4 (Upper-Bound Modification)**.** *Let $l < 0$ be any fixed integer constant. We delete from each constraint tuple any constraint with a bound that is less than l. Given a Datalog program P, the result of a bottom-up evaluation of P using this modification is denoted P^l.*

These modifications lead to the following approximation theorem:

Theorem 3.1 (Revesz 1999)**.** *For any Datalog program P and constant $l < 0$ the following is true:*

$$P_l \subseteq lfp(P) \subseteq P^l \tag{3.8}$$

where lfp(P) is the least fixed point of P. Further, P_l and P^l can be computed in finite time.

We implemented these two constraint modifications in MLPQ to allow the over-approximation and under-approximation of the semantics of Datalog with addition constraints for software verification.

3.2. Improvements to MLPQ

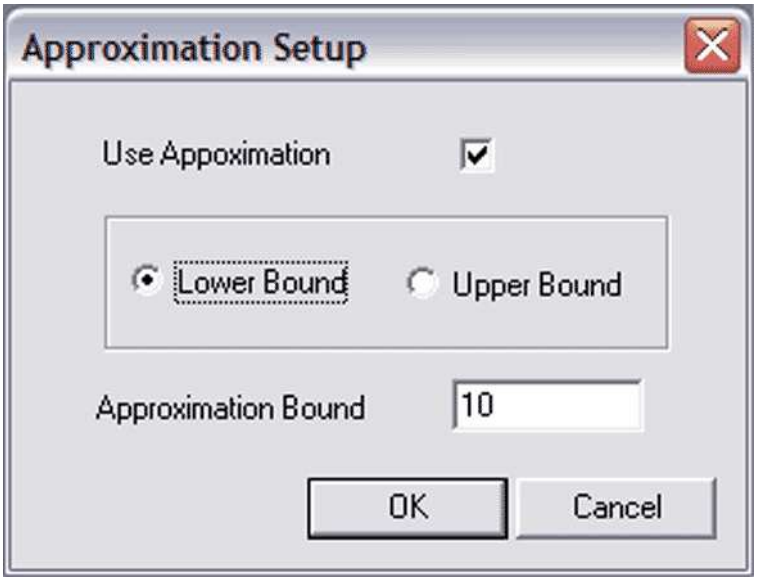


Figure 3.2. Approximation dialog box.

MLPQ is a research implementation of a Constraint Database system developed using Microsoft Visual C++. Over several years many different graduate students at the University of Nebraska-Lincoln have contributed to this project. The on-going effort to add new features and fix bugs has made this program a viable research tool.

We implemented the lower-bound and upper-bound approximation techniques given in Definitions 3.3 and 3.4. In addition we made several memory handling modifications to speed up the query evaluation and conserve system resources.

The program does not apply an approximation modification by default, but the user may turn the modification on by clicking the "Apx" button shown in Figure 3.3. Pressing the "Apx" button pops up a dialog box (shown in Figure 3.2) which allows the user to turn on the approximation, set the "Approximation Bound" l and select the type of evaluation to use. MLPQ implements constraints using predicates $=, <$

Figure 3.3. Approximation added to toolbar.

and $\leq$ whereas the theory (Revesz 2002) uses $>$ or $\geq$. This change in predicate has led to a completely symmetric, but consistent implementation. As such the value for l must be positive and increased for increased accuracy, whereas, in the theory, l must be decreased to decrease error. In the experiments we express l as positive to be consistent with the implementation.

3.3. Framework for Translating Programs into Datalog

The framework we give adapts the standard pre-condition transition system used in static analysis and compiler optimization techniques.

Definition 3.5 (Transition System)**.** *A transition system is a tuple* $(S, \wedge, \rightarrow)$ *where* S *is a set of states,* $\wedge$ *is a set of labels and* $\rightarrow \subseteq S \times \wedge \times S$ *is a ternary relation of labeled transitions. If* $p, q \in S$ *and* $\beta \in \wedge$*, then* $(p, \beta, q) \in \rightarrow$ *is written as:*

$$p \xrightarrow{\beta} q \tag{3.9}$$

where β *is a set of conditions and operations to the source state variables that must be made to enter the target state.*

The abstract domain we use consists of addition and difference constraints. The framework translates a program into Datalog in two steps:

(1) Create the transition system T.

(2) Translate that system into a Datalog query that uses only addition and difference constraints.

Step (1) and (2) perform the step of translating a program P to a transition system and then into Datalog with addition constraints. The two translations correspond to the first optional step in Figure 2.1, which translates P into P'.

Given a program P with n lines of code and m variables, step (1) gives the transition system where a program statement (or state) $p_i \in S$ denotes the program statement on line i about to be executed. A transition from some state p_j to p_i denoted $p_j \xrightarrow{\beta} p_i$ represents the rule to enter state p_i where β contains the "execution" of the program statement on line j. The values changed by β affect the values available for execution in p_i. *How the new values affect the* p_i *state will determine the type of approximation.* In Datalog these values will be added to the

set of existing values. In abstract interpretation the new values widen the existing invariants.

Example 3.2 (Transition System Framework)**.** Suppose we have the program:

L1: $a = 1$; $b = 2$; $c = 1$
L2: $d = b^2 - 4ac$
L3: return d

If we assume no initialization then we must allow any values on entry to L1. Figure 3.4 shows this transition system where $U(v)$ means the variable v is unconstrained.

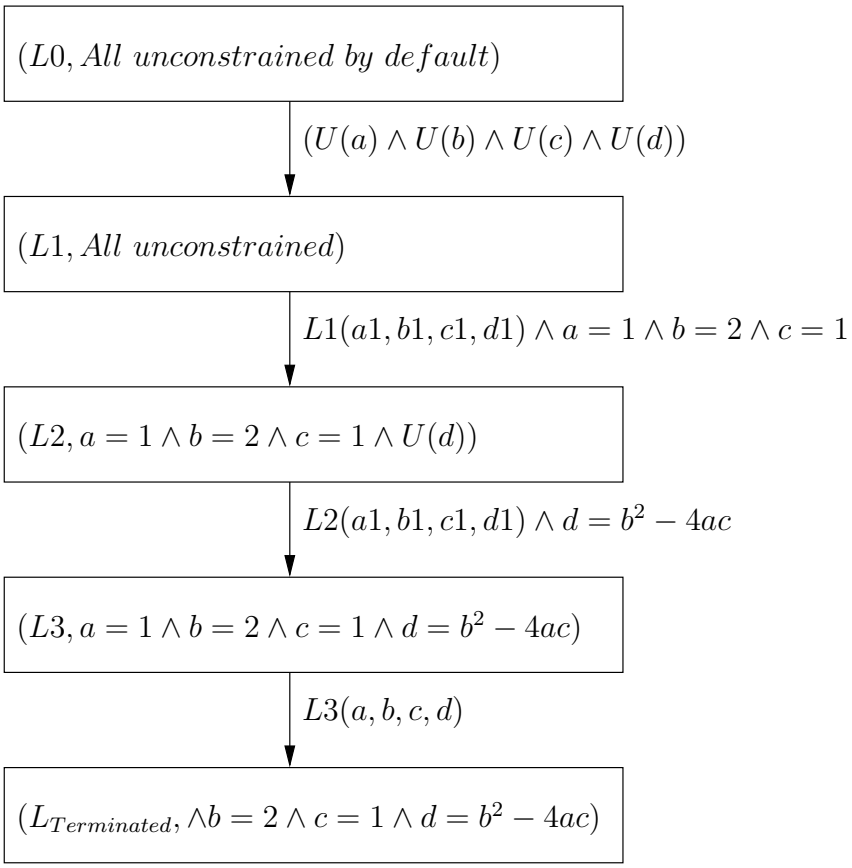


Figure 3.4. Example 1 transition system.

Translating the output of step (1) consists of creating a Datalog rule from each $\xrightarrow{\beta}$ rule on an edge from any state to p_i. We use the difference relation D from Equation (3.1), the multiplication relation M from Equation (3.2) and the Unconstrained relation defined by:

$$\begin{aligned} D(x) :- \quad & x \geq 0. \\ D(x) :- \quad & x < 0. \end{aligned} \tag{3.10}$$

Example 3.3 (Translation to Datalog)**.** Continuing Example 3.2, gives the following Datalog program:

```
begin%RECURSIVE%
L0(a,b,c,d) :- U(a),U(b),U(c),U(d).
L1(a,b,c,d) :- L0(a,b,c,d).
L2(a,b,c,d) :- L1(a1,b1,c1,d), a=1, b=2, c=1.
L3(a,b,c,d) :- L2(a,b,c,d1), M(b,b,b2), M(a,c,e1), M(e1,4,e2), D(b2,e2,d).
LT(a,b,c,d) :- L3(a,b,c,d).
end%RECURSIVE%
```

Clearly this Datalog program can be optimized by removing L0 and substituting the body of L0 into the body of L1. Also LT is the same as L3 and thus we can delete it as well. However you may not always be able to do this in general since there may be calculations in a return statement. We use this framework to translate the examples in Chapter 4.

CHAPTER 4

Verification Experiments and Results

Sections 4.1–4.3 demonstrate the techniques presented with several programming examples starting with a simple recursive goto program and ending with the subway counter automaton. Section 4.4 gives conclusions and future work.

4.1. Recursive goto Program

The simple program below demonstrates how a previous abstract interpretation approach gives an invariant that includes an error state that the constraint-database invariant correctly excludes.

1. a ← 0 2. a ← a + 1 3. **if** a > 2 **then goto** 6 4. **if** a = 2 **then goto** 7 5. **goto** 2 6. ... 7. ...	→	begin%RECURSIVE% L2(a) :- a=0. L2(a) :- L5(a). L3(a) :- L2(a1), a-a1=1. L4(a) :- L3(a), a≤2. L5(a) :- L4(a), a<2. L5(a) :- L4(a), a>2. L6(a) :- L3(a), a>2. L7(a) :- L4(a), a=2. end%RECURSIVE%

Miné (2001) defines an abstract interpretation widening technique as follows:

Definition 4.1 (Widening Operator of Miné (2001))**.** *Let M and N be two ABMs. Then the* widening *of M by N, written as $M \nabla N$ is defined as:*

$$[M \vee N][i,j] = \left\{ \begin{array}{lll} M[i,j] & \text{if} & M[i,j] \leq N[i,j] \\ -\infty & \text{if} & N[i,j] \leq M[i,j] \end{array} \right\} \tag{4.1}$$

Table 4.1 shows invariants found by two passes using the Miné widening technique given above. In the second entry, an invariant of $a \geq 1$ entering line (3) indicates that line (6) is executed.

Table 4.1. Invariants obtained by Miné widening.

Line	1st Entry	2nd Entry
2	$0 \leq a \leq 0$	$0 \leq a$
3	$1 \leq a \leq 1$	$1 \leq a$ if condition $a > 2$ true goto 6
4	$1 \leq a \leq 1$	
5	$1 \leq a \leq 1$	

When evaluating the Datalog program semantics using MLPQ, the invariant for line 3 never indicates that $a > 2$, and hence, we find that the `L6(a)` relation is empty. If line (6) identifies an error, then our program technique identifies the error state correctly where this abstract interpretation method does not.

4.2. Ship resupply problem

Suppose a yacht is traveling through the ocean between two locations. The yacht does not have enough supplies to make the trip, hence it must resupply at several possible locations. The program below determines if a point $(22, 19)$ can be

reached from a starting position of $(0, 0)$. It includes the `Depot` relation containing possible resupply locations. The `Leg` and `Reach` rules calculate Euclidian distance using the `D` (difference) and `M` (multiplication) relations. The approximation value l limits the evaluation of `D` and `M`, which may be pre-computed to save time. The `reach` relation determines if the destination can be reached. This example can be extended by adding a relation for multiple destinations. In that case, knowing error destinations would allow us to query the `reach` relation for incorrect values.

The results of running this program in MLPQ verify that the `reach` relation contains the values $x = 22$ and $y = 19$. The existence of these values verifies the Resupply Depot program as correct.

```
begin%SupplyDepotOptimized%
Depot(id,x,y)      :- id=1,x=0,y=19.
Depot(id,x,y)      :- id=2,x=6,y=8.
Depot(id,x,y)      :- id=3,x=15,y=12.
Depot(id,x,y)      :- id=4,x=25,y=5.
D(x,y,z)           :- x-y=0,z=0.
D(x,y,z)           :- D(x1,y,z1),x-x1=1,z-z1=1.
D(x,y,z)           :- D(x1,y,z1),x-x1=-1,z0z1=-1
M(x,y,z)           :- x=0,y=0,z=0.
M(x,y,z)           :- M(x1,y,z1),D(z,z1,y),x-x1≥1,x1-x≥-1.
M(x,y,z)           :- M(x,y1,z1),D(z,z1,x),y-y1≥1,y1-y≥-1.
Leg(x,y)           :- x=0,y=0.
Leg(x,y)           :- Leg(x1,y1),Depot(id,x,y), AD(x,x1,dx),AD(y,y1,dy),
                      M(dx,dx,dx2),M(dy,dy,dy2), dx2+dy2≤100,dx≤10,dy≤10.
Reach(x,y)         :- x=22,y=19,Leg(x1,y1), AD(x,x1,dx),AD(y,y1,dy),
                      M(dx,dx,dx2),M(dy,dy,dy2),dx2+dy2≤100,dx≤10,dy≤10.
end%SupplyDepotOptimized%
```

4.3. Subway Counter Automaton

Consider the subway train speed regulation system in Figure 2.2 (from Chapter 2) described by Halbwachs (1993). Each train detects "beacons" that are marks placed along the track and receives a "second" signal from a central clock.

Let b and s be counter variables for the number of beacons and second signals received. Further, let d be a counter variable that describes how long the train is decelerating by applying its brake. The goal of the speed regulation system is to keep $\mid b - s \mid \leq 20$ while the train is running.

The speed of the train is adjusted as follows. When $s + 10 \leq b$, then the train notices it is early, and applies the brake as long as $b > s$. Continuously braking causes the train to stop before encountering 10 beacons.

When $b + 10 \leq s$, the train is late and will be considered late as long as $b < s$. As long as any train is late, the central clock will not emit the second signal.

The counter automaton enforces the conditions described above using guard constraints followed by question marks, and $x++$ and $x--$ as abbreviations for the assignments $x := x + 1$ and $x := x - 1$, respectively.

The subway counter automaton from Figure 2.2 can be translated into the Datalog program below. It expresses the semantics (combinations of states and state variable values) of the automaton using difference constraints.

```
//Subway Automaton
begin%RECURSIVE%
ONTIME(b,s,d) :- b=0, s=0, d=0.
ONTIME(b,s,d) :- STOPPED(b,s1,d), b-s1=1, s-s1=1.
ONTIME(b,s,d) :- ONTIME(b1,s,d), b1-s<9, b-b1=1.
ONTIME(b,s,d) :- ONTIME(b,s1,d), b-s1>-9, s-s1=1.
ONTIME(b,s,d) :- ONBRAKE(b,s1,d), b-s1=1, s-s1=1.
ONTIME(b,s,d) :- LATE(b1,s,d), b1-s=-1, b-b1=1.
ONBRAKE(b,s,d) :- ONTIME(b1,s,d1), b1-s=9, b-b1=1, d=0.
ONBRAKE(b,s,d) :- ONBRAKE(b1,s,d1), d1<9, b-b1=1, d-d1=1.
ONBRAKE(b,s,d) :- ONBRAKE(b,s1,d), b-s1>1, s-s1=1.
STOPPED(b,s,d) :- ONBRAKE(b1,s,d), d≤9, b-b1=1.
STOPPED(b,s,d) :- STOPPED(b,s1,d), b-s1>1, s-s1=1.
LATE(b,s,d) :- ONTIME(b,s1,d), b-s1=-9, s-s1=1.
LATE(b,s,d) :- LATE(b1,s,d), b1-s<-1, b-b1=1.
end%RECURSIVE%
```

Error Condition: Suppose that this automaton is correct if $|b - s| < 20$ in all states at all times. Then, this automaton is incorrect if $|b - s| \geq 20$ at least in one state at one time. The table below shows the result of the under-approximation using the MLPQ constraint database system.

MLPQ Under Approximation

BRAKE	LATE	ONTIME	STOPPED
$1 \leq b - s \leq 19$	$-10 \leq b - s \leq -1$	$-9 \leq b - s \leq 9$	$1 \leq b - s \leq 20$
$10 \leq b \leq 19$	$10 \leq s \leq 19$	$0 \leq b \leq 9$	$11 \leq b \leq 20$
$0 \leq s \leq 18$	$0 \leq d \leq 9$	$0 \leq s \leq 18$	$0 \leq s \leq 9$
$0 \leq d \leq 9$		$0 \leq d \leq 9$	$0 \leq d \leq 9$

The above was obtained by using an approximation bound of $l = 30$. If l is increased, then the upper bounds of b and s increase. Therefore, in the limit, those upper bounds can be dropped.

Further, for any value of u, since the above is an under-approximation, any possible integer solution of the constraints below the state names *must occur* at some time. For example, the STOPPED relation must contain the case $b - s = 20$ at some time. Therefore, this automaton is incorrect by our earlier assumption.

The Verimag laboratory has software for testing program correctness using abstract interpretation. Halbwachs (1993) gave the following over approximation derived using Verimag's software for the subway automaton.

Verimag Over Approximation

BRAKE	LATE	ONTIME	STOPPED
$1 \leq b - s \leq d + 10$	$-10 \leq b - s \leq -1$	$-9 \leq b - s \leq 9$	$1 \leq b - s \leq 19$
$d + 10 \leq b$	$s \geq 10$	$b \geq 0$	$19 \leq 9s + b$
$0 \leq d \leq 9$		$s \geq 0$	$b \geq 10$

Surprisingly, this result does not match our result. In particular, the over approximation for the STOPPED state contains the constraint $b - s \leq 19$, which says that the value of $b - s$ cannot be 20, but our lower bound says that 20 must be one of the cases. To resolve this apparent contradiction, we need to look more closely at the automaton. We can see that the following is a valid sequence of transitions, where $S(b, s, d)$ represents the values of b, s, and d is each state $S \in \{$BRAKE, INITIAL, LATE, ONTIME, STOPPED$\}$.

$$\text{INITIAL}(0,0,0) \longrightarrow \text{ONTIME}(0,0,0) \longrightarrow \text{ONTIME}(1,0,0) \longrightarrow \text{ONTIME}(2,0,0) \longrightarrow$$
$$\text{ONTIME}(3,0,0) \longrightarrow \text{ONTIME}(4,0,0) \longrightarrow \text{ONTIME}(5,0,0) \longrightarrow \text{ONTIME}(6,0,0) \longrightarrow$$
$$\text{ONTIME}(7,0,0) \longrightarrow \text{ONTIME}(8,0,0) \longrightarrow \text{ONTIME}(9,0,0) \longrightarrow \text{BRAKE}(10,0,0) \longrightarrow$$

Table 4.2. Subway automaton running times and memory usage.

Bound	**Over Approximation**	**Under Approximation**
22	1:55:09 @ 312,396KB	0:20:48 @ 50,164KB
21	1:43:28 @ 286,264KB	0:16:35 @ 43,308KB
20	1:33:16 @ 277,648KB	0:12:51 @ 32,364KB
19	1:15:50 @ 231,596KB	0:09:32 @ 31,028KB
18	1:14:35 @ 237,744KB	0:07:01 @ 25,900KB

$\textsc{Brake}(11,0,1) \longrightarrow \textsc{Brake}(12,0,2) \longrightarrow \textsc{Brake}(13,0,3) \longrightarrow \textsc{Brake}(14,0,4) \longrightarrow$
$\textsc{Brake}(15,0,5) \longrightarrow \textsc{Brake}(16,0,6) \longrightarrow \textsc{Brake}(17,0,7) \longrightarrow \textsc{Brake}(18,0,8) \longrightarrow$
$\textsc{Brake}(19,0,9) \longrightarrow \textsc{Stopped}(20,0,9)$

Note that $\textsc{Stopped}(20,0,9)$ contradicts the first constraint in the Verimag over approximation for the Stopped state. Hence, we suspect that the Verimag software contains some bug or there was some problem in data entry.

But is the MLPQ bound tight? We made several runs with different l values ranging from 10 to 30 for both over and under approximations. By increasing the l value to 20 alone and performing the evaluation with both approximations, we derive a tight bound $-10 \leq b - s \leq 20$ across the four constraint relations.

The running times and memory usage for various runs are given in Table 4.2. These experiments were run on an AMD X2 64bit computer with 1 GB of RAM.

4.4. Conclusions and Future Work

Using constraint databases with the approximation techniques implemented, we verified correctness and falsified correctness of Datalog programs and described a

framework for translating imperative programs into Datalog with addition constraints. Using these techniques we have given tighter bounds in Section 4.1 than the previous method of Miné. In Section 4.3 we discovered errors in published results of the Verimag system. We have shown that our method can find tight bounds for constraints involving unbounded variables in the subway automaton. With large result sets representing the semantics of programs, the constraint database approach provides a natural way to find error conditions using the standard query languages: Datalog and SQL. However there are limitations to the use of our system. 1) In the worst case the system runs in exponential time of the input. 2) The system uses quite a bit of memory, which depends on l and the query. However, it is still a good alternative to other systems when they do not give satisfactory results.

Eliminating disjoint and non-convex regions from the abstract representation has been the approach to software verification up to this point. In the future we wish to explore abstract domains with limited numbers of non-convex, disjoint regions. We are currently working on a project called VODAC (Verification Of Datalog with Addition Constraints) specifically to continue this research area. This program will also allow research in the area of query optimization and indexing of constraint databases with addition constraints.

Part 2

Spatiotemporal Aggregation

CHAPTER 5

Literature Review

This chapter reviews the literature specific to aggregation including estimation and indexing techniques. Spatial and spatiotemporal databases have attracted an enormous amount of interest, and there exists a wide range of literature with only a titular relationship to our work. For books on the subjects of spatiotemporal and constraint databases we suggest: Rigaux et al. (2001), Revesz (2002), Samet (1990, 2005), and Guting & Schneider (2005).

5.1. Threshold and Other Spatiotemporal Aggregation

There exists only a few previous algorithms to compute MaxCount (Revesz & Chen 2003, Chen & Revesz 2004, Anderson 2006). None of those previous algorithms provides efficient queries without rebuilding the index (i.e., they do not provide dynamic updates).

Previous *approximate* MaxCount solutions use indices (Acharya et al. 1999) that minimize the skew of point distributions in the buckets by creating hyper-buckets based on the properties of all points at index creation time. Updates require the index to be rebuilt because the buckets depend on the point distribution at a specific time. In contrast, the probabilistic method we present *recognizes* point density skew in each bucket instead and creates a density distribution to model it. We present the first efficient and dynamic algorithm for MaxCount. Table 5.1

compares the results of earlier MaxCount algorithms with our current algorithm where N is the number of points and B is the number of buckets in the index.

Table 5.1. MaxCount aggregation complexity on linearly moving objects.

Max. Dim.	**Worst Case Time**	**Space**	**Exact or Est.**	**Static or Dynamic**	**Reference**
1	$O(log\ N)$	$O(N^2)$	Exact	Static	Revesz & Chen (2003)
1	$O(B \log B)$	$O(B)$	Est.	Static	Chen & Revesz (2004)
2	$O(B \log B)$	$O(B)$	Est.	Static	Anderson (2006)
d	$O(B)$	$O(B)$	Est.	Dynamic	Dissertation
d	$O(N)$	$O(1)$	Exact	Dynamic	Dissertation

We present the following threshold aggregation operators formally defined below:

(1) MaxCount

(2) ThresholdRange

(3) ThresholdCount

(4) ThresholdSum

(5) ThresholdAverage

Definition 5.1 (MaxCount (MinCount))**.** *Let S be a set of moving points. Given a dynamic query space R defined by two moving points Q_1 and Q_2 as the lower-left and upper-right corners of R, and a time interval T, the* MaxCount *(*MinCount*) operator finds the time $t_{\max(\min)}$ and maximum (or minimum) number of points $M_{\max(\min)}$ in S that R can contain at any time instance within T.*

Throughout this dissertation we develop the MaxCount operator because wherever we find a maximum in the dissertation, a minimum can be found similarly.

Definition 5.2 (ThresholdRange)**.** *Let S be a set of moving points. Given a dynamic query space R defined by two moving points Q_1 and Q_2 as the lower-left and upper-right corners of R, a time interval T, and a threshold value M, the* ThresholdRange *operator finds the set of time intervals T_M where the count of objects in R is larger than M.*

ThresholdRange is directly related to MaxCount in that when M is raised to $M_{\max}$, then ThresholdRange returns a time interval containing $t_{\max}$ and during this time interval, the count will be $M_{\max}$.

Definition 5.3 (ThresholdCount)**.** *Given a* ThresholdRange*,* ThresholdCount *returns the number of time intervals.*

Definition 5.4 (ThresholdSum)**.** *Given a* ThresholdRange*,* ThresholdSum *returns the total time T_s during which the count is above M. That is, for each $T_i \in T_M$,* ThresholdSum *return:*

$$T_s = \sum_i |T_i| \tag{5.1}$$

where $|T_i|$ means the length of the interval.

Definition 5.5 (ThresholdRange)**.** *Given a* ThresholdRange*,* ThresholdAverage *returns the average length of the intervals in T_M.*

To our knowledge, we present the first proposal of these threshold aggregate operators for moving points. Together MaxCount (and MinCount), ThresholdRange, ThresholdCount, ThresholdSum, and ThresholdAverage form

a complete set of threshold aggregation operators comparable to the aggregation operators given in relational databases.

In addition to the threshold aggregation operators, we also use our bucketing method to implement the CountRange defined as follows.

Definition 5.6 (CountRange)**.** *Let S be a set of moving points. Given a dynamic query space R defined by two moving points Q_1 and Q_2 as the lower-left and upper-right corners of R and a time interval T, the* CountRange *query returns the total number of points that intersect R in T.*

Definition 5.7 (Spatiotemporal Range)**.** *Let S be a set of moving points. Given a rectangular query space R defined by two points Q_1 and Q_2 as the lower-left and upper-right corners of R, and a time interval T, the* Spatiotemporal-Range *query returns the objects that intersect R in T.*

Notice that Spatiotemporal-Range differs from our implementation in that it does not allow Q_1 and Q_2 to move. In our implementation, we consider the more general case where the query space can move.

We can modify Spatiotemporal-Range algorithms to return the CountRange by counting the objects returned. Several other algorithms were proposed directly for the CountRange problem. We summarize previous Spatiotemporal-Range and CountRange algorithms in Table 5.2, where N is the number of moving objects or points in the database, d is the dimension of the space, and B is the number of buckets. All algorithms listed are dynamic, which means that they allow insertions and deletions of moving objects without rebuilding the index.

Table 5.2. RANGE and COUNTRANGE aggregation summary.

Max. Dim.	**Worst Case Time**	**Worst case Space**	**Exact or Est.**	**Reference**
2	$O(N^{\frac{3}{4}+\epsilon}+k)$	$O(N)$	Exact	Kollios et al. (1999)
2	$O(\log_2 N+k)$	$O(N^2)$[1]	Exact	
2	$O(N)$	$O(N)$	Exact	Papadopoulos et al. (2002)
3	$O(N)$	$O(N)$	Exact	Saltenis et al. (2000)
d	$O(N)$	$O(N)$	Exact	Porkaew et al. (2001)
d	$O(B^{d-1}\log_B^d N)$	$O(\frac{N}{B}\log_B^{d-1} N)$	Exact	Zhang et al. (2003)
2	$O(\log_B N+C)/B$	$O(N)$	Est.	Kollios et al. (1999)[2]
2	$O(B)$	$O(B)$	Est.	Choi & Chung (2002)
d	$O(B)$	$O(B)$	Est.	Tao et al. (2003)
d	$O(\sqrt{N})$	$O(N)$	Est.	Tao & Papadias (2005)
d	$O(B)$	$O(B)$	Est.	This Dissertation

In all our work we consider time as a continuous variable. Time as a discrete variable is discussed in both temporal and spatiotemporal aggregation by Agarwal et al. (2003), Tao & Papadias (2005) and Bohlen et al. (2006). In the discrete approach, time stamps describe the temporal nature of objects. This approach is less relevant to our work, but is relevant to many applications.

5.2. Indices and Estimation Techniques

There are many ways our work is indirectly related to previous work on indexing structures and estimation techniques. COUNT and MAX aggregation operators have only a titular relationship to the MAXCOUNT aggregation, because one cannot

[1]This is a restricted future time query with expected $O(N)$ space that becomes quadratic if the restriction is too far into the future.

[2]$C = K + K'$, where K' is the approximation error.

[3]Although Tao, Sun & Papadias (2003) allow dynamic updates, over time the index must be rebuilt.

use the COUNT and MAX aggregation operators to implement the MAXCOUNT aggregation. Nevertheless, several techniques used in the MAXCOUNT problem are also used in other indices and algorithms designed for range, max/min, and count queries. We summarize several of these related techniques next.

5.2.1. Indices

The index structure of Agarwal et al. (2003) finds the 2-dimensional moving points contained in a rectangle in $O(\sqrt{N})$ time. Gunopulos et al. (2005) gave a selectivity estimation with a histogram structure of overlapping buckets designed to approximate the density of multi-dimensional data. The algorithm runs in constant time $O(d|B|)$, where d is the number of dimensions and B is the number of buckets. Gupta et al. (2004) gave a technique for answering spatiotemporal range, intercept, incidence, and shortest path queries on objects that move along curves in a planar graph. Civilis et al. (2004, 2005) also gave indexing methods that use networks, such as roads, to predict position and motion changes of objects that follow roads and characteristics of routes. Zhang et al. (2001) proposed the *multiversion SB-tree* to perform range temporal aggregates: SUM, COUNT and AVG in $O(\log_b n)$, where b is the number of records per block and n is the number of entries in the database. Revesz (2005) gave efficient rectangle indexing algorithms based on point dominance to find count interpreted in k dimensions using the following concepts:

(1) *stabbing* gives the number of objects that contain a point;

(2) *contain* gives the number of rectangles that contain the query rectangle;

(3) *overlap* gives the number of rectangles that overlap the query rectangle; and

(4) *within* gives the number of rectangles within the query space.

These four operators have a running time of $O(\log^k n)$ where k is the number of dimensions and n is the number of points.

Saltenis et al. (2000) gave an R*-tree based indexing technique for 1, 2, and 3 dimensional moving objects that provide time-slice queries (selection queries), windows queries, and moving queries. Window queries return the same information as range queries, but with a valid time window starting at the current time and continuing to t_h. Window queries may request predictions for range queries within this window of time. Moving queries, similar to incidence queries, return the points that are contained within the space connecting one rectangle at a start time to a second rectangle at an end time. The proposed time parameterized R-tree (TPR-Tree) search runs in expected *logarithmic time*. Another R*-tree extension given by Cai & Revesz (2000) forms tighter parametric bounding boxes than Saltenis et al. (2000) and has similar running time. Tao, Papadias & Sun (2003) proposed the TPR*-Tree that extends the TPR-Tree with improved insert and delete algorithms. In the context of a variety of count queries it performs similarly to previous indices.

Recently, Pelanis et al. (2006) proposed the R^{PPF}-tree that indexes past, present and predictive positions of moving points, and extends the previous work on TPR-Trees (Saltenis et al. 2000) with a partial persistence framework. Earlier work by Tayeb et al. (1998) adapted the PMR-quadtree (Samet 1990), a variant of the quadtree structure, for indexing moving objects to answer time-slice queries, which

they called instantaneous queries, and infinitely repeated time-slice queries, called continuous queries. Search performance is similar to quadtrees and allows searches in $O(\log N)$ time.

Mokhtar et al. (2002) use the sweeping technique from computational geometry to define a query language to evaluate past, present, and future positions of moving objects in constraint databases.

Finally, Hadjieleftheriou et al. (2003) use an efficient approximation method to find areas where the density of objects is above a specific threshold during a specific time interval. This method comes the closest to the method used in our aggregation operators, but does not allow for the query to move or change shape over time. In fact, this method is not applied to counting at all.

Note that each of these indexing methods that return the moving points in a query window or rectangle can be easily modified to return instead the *count* of the number of moving points. However, they may not be easily extended to provide a MaxCount within a changing, moving query space.

With a few exceptions you can see that Count aggregation is $O(\log N + d)$ for exact methods and $O(B)$ or better for estimation methods. The hidden constant in the exact method is the number of buckets that must be traversed to find the Count. Estimation methods vary in many ways and asymptotic running time doesn't always give a meaningful estimate as to how big B will be.

5.2.2. Estimation Techniques

Our work is related to several other papers that *estimate* the count aggregate operation on spatiotemporal databases.

Acharya et al. (1999) gave an algorithm that can estimate the COUNT of the number of the rectangles that intersect a query rectangle for Selectivity Estimation. Choi & Chung (2002) and Tao, Sun & Papadias (2003) proposed methods that can estimate the COUNT of the moving points in the plane that intersect a query rectangle. More recently, Kollios et al. (2005) gave a predictive method based on dual transformations.

Wolfson & Yin (2003) and Trajcevski et al. (2004) gave a method for generating pseudo trajectories of moving objects. Most of these estimation algorithms use *buckets* as basic building structures of the index. In extending this idea, we use $2d$-dimensional hyper-buckets in our algorithms where d is the number of dimensions in the moving-objects space.

CHAPTER 6

Dynamically Indexing Linear Motion

We present an updatable *skew-aware* bucket for indices that models the skewed point distributions in each bucket. The skew-aware technique allows the index structure to perform inserts, deletes, and updates in *fast constant time* using a HashTable to store the buckets. Many spatiotemporal applications, such as tracking clients on a wireless network, particularly need these fast updates and no other MaxCount presented prior to this can meet that requirement. Because the buckets are spatially defined, the bucketing technique also easily adapts to other spatial and spatiotemporal indices such as the R-tree Guttman (1984). Hence the technique performs well for applications where search operations or update operations occur more frequently by using an appropriate index.

Our algorithm uses a sweeping method to evaluate the threshold aggregation operators similar to previous approaches from Chen & Revesz (2004), Revesz & Chen (2003) and Anderson (2006). The algorithm differs in that the sweeping algorithm integrates a skew-aware density function over the spatial dimensions of the bucket to obtain the time dependent count function. The density function in the bucket increases accuracy over methods given in (Chen & Revesz 2004, Anderson 2006) while maintaining the same number of buckets. This idea is a crucial improvement because we model the point distribution skew in a bucket, whereas previous methods modeled skew by increasing the number of buckets. We also present a precise

algorithm for evaluating the threshold aggregation operators that requires no index and runs in $O(N)+O(n \log n)$ time and $O(n)$ space where N is the number of points in the database and n is the value of a CountRange query using the same query space and time. Both the threshold aggregation algorithms and the skew-aware index presented in this dissertation are implemented and analyzed in 3-dimensional space. We show that the approximation achieves good results while significantly reducing the running times.

Section 6.1 describes the problems related to creating hyper-buckets (also referred to as just buckets) and a specific solution for creating 6-dimensional buckets for 3-dimensional linearly moving points. In all cases, we can extend our method to d-dimensions. Section 6.2 describes the method for inserting and deleting a point from a bucket. Section 6.3 applies two different data structures to contain the buckets suited for applications where either inserts and deletes or threshold aggregation queries dominate.

6.1. Hyper-Buckets: Creating the Buckets

Definition 6.1 (Hex Representation)**.** *Define each 3-dimensional linearly moving point p by parametric linear equations in t as follows:*

$$p = \begin{cases} p_x &= v_x t + x_0 \\ p_y &= v_y t + y_0 \\ p_z &= v_z t + z_0 \end{cases} \tag{6.1}$$

where the corresponding hex representation *of p is the tuple $(v_x, x_0, v_y, y_0, v_z, z_0)$. For simplicity we often denote the six-tuple as $(x_1, ..., x_6)$.*

Consider a relation $D(x_1, .., x_6)$ that contains the *hex representation* of linearly moving points in 3 dimensions. Then D represents a 6-dimensional *static* space. Divide the space into axis-aligned hyper-rectangles where the k^{th} axis has d_k divisions on it.

Definition 6.2 (Hyper-bucket or Bucket)**.** *Define each bucket B_i by inequalities of the form:*

$$\begin{aligned} v_{x,L} \leq v_x < v_{x,U} \quad \bigwedge \quad x_{0,L} \leq x_0 < x_{0,U} \quad \bigwedge \\ v_{y,L} \leq v_y < v_{y,U} \quad \bigwedge \quad y_{0,L} \leq y_0 < y_{0,U} \quad \bigwedge \\ v_{z,L} \leq v_z < v_{z,U} \quad \bigwedge \quad z_{0,L} \leq z_0 < z_{0,U} \end{aligned} \tag{6.2}$$

where we denote the lower bound as:

$$(v_{x,L}, x_{0,L}, v_{y,L}, y_{0,L}, v_{z,L}, z_{0,L}) \tag{6.3}$$

and the upper bound as

$$(v_{x,U}, x_{0,U}, v_{y,U}, y_{0,U}, v_{z,U}, z_{0,U}). \tag{6.4}$$

Each hyper-rectangle defines the spatial dimensions of a possible bucket, where only buckets that contain points are included in the index. The maximum number of possible buckets is given by $m = \prod_k d_k$.

Definition 6.3 (Histograms)**.** *Given a* 6*-dimensional bucket B_i containing b_i points, build the* histograms $h_{i,1}$,...,$h_{i,6}$ *for each axis using s subdivisions as follows. To create histogram $h_{i,j}$, divide bucket B_i into s parallel subdivisions along the jth*

axis, and record separately the number of points within B_i that fall within each subdivision.

Example 6.1 (Building Histograms)**.** Consider a set of 6-dimensional points projected onto the v_x, x_0 plane as shown in Figure 6.1. Assume that the number of

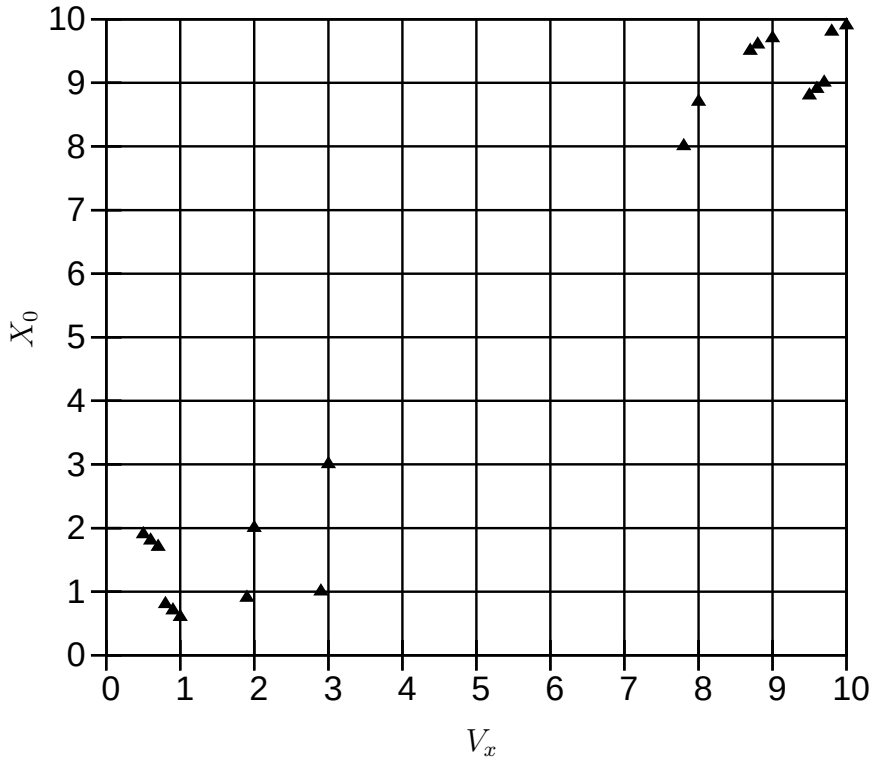


Figure 6.1. Points projected onto v_x, x_0 plane.

subdivisions is $s = 10$ along both v_x and x_0. Figures 6.2 and 6.3 show $h_{i,1}$ and $h_{i,2}$. For example, the subdivision $0 \leq v_x < 1$ contains six points and hence the first bar of histogram $h_{i,1}$ rises to level 6. The other values can be determined similarly.

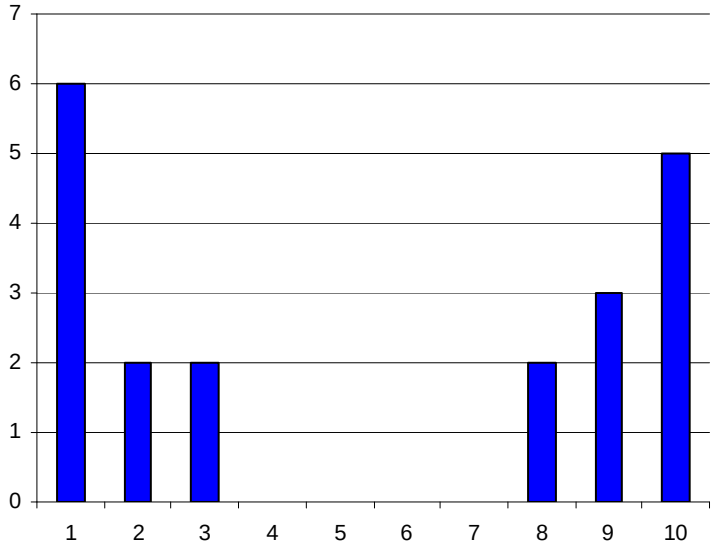


Figure 6.2. V_x histogram.

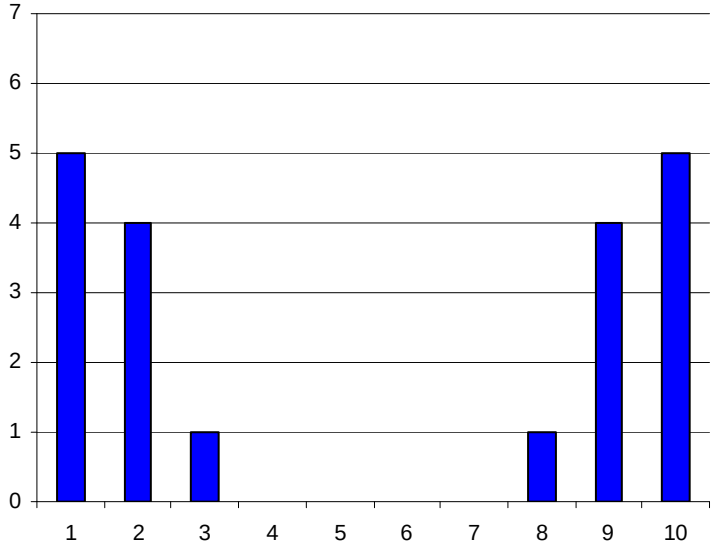


Figure 6.3. X_0 histogram.

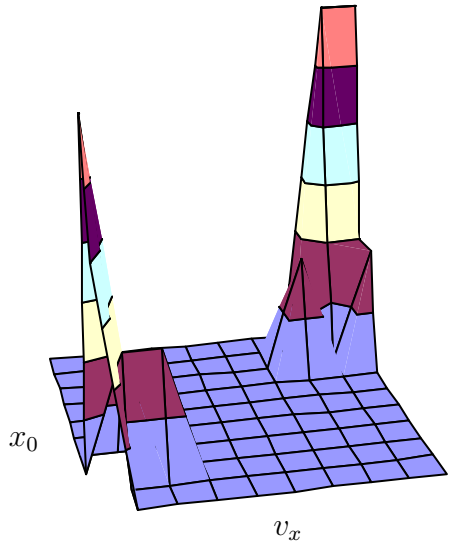


Figure 6.4. Accurate 2D distribution function.

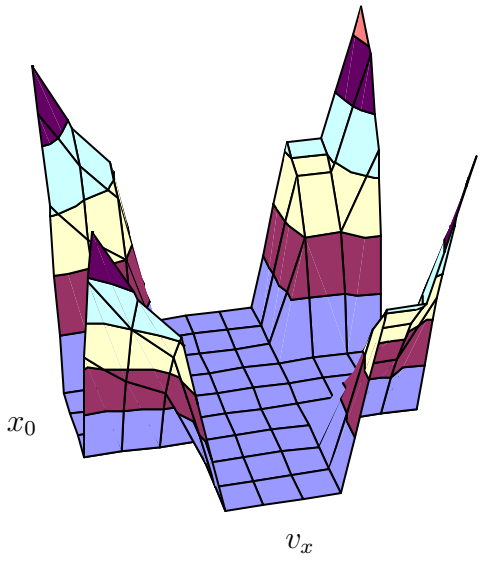


Figure 6.5. Inaccurate 2D distribution function.

Histograms tell much about the distribution of the points in a bucket but they introduce some ambiguity. For example, the histograms in Figures 6.2 and 6.3 match both of the $2d$-distributions in Figures 6.4 and 6.5.

Definition 6.4 (Axis Trend Function)**.** *The* axis trend function $f_{i,j}(x_j)$ *is some polynomial function for bucket* B_i *and axis* j *such that the following hold:*

(1) $f_{i,j} \geq 0$ *over* B_i*.*

(2) $f'_{i,j}$*, the derivative* $f_{i,j}$*, does not change sign over the valid range.*

The bucket trend function f_i *for bucket* B_i *is the following:*

$$f_i = \prod_j f_{i,j} \tag{6.5}$$

Condition 1 ensures that the bucket trend function, built from the axis trend functions, does not contain a negative probability region. Condition 2 requires that the bucket density increase, decrease, or remain constant when considering any single axis. This condition smoothes out the bumps in the point density of the buckets and gives a polynomial that approximates the point density.

Lemma 6.1. *Given a bucket* B_i *with bucket trend functions* $f_{i,j}$*, let* r_1 *and* r_2 *be identically sized regions in bucket* B_i*. If the density in* B_i *along each axis monotonically increases from* r_1 *to* r_2 *the following holds:*

$$\int_{r_2} f_i \; d\phi \geq \int_{r_1} f_i \; d\phi \tag{6.6}$$

Proof. Increasing densities from r_1 to r_2 translates into histograms that also increase from r_1 in the direction of r_2 along each axis. The translation from histograms to the axis trend functions gives the following conditions:

$$f_{i,j}(x_{2,j}) \geq f_{i,j}(x_{1,j}) \tag{6.7}$$

where $x_{1,j}$ and $x_{2,j}$ are the j^{th} coordinates of the points in r_1 and r_2 respectively, and are located the same distance from the j^{th} coordinates of the lower bounds of r_1 and r_2 respectively. Since this constraint holds for each j and $f_{i,j} \geq 0$ we have:

$$f_i(x_2) \geq f_i(x_1) \tag{6.8}$$

Hence by the properties of integration we conclude

$$\int_{r_2} f_i \; d\phi \geq \int_{r_1} f_i \; d\phi \tag{6.9}$$

□

Definition 6.4 allows a whole class of polynomial functions, and Lemma 6.1 applies to each member of that class. However, in the following, we use a particular polynomial function derived from the product of linear functions, which are obtained by using the least squares method for each histogram. This derivation is illustrated in Example 6.2.

Example 6.2 (Building Trend Functions)**.** Split Figure 6.1 into 4 buckets as shown in Figure 6.6 and assume $s = 5$. Correspondingly, split Figure 6.2. Then each histogram becomes two histograms. For example, the left side from (0 to 5) of histogram $h_{i,1}$ is the histogram for bucket C along the v_x axis. From the histograms build the axis trend functions along each axis for each bucket using the least squares method. From these build the bucket probability functions given by Equation (6.5).

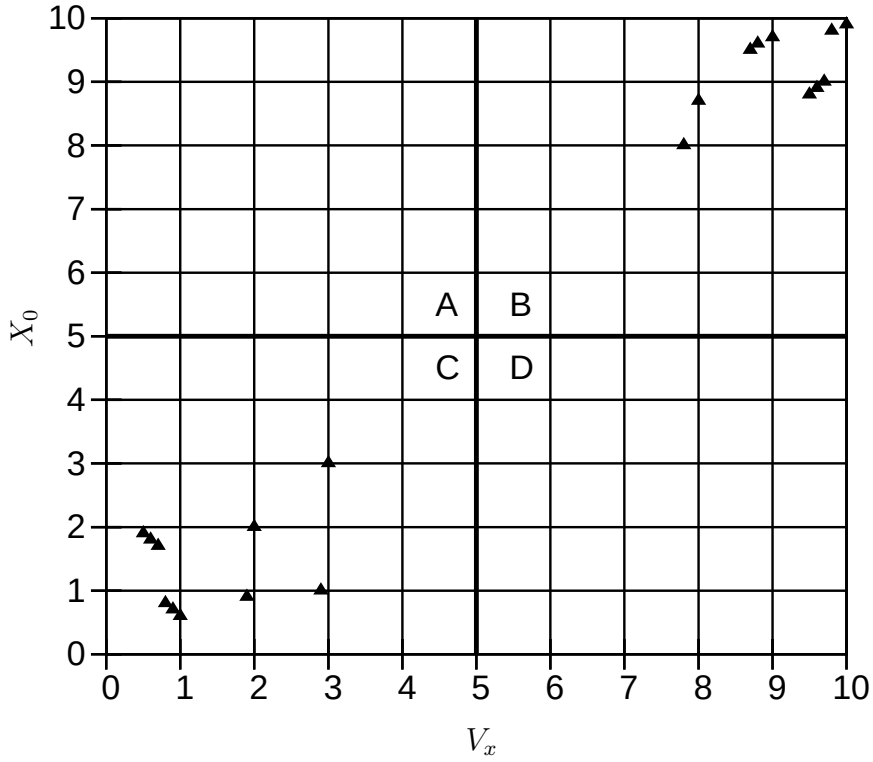


Figure 6.6. Points projected onto v_x, x_0 plane.

Clearly for buckets A and D, we have:

$$f_A = 0 \tag{6.10}$$

$$f_D = 0. \tag{6.11}$$

To find a linear equation of the form $y_i = ax_i + b$ where y_i is the approximated data, and x_i is the point of subdivision, the least squares method gives two linear

equations called the normal equations:

$$a\sum x_i^2 + b\sum x_i = \sum x_i y_i \tag{6.12}$$

$$a\sum x_i + bN = \sum y_i. \tag{6.13}$$

For bucket C and axis v_x, the normal equations become:

$$55a + 15b = 16 \tag{6.14}$$

$$15a + 5b = 10. \tag{6.15}$$

Of course, the numbers 55 and 15 in the above equations will not change since we always have histograms numbered such that $x_i = 1, ..., 5$. Solving these equations gives $a = -\frac{7}{5}$ and $b = \frac{31}{5}$. Hence, the axis trend function for bucket C, axis v_x is

$$f_{C,v_x} = -\frac{7}{5}v_x + \frac{31}{5}. \tag{6.16}$$

Similarly for the x_0 axis:

$$f_{C,x_0} = -\frac{7}{5}x_0 + \frac{32}{5}. \tag{6.17}$$

Since each function decreases from left to right, evaluate each function at the lower end point (5) to ensure property (1) of Definition 6.4:

$$f_{C,v_x}(5) = -\frac{4}{5} \tag{6.18}$$

$$f_{C,x_0}(5) = -\frac{3}{5}. \tag{6.19}$$

Thus, we must add $\frac{4}{5}$ to each axis trend function and f_C is calculated using Equation (6.5) as:

$$f_C = \frac{1}{25}(7v_x + 35)(-7x_0 + 36). \tag{6.20}$$

Definition 6.5 (Normalized Trend Functions)**.** *Let n be the number of points in the database, b_i the number of points in bucket B_i, and f_i be given by Equation (6.5). The* normalized trend function *F_i for bucket B_i is:*

$$F_i = \frac{b_i f_i}{n \int\limits_{B_i} f_i \; d\phi} \tag{6.21}$$

and the percentage of points *in bucket B_i is:*

$$p = \int\limits_{B_i} F_i \; d\phi. \tag{6.22}$$

Example 6.3 (Calculating Normalized Trend Functions)**.** Continuing from Example 6.2, integrating Equation (6.20) over the bucket gives:

$$\int\limits_0^5 \int\limits_0^5 f_C \; dx_0 dv_x = \frac{1295}{4}. \tag{6.23}$$

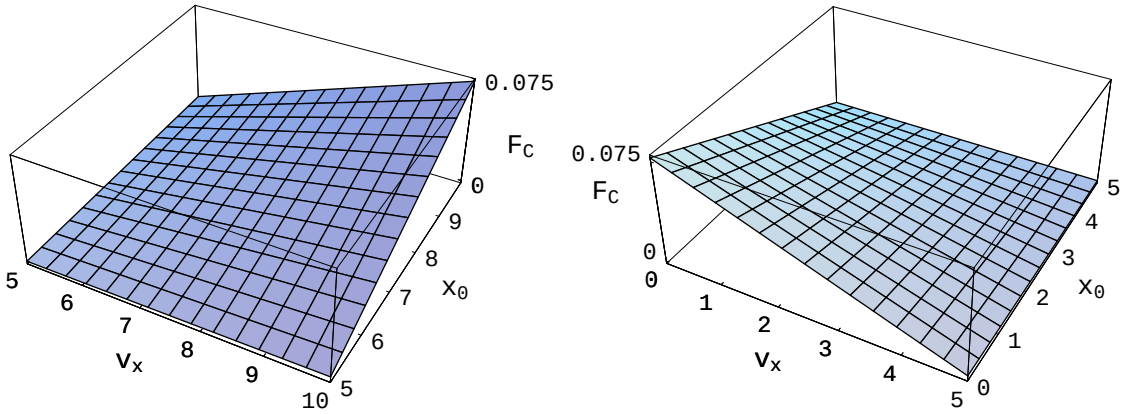
Given that $\frac{b_i}{n} = \frac{1}{2}$, calculating Equation (6.21) for bucket B_C gives:

$$\begin{aligned} F_C &= \frac{1}{2} \times \frac{4}{1295} \times \frac{1}{5}(7v_x + 35)(-7x_0 + 36) \\ &= \frac{2}{6475}(7v_x + 35)(-7x_0 + 36) \end{aligned} \tag{6.24}$$

A similar process on bucket B_B gives:

$$F_B = \frac{4}{251850}(13v_x - 61)(13x_0 - 63). \tag{6.25}$$

Equations (6.24) and (6.25) are shown in Figure 6.7.



(F_B) (F_C)

Figure 6.7. Normalized trend functions in the v_x, x_0 plane.

Lemma 6.2. *Let B_i be a bucket, n the number of points in the databases, and p be as defined in Equation (6.22). Then np is the number of points in bucket B_i.*

Proof. By Equation (6.21) and (6.22) we have:

$$\begin{aligned} np &= n \int_{B_i} F_i \, d\phi \\ &= n \int_{B_i} \frac{b_i}{n} \frac{f_i}{\int_{B_i} f_i \, d\phi} \, d\phi \\ &= n \frac{b_i}{n} \cdot \frac{\int_{B_i} f_i \, d\phi}{\int_{B_i} f_i \, d\phi} \\ &= b_i. \end{aligned} \tag{6.26}$$

Clearly the above calculations take only $O(1)$ time. □

6.2. Inserts and Deletes

We can maintain the index while deleting or inserting a point for any bucket B_i by recalculating the trend function F_i for the bucket.

Lemma 6.3. *Insertion and deletion of a moving point can be done in* $O(1)$ *time.*

Proof. When we insert or delete a point, we need to update the histograms and the normalized trend function. Let the point to insert/delete be P_a represented using the hex representation as $(a_0, a_1, a_2, a_3, a_4, a_5)$, let d_j, for $0 \leq j \leq 5$ be the bucket width in the j^{th}, and let s be the number of subdivisions in each histogram. The concatenation of $id_0, \ldots, id_5$ gives the ID_i of bucket i to insert (or delete) P_a into where each id_l and $0 \leq l \leq 5$ is defined by:

$$id_l = \left\lfloor \frac{a_l}{d_l} \right\rfloor. \tag{6.27}$$

The calculation of ID_i and retrieving bucket B_i takes $O(1)$ time using a HashTable.

Let $hw_{i,j}$ be the histogram-division width for the j^{th} calculated as $hw_{i,j} = \left\lceil \frac{d_j}{s} \right\rceil$. Then p is projected onto each dimension to determine which division of the histogram to update. For the j^{th} dimension the k^{th} division of histogram $h_{i,j}$ is given as follows:

$$k(j) = \left\lfloor \frac{a_j - id_j * d_j}{hw_k} \right\rfloor \tag{6.28}$$

Let $h_{i,j,k}$ be the histogram division to update for each histogram. Update $h_{i,j,k}$ and the sums $\sum y_i$, and $\sum x_i y_i$ from Equations (6.12) and (6.13). N, $\sum x_i$ and $\sum x_i^2$ do not need updating since the number of histogram divisions s is fixed within the database.

We can now recalculate each $f_{i,j}$ in constant time by solving the 2×3 matrix corresponding to Equations (6.12) and (6.13) for each histogram. For each $f_{i,j}$ calculate the endpoints to determine the required shift amount (Definition 6.4, property 1) and calculate f_i from Equation (6.5). Now we calculate F_i using Equation (6.5). Each of these steps depends only on the dimension of the database. Hence for any fixed dimension we can rebuild the normalized trend function F_i in $O(1)$ time. □

Example 6.4 (Updates)**.** Suppose we wish to delete point $(0.9, 0.9)$ from the points shown in Figure 6.1. Calculating the ID gives $Concatenate\left(\left\lfloor \frac{0.9-0}{1} \right\rfloor, \left\lfloor \frac{0.9-0}{1} \right\rfloor\right) = 0$ which is bucket C. Projecting this result down onto the axes gives us the histograms h_{0,v_x} and h_{0,x_0} on both axes: $k(v_x) = \left\lfloor \frac{0.9-0}{1} \right\rfloor = 0$ and $k(x_0) = \left\lfloor \frac{0.9-0}{1} \right\rfloor = 0$

Update the summation values indicated in Lemma 6.3 from Equations (6.12) and (6.13) as: $\sum y_i = \sum y_i - 1 = 9$ and $\sum x_i y_i = \sum x_i y_i - 1 * 1 = 15$. This calculation

gives us the following linear equations from (6.12) and (6.13).

$$55a + 15b = 15 \tag{6.29}$$

$$15a + 5b = 9 \tag{6.30}$$

Solving these equations gives $a = -6/5$ and $b = 27/5$, and we obtain $f_{C,v_0} = -\frac{6}{5}v_x + \frac{27}{5}$. Using a similar process gives $f_{C,x_0} = -\frac{6}{5}x_0 + \frac{28}{5}$. Checking the endpoints shows that $f_{C,v_0}(5) = -11.4$ is the smallest number. Adding 11.4 as a constant to f_{C,v_0} and f_{C,x_0} gives f_C as:

$$f_C = \frac{6}{25}(-v_x + 14)(-6x_0 + 85) \tag{6.31}$$

Integrating Equation (6.31) over the bucket gives $h = \int_0^5 \int_0^5 F_C \; dx_0 \; dv_x = 4830$. Hence, our recalculated normalized trend function for bucket C is:

$$F_C = \frac{2}{4025}(-v_x + 14)(-6x_0 + 85) \tag{6.32}$$

6.3. Index Data Structures

There is no need to create a bucket unless it contains at least one point. We consider two candidate data structures for organizing the buckets: HashTables and Trees.

For databases where inserts and deletes are the most common operation, the HashTable approach will allow these operations to run in constant time. However, the MaxCount operation will require an enumeration of all the buckets and thus

at least a running time of $O(B)$. As long as the number of buckets is reasonable, this approach works well.

For databases where MaxCount is the most common operation, use an R-tree structure (Guttman 1984, Beckmann et al. 1990) where the elements to be inserted are the buckets. This approach speeds up the MaxCount query to $O(\log |B| + R)$ where R is the number of buckets needed to calculate the query. The insert and delete costs for these R-trees are $O(\log |B|)$, because buckets do not overlap.

Since buckets do not change shape, the database is decomposable and allows each type of aggregation to be calculated from simultaneous executions on subspaces of the index space. We discuss the method and ramifications of this capability at the end of Section 7.4.

CHAPTER 7

Dynamic MaxCount

Section 7.1 reviews point domination in higher dimensions. Section 7.2 examines finding the percentage of points in a bucket that are in the query space as a function of time. Section 7.3 puts the two previous sections together to create the dynamic MaxCount algorithm for d-dimensions.

7.1. Point Domination in 6-Dimensional Space

Let B be the set of 6-dimensional hyper-buckets in the input where each hyper-bucket B_i has an associated normalized trend function F_i as in Definition 6.5. Let the vertices of B_i be denoted $v_{i,j}$ where $1 \leq j \leq 64$, because there are 2^6 corner vertices to a 6-dimensional hyper-cube.

Definition 7.1 (Point Domination)**.** *Given two linearly moving points in three dimensions*

$$P(t) = \begin{cases} p_x = x_1 t + x_2 \\ p_y = x_3 t + x_4 \\ p_z = x_5 t + x_6 \end{cases} \quad \text{and} \quad Q(t) = \begin{cases} q_x = v_x t + x_0 \\ q_y = v_y t + y_0 \\ q_z = v_z t + z_0 \end{cases} \tag{7.1}$$

$Q(t)$ dominates $P(t)$ if and only if the following holds:

$$(p_x < q_x) \quad \wedge \quad (p_y < q_y) \quad \wedge \quad (p_z < q_z). \tag{7.2}$$

The previous definition takes 6-dimensional points defined in Definition 6.1 and places them into three inequalities of the form $x_2 < -t(x_1 - v_x) + x_0$. Each inequality defines a region below a line with slope $-t$.

Definition 7.2 (x-view, y-view and z-view projections)**.** *Projecting the inequalities from Definition 7.1 onto their respective dual planes allows a visualization in three 2-dimensional planes. Define these three projections as the $x-$view, $y-$view and $z-$view respectively. Because the time $-t$ defines the slopes of each line, all views contain lines with identical slopes. (See Figure 7.1)*

Definition 7.3 (Query Space)**.** *Given two moving query points $Q_1(t)$ and $Q_2(t)$ and lines l_{x1}, l_{x2}, l_{y1}, l_{y2}, l_{z1}, l_{z2} crossing them in their respective hexes with slopes $-t$, the intersection of the bands formed by the area between l_{x1} and l_{x2}, l_{y1} and l_{y2}, and l_{z1} and l_{z2} in the 6-dimensional space forms a hyper-tunnel that defines the* query space *as shown in Figure 7.1.*

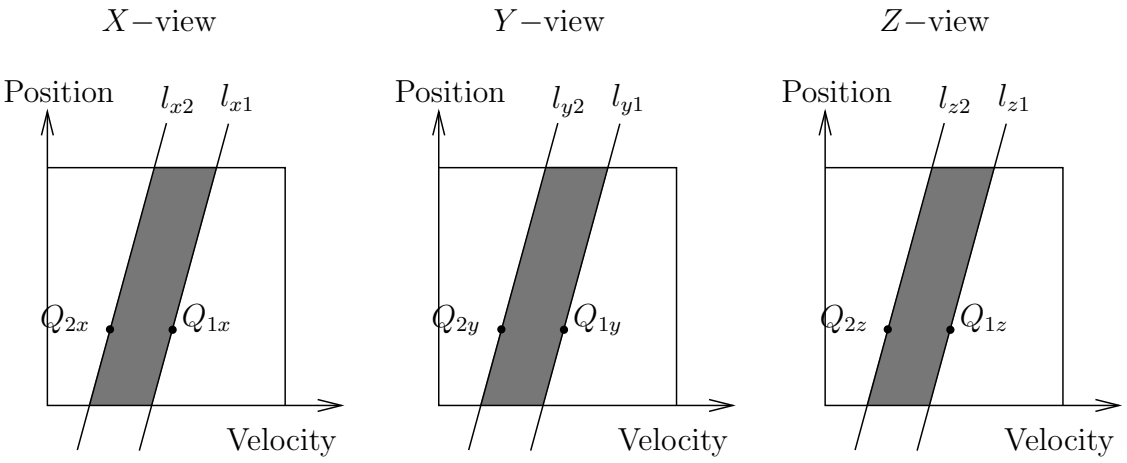


Figure 7.1. Views.

We can now visualize the query in space and time as the *query space* sweeping through a bucket as the slopes of the lines change with time. Using the above, it is now easy to prove the following lemma.

Lemma 7.1. *At any time t, the moving points whose hex-representation lies below (or above) l_{x1}, l_{y1} and l_{z1} in their respective views are exactly those points that lie below (or above) Q_1 in the original 3-dimensional plane.*

Proof. Let $Q_x(t) = v_x t + x_0$ where v_x and x_0 are constants and consider any x component of a point $P_x(t) = x_1 t + x_2$ that lies below Q on the x-axis. Then

$$x_1 t + x_2 < v_x t + x_0 \tag{7.3}$$

$$x_2 < -t(x_1 - v_x) + x_0 \tag{7.4}$$

Obviously, at any time t these are the points below the line $x_2 = -t(x_1 - v_x) + x_0$, which has a slope of $-t$ and goes through (v_x, x_0). This representation is the dual of point Q_x. By Definition 7.3, this is exactly the line l_{x1}. We can prove similarly that the points with duals above l_{x1} are above Q_1 at any time t. The proof that points whose hex-representations are above or below l_{y1}, and l_{z1} are exactly those points that lie above or below Q_1 is similar to the proof for points above or below l_{x1}. By Definition 7.1, we conclude that the points dominated by Q_1 in the dual space are those points that are below l_{x1}, l_{y1}, and l_{z1} in the x-view, $y-$view, and z-view, respectively. Similarly, we conclude that the points that dominate Q_1 in the dual space are those points that are above l_{x1}, l_{y1}, and l_{z1} in the x-view, $y-$view, and z-view, respectively. □

ID	Dimension 1		Dimension 2		Dimension 3	
	X0	X1	X2	X3	X4	X5
1	5.345	7.543	5.345	8.158	5.345	5.488
2	6.354	9.023	6.354	5.488	6.354	5.159
3	7.159	8.885	7.159	6.685	7.159	7.346
4	7.645	9.117	7.645	5.159	7.645	8.885
5	8.153	7.346	8.153	6.335	8.153	7.543
6	8.156	6.335	8.156	7.346	8.156	9.023
7	9.125	5.159	9.125	9.117	9.125	9.117
8	9.118	6.685	9.118	8.885	9.118	6.335
9	9.688	5.488	9.688	9.023	9.688	8.158
10	9.874	8.158	9.874	7.543	9.874	6.685

Figure 7.2. Example points.

Throughout the examples in this chapter, we use the points shown in Figures 7.2 and 7.3 – 7.5 to demonstrate the evaluation of a MaxCount query. We begin by creating the index.

Example 7.1 (Creating the Index)**.** Consider a relation that contains the 6-dimensional space 10 units $(0 \ldots 10)$ in each dimension. If we break this up into buckets that are 5 units long in each dimension, we have 2^6 buckets. Although these divisions make a space with 64 buckets, all the points are contained in a single bucket whose index is $(2, 2, 2, 2, 2, 2)$. All the points listed in Figure 7.2 have the same velocities for each dual plane. Notice the columns for x_1, x_3, and x_5 all have the same values in different orders. The projection of the points onto the 3 dual planes shown in Figures 7.3 – 7.5 does not immediately show this organization. Projecting the points onto each axis and creating histograms with 5 divisions is shown for the Velocity and Position axes in Figures 7.6 and 7.7.

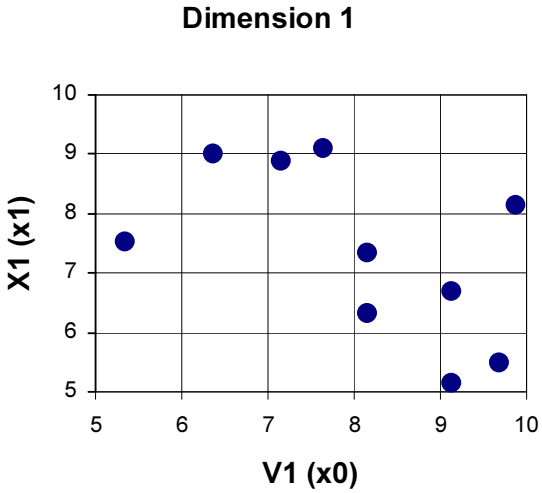


Figure 7.3. Points projected onto the X-view.

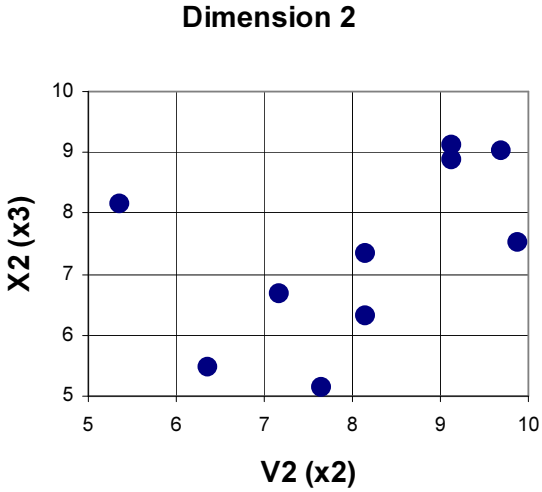


Figure 7.4. Points projected onto the Y-view.

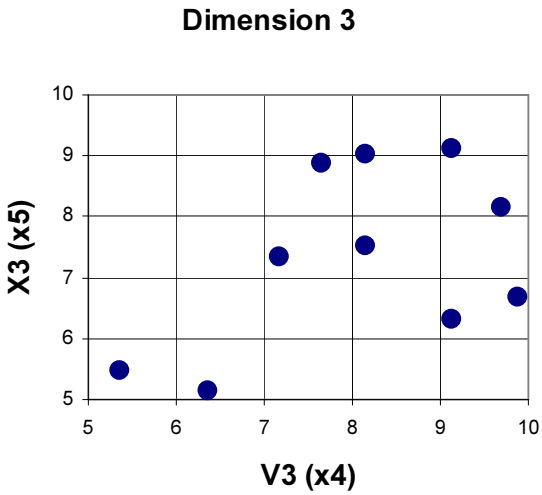


Figure 7.5. Points projected onto the Z-view.

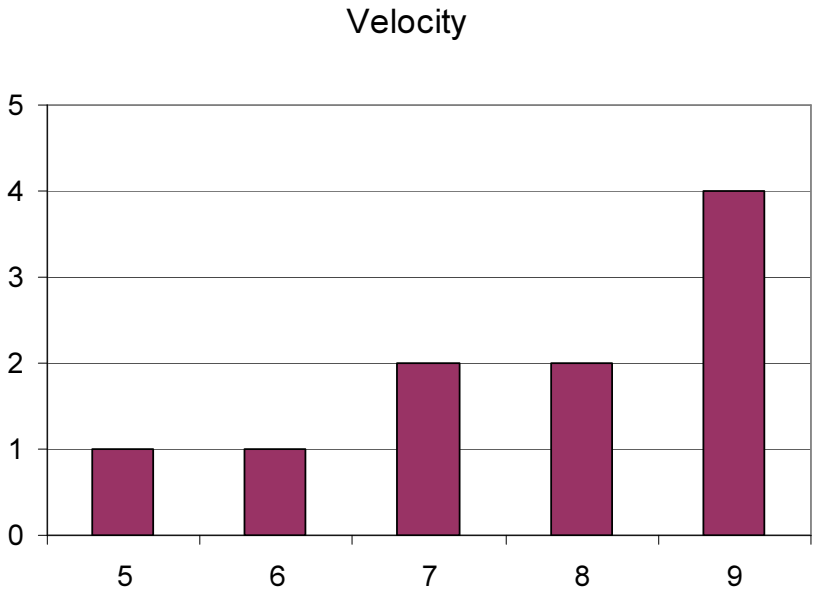


Figure 7.6. Velocity histogram.

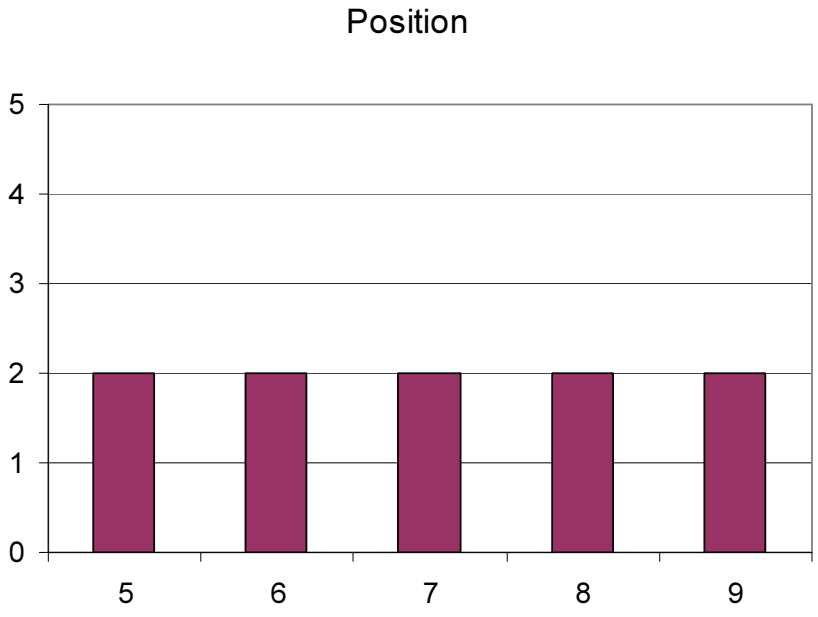


Figure 7.7. Position histogram.

Each velocity dimension has the same histogram. Similarly each position dimension has the same histogram. To create these histograms each point is projected onto the axis. For example point 1 projected onto the x_1 axis is given as:

$$5.345, 7.543, 5.345, 8.158, 5.345, 5.488 \rightarrow 5.345. \tag{7.5}$$

Calculate the widths of the histograms as:

$$Histogram\_Width = (10 - 5)/5 = 1 \tag{7.6}$$

We determine the histogram for each point by looping through the points as follows:

$$division = \lfloor((point - lowerbound)/Histogram\_Width)\rfloor \tag{7.7}$$

For example the lowest and highest points in velocity would be added to the division calculated as $\lfloor(5.84 - 5)/1\rfloor = 0$ and $\lfloor(9.468 - 5)/1\rfloor = 4$.

The histograms translate into a set of points for each view given as:

$$Velocity = \{(0, 1), (1, 1), (2, 2), (3, 2), (4, 4)\} \tag{7.8}$$

$$Position = \{(0, 2), (1, 2), (2, 2), (3, 2), (4, 2)\} \tag{7.9}$$

Before applying the least squares method each division number must be translated back into the bucket. Translation is done as follows:

```
for i ← 0 to number_of_divisions −1
    point[i][0] ← i ∗ histogram_width + lowerbound
    point[i][1] ← histogram_value[i]
end for
```

Translation of the points from (7.8) and (7.9) gives: The histograms for velocity and position in each view are given as:

$$Velocity = \{(5,1),(6,1),(7,2),(8,2),(9,4)\} \tag{7.10}$$

$$Position = \{(5,2),(6,2),(7,2),(8,2),(9,2)\}. \tag{7.11}$$

Using the least squares method to fit each of these to a line yields the following for each velocity and position dimension:

$$Velocity : y = 0.7x - 2.9 \tag{7.12}$$

$$Position : y = 0x + 2. \tag{7.13}$$

Evaluating Equations (7.12) and (7.13) at the end points to find the shift value for the axis trend function to add to each equation gives:

$$Velocity : y(5) = 1,\ y(10) = 4.3 \tag{7.14}$$

$$Position : y(5) = y(10) = 2. \tag{7.15}$$

In this case no constant needs to be added to our equation and the trend function becomes:

$$f_i = (0.7x_0 - 2.9)(0x_1 + 2)(0.7x_2 - 2.9)(0x_3 + 2)(0.7x_4 - 2.9)(0x_5 + 2) \quad (7.16)$$

Calculating F_i from Equation (6.21) requires integrating f_i over the bucket where $\int_{B_i} \equiv \int_5^{10} ... \int_5^{10}$ and where $d\phi \equiv dx_0 dx_1 dx_2 dx_3 dx_4 dx_5$ gives

$$\begin{aligned} \int_{B_i} f_i d\phi &= 8 \int_{B_i} (0.7x_0 - 2.9)(0.7x_2 - 2.9)(0.7x_4 - 2.9) d\phi \\ &= 1622234.375. \end{aligned} \quad (7.17)$$

Since all the points reside in a single bucket, $b_i = n$, the constant c is given by $c = 1/1622234.375 \approx 6.164 \times 10^{-7}$. Then F_i is given by

$$\begin{aligned} F_i &\approx c\ (0.7x_0 - 2.9)(0x_1 + 2)(0.7x_2 - 2.9)(0x_3 + 2)(0.7x_4 - 2.9)(0x_5 + 2) \\ &= 8c(0.7x_0 - 2.9)(.7x_2 - 2.9)(.7x_4 - 2.9) \end{aligned} \quad (7.18)$$

So far we have calculated the normalized trend function F_i for just one bucket. This calculation finishes the index creation process where the index contains a single bucket defined by $lowerbound = (5, 5, 5, 5, 5, 5)$ and $upperbound = (10, 10, 10, 10, 10, 10)$.

7.2. Approximating the Number of Points in a Bucket

As a line through a query point sweeps across a bucket, the points in the bucket that dominate the query point are approximated by the integral over the region

above the line. In each of the three views the query space intersects the plane giving the cases shown in Figure 7.8.

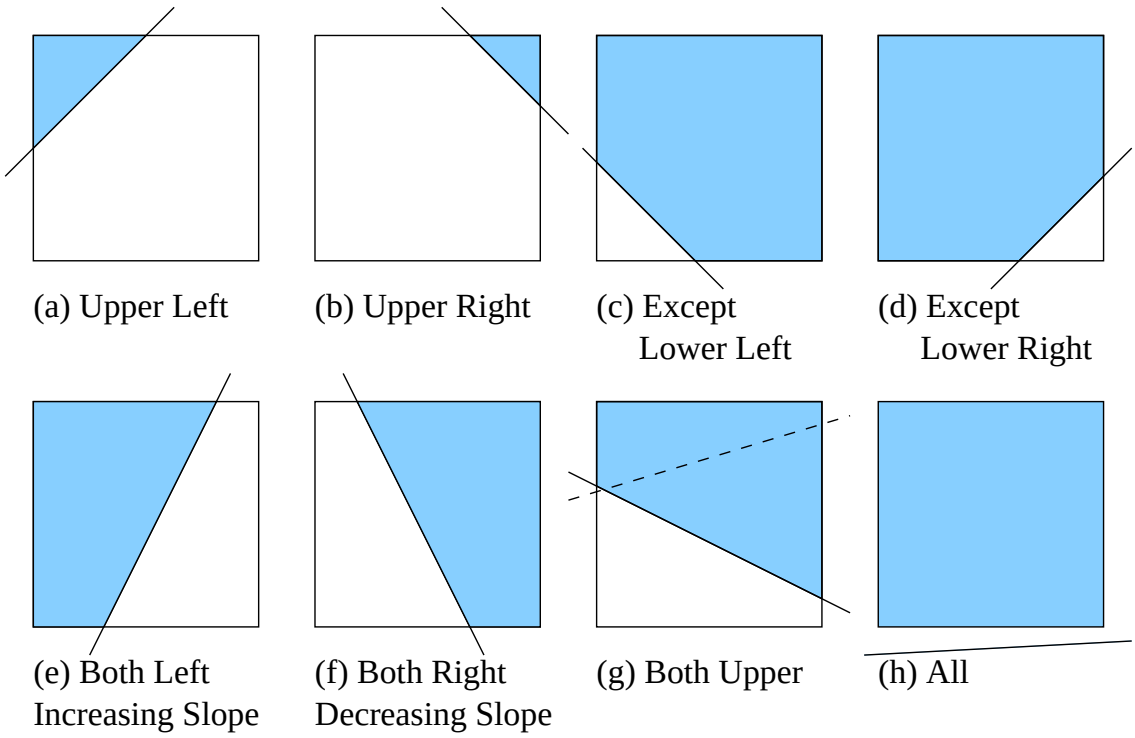


Figure 7.8. Sweep algorithm cases.

Definition 7.4 (Percentage Function)**.** *Integrating over the region above the line gives an approximation of the percentage of points in the query space. We define the percentage function given as:*

$$p = \int_{r_1} F_i \; d\phi \tag{7.19}$$

where r_1 is the region of the bucket in the query space. If two lines go through the same bucket we have the smaller region r_2 subtracted from the larger region r_1 as

follows.

$$\triangle p = \int_{r_1} F_i \; d\phi - \int_{r_2} F_i \; d\phi. \tag{7.20}$$

Here, regions r_1 and r_2 correspond to regions above Q_1 and Q_2 in Figure 7.1, respectively. Finding the number of points in the bucket requires multiplying Equation (7.19) or (7.20) by n.

For each case shown in Figure 7.8, we describe the function that results from integration in one view. To extend the result to any number of views, we take the result from the last view and integrate it in the next view. If the region below the line were desired, $p_{lower} = \frac{b_i}{n} - p$ gives the percentage of points below the line.

For cases (a)-(h) below, let $Q = (x_{1,q}, x_{2,q}, ..., x_{6,q})$. For the x-view, let the lower left corner vertex be $(x_{1,l}, x_{2,l})$ and the upper right corner vertex be $(x_{1,u}, x_{2,u})$. In addition each line denoted l is given by $x_2 = -t(x_1 - x_{i,q}) + x_{i+1,q}$ and corresponds to the lines shown in the corresponding case in Figure 7.8. Actual solutions for each of the integrals are given in the Appendix A and used in the implementation.

Case (a): For this case l crosses the bucket at $x_{1,l}$ and $x_{2,u}$. The integral over the shaded region is given by the following:

$$p_a = \int_{x_{1,l}}^{\frac{x_{2,u}-x_{2,q}}{-t}+x_{1,q}} \int_{-t(x_1-x_{1,q})+x_{2,q}}^{x_{2,u}} F_i \; dx_2 dx_1 \tag{7.21}$$

Notice that the lower bound of the integral over dx_2 contains x_1. This dependence within each view does not affect the integration in the remaining four dimensions.

The solution to Equation (7.21) has the form:

$$at^2 + bt + c + \frac{d}{t} + \frac{e}{t^2}. \tag{7.22}$$

Case (b): For this case l crosses the bucket at $x_{1,u}$ and $x_{2,u}$. The integral over the shaded region is given by:

$$p_b = \int_{-\frac{(x_{2,u}-x_{2,q})}{t}+x_{1,q}}^{x_{1,u}} \int_{-t(x_1-x_{1,q})+x_{2,q}}^{x_{2,u}} F_i \; dx_2 dx_1. \tag{7.23}$$

The solution has the form of Equation (7.22).

Case (c): For this case l crosses the bucket at $x_{1,l}$ and $x_{2,l}$. The integral over the shaded region above the line is given by:

$$p_e = \int_{x_{1,l}}^{\frac{x_{2,l}-x_{2,q}}{-t}+x_{1,q}} \int_{-t(x_1-x_{1,q})+x_{2,q}}^{x_{2,u}} F_i \; dx_2 dx_1 \; + \int_{\frac{x_{2,l}-x_{2,q}}{-t}+x_{1,q}}^{x_{1,l}} \int_{x_{2,l}}^{x_{2,u}} F_i \; dx_2 dx_1. \tag{7.24}$$

The solution has the form of Equation (7.22).

Case (d): For this case l crosses the bucket at $x_{1,u}$ and $x_{2,l}$. The integral over the shaded region is given by:

$$p_f = \int_{\frac{x_{2,l}-x_{2,q}}{-t}+x_{1,q}}^{x_{1,u}} \int_{-t(x_1-x_{1,q})+x_{2,q}}^{x_{2,u}} F_i \; dx_2 dx_1 \; + \int_{x_{1,l}}^{\frac{x_{2,l}-x_{2,q}}{-t}+x_{1,q}} \int_{x_{2,l}}^{x_{2,u}} F_i \; dx_2 dx_1. \tag{7.25}$$

The solution has the form of Equation (7.22).

Case (e): For this case l crosses the bucket at $x_{1,l}$ and $x_{1,u}$. The integral over the shaded region is given by:

$$p_c = \int_{x_{2,l}}^{x_{2,u}} \int_{x_{1,l}}^{\frac{x_2 - x_{2,q}}{-t} + x_{1,q}} F_i \; dx_1 dx_2. \tag{7.26}$$

The solution has the form of

$$c + \frac{d}{t} + \frac{e}{t^2} \tag{7.27}$$

which is like Equation (7.22) with $a = b = 0$.

Case (f): Similar to case(e), l crosses the bucket at $x_{1,l}$ and $x_{1,u}$. The integral over the shaded region is given by:

$$p_d = \int_{x_{2,l}}^{x_{2,u}} \int_{\frac{x_2 - x_{2,q}}{-t} + x_{1,q}}^{x_{1,u}} F_i \; dx_1 dx_2. \tag{7.28}$$

The solution has the form of Equation (7.27).

Case (g): For this case l crosses the bucket at $x_{1,l}$ and $x_{1,u}$. The integral over the shaded region is given by:

$$p_g = \int_{x_{1,l}}^{x_{1,u}} \int_{-t(x_1 - x_{1,q}) + x_{2,q}}^{x_{2,u}} F_i \; dx_2 dx_1. \tag{7.29}$$

The solution has the form

$$at^2 + bt + c \tag{7.30}$$

which is like Equation (7.22) with $d = e = 0$.

Case (h): The line l crosses below all the corner vertices hence the integral of the function is given as:

$$p_h = \int_{x_{1,l}}^{x_{1,u}} \int_{x_{2,l}}^{x_{2,u}} F_i \; dx_2 dx_1. \tag{7.31}$$

The solution has the form of Equation (7.30).

The above cases have solutions for each view in the form of Equation (7.22). Hence the percentage function for a single bucket as a function of t is of the form:

$$p = \left(a_x t^2 + b_x t + c_x + \frac{d_x}{t} + \frac{e_x}{t^2}\right)\left(a_y t^2 + b_y t + c_y + \frac{d_y}{t} + \frac{e_y}{t^2}\right) \left(a_z t^2 + b_z t + c_z + \frac{d_z}{t} + \frac{e_z}{t^2}\right) \tag{7.32}$$

where $t \neq 0$ when $d_x, d_y, d_z, e_x, e_y, e_z \neq 0$. Finally, renaming variables gives the general form:

$$p = a_6 t^6 + a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + c + \frac{d_1}{t} + \frac{d_2}{t^2} + \frac{d_3}{t^3} + \frac{d_4}{t^4} + \frac{d_5}{t^5} + \frac{d_6}{t^6} \tag{7.33}$$

where $t \neq 0$ when $d_i \neq 0$ for $1 \leq i \leq 6$. Since Equation (7.33) is closed under subtraction, $\triangle p$ from Equation (7.20) will also have the same form.

As the *query space* from Definition 7.3 sweeps through a bucket, it crosses the bucket corner vertices. Each time a corner vertex crosses the *query space* boundary, the case that applies may change in one or more of the views.

Definition 7.5 (Bucket and Index Time-Intervals)**.** *The span of time in which no vertex from bucket B_i enters or leaves the query space defines a* bucket time-interval. *We denote the time-interval as a half-open interval $[l, u)$ where l is the*

lower bound and u is the upper bound. Each bucket time-interval *has an associated percentage function $\triangle p$ given by Equation (7.20). We define the* index time-interval *similarly except that the span of time is defined when no vertex from* any *bucket in the index enters or leaves the query space.*

As we will see, index time-intervals are created from individual bucket intervals. Throughout the rest of this dissertation we use *time intervals* when the context clearly identifies which type we mean.

Definition 7.6 (Time-Partition Order)**.** *Let B be the set of buckets. Let Q_1 and Q_2 be two query points and $(t^{[}, t^{]})$ be the query time interval. We define the* Time-Partition Order *to be the set of ordered time instances $TP = t_1, t_2, ..., t_i, ..., t_k$ such that $t_1 = t^{[}$ and $t_k = t^{]}$, and each $[t_i, t_{i+1})$ is an* index time-interval.

Example 7.2 (Calculating Bucket Time-Intervals)**.** Continuing Example 7.1, let Q be a query defined by:

$$q_1 = (9.5,\ 8,\ 9.5,\ 8,\ 9.5,\ 8) \tag{7.34}$$

$$q_2 = (8.5,\ 5,\ 8.5,\ 5,\ 8.5,\ 5) \tag{7.35}$$

$$T = (0.1,\ 10) \tag{7.36}$$

where q_1 and q_2 form the query space over the query time interval T. To determine time intervals when corner vertices do not change, find the slopes of lines through both query points and each corner vertex of the bucket. Figure 7.9 shows lines from the two query points to the corner vertices for the first dimension. Since

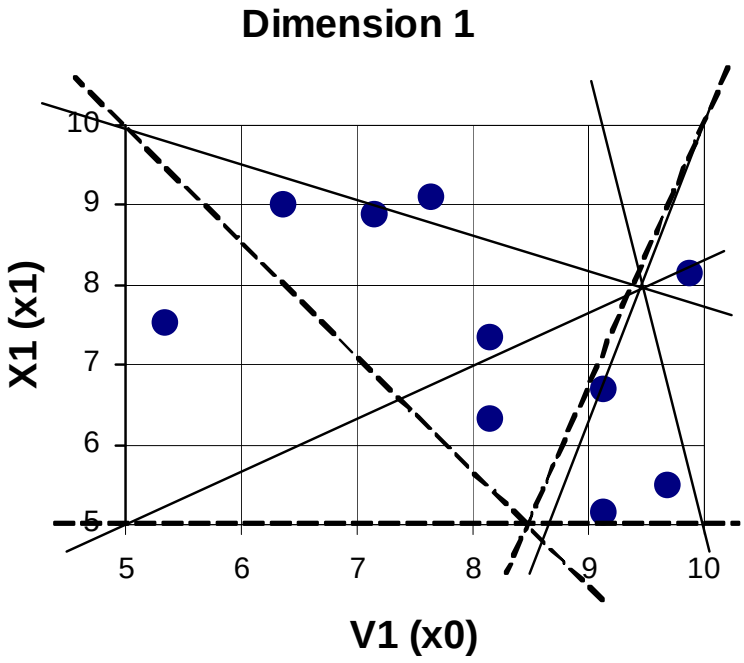


Figure 7.9. Lines from query points to corner vertices.

the query points are the same in each dimension each will appear the same. The set of times when lines through q_1 (shown as solid lines) cross corner vertices is $\{0.\overline{4}, 6\}$. The set of times when lines through q_2 (shown as dotted lines) cross corner vertices and are in the time interval is $\{1.42857\}$. The union of these two sets along with the end points makes up the times used to create the time intervals: $\{(.1, 0.\overline{4}), (0.\overline{4}, 1.42857), (1.42857, 6), (6, 10)\}$.

Integration over the *spatial dimensions* of the eight possible cases presented in Figure 7.8 gave a function of the form of Equation (7.33). *Maximizing* Equation (7.33) in the *temporal dimension* by first taking the derivative, we get:

$$\begin{aligned} \triangle p' &= (6a_6t^{12} + 5a_5t^{11} + 4a_4t^{10} + 3a_3t^9 + 2a_2t^8 + a_1t^7 \\ &\quad -d_1t^5 - 2d_2t^4 - 3d_3t^3 - 4d_4t^2 - 5d_5t - 6d_6)/t^7 \end{aligned} \tag{7.37}$$

where $t \neq 0$. Solving $\triangle p' = 0$ requires finding the roots of this 12-degree polynomial, *which is not possible using an exact method*. Hence we need a numerical method for solving the polynomial.

The following factors influenced the choice of the numerical method:

(1) Speed of the algorithm is more important than accuracy because we don't expect the original function to change dramatically over an index time-interval.

(2) The algorithm must converge toward a solution within the interval, that is the algorithm must be stable.

(3) Given that we are maximizing Equation (7.33) over a short time interval, we don't expect Equation (7.37) to have more than one solution. This assumption may seem naive, but it is reasonable given factor (1).

Factor (1) above is related to (3) in that it indicates that points close together have similar values, but emphasizes that speed is the goal. Factor (2) above eliminates several algorithms from consideration, but must be required to keep from choosing a solution that is not within the time interval evaluated.

Of the three points to consider, (3) is probably the least intuitive. Consider the following conjecture:

Conjecture 7.1. *Given p for a set of buckets, if the Euclidean distance between two maxima is small, then the difference between the maxima is small.*

Consider the physical characteristics of the system. The value of p over the time interval changes no more than b_i for any bucket B_i. Clearly p either increases as it encompasses more of the bucket or decreases at as it encompasses less of the bucket. When p represents the distribution over several buckets, each bucket contributes a decreasing or increasing amount over the time interval. Clearly p is bounded below by 0 and above by $\sum_i b_i$. Hence, the rate at which the derivative p' changes is characterized by the physical system and reflects the differences in the buckets as t changes. Since p does not change dramatically over t for any bucket, then change in several buckets over t will likewise not be dramatic. Hence if the distance between two maxima is small, the maxima have a small difference in magnitude. *This rational for the conjecture above is verified by the experiments.*

Based on these factors, we use a common method for the first approximation: we look at the graph of p'. Programmatically check c intervals of Equation (7.37) for a change in sign. If there exists a sign change, use the bisection method to find the root. If two points lie within ϵ of 0, we perform a check for each of these intervals when no change of sign is found. If some roots exist, we check them for maximal values along with the end points.

Lemma 7.2. *The approximate maximum within a time interval can be found in* $O(1)$ *time.*

Proof. Each *time interval* has an associated probability function $\triangle p$ which is calculated in $O(1)$ time. Finding $\triangle p' = 0$ also takes $O(1)$ time. By placing a constant bound on the number of iterations in the bisection method, we bound the time required in the numerical section of the algorithm by a constant. Plugging in the solution found by the bisection method along with the end points also takes $O(1)$ time. Hence, the running time to find the maximum within a bucket is $O(1)$. □

We chose to limit the number of iterations in the bisection method to 10, which limits the running time to a small constant value. This value was chosen based on empirical observation that index time-intervals remain small (about 0.01 to 4). Hence, using the bisection method allows us to narrow our search down to an interval at least as small as $\frac{1}{256}$ units of time. If time is measured in hours, this interval equates to only 14 seconds.

Example 7.3 (Building Time-Intervals and Finding MaxCount)**.** Continuing Example 7.2 we build the functions for time intervals

$$\{(.1, 0.\overline{4}), (0.\overline{4}, 1.42857), (1.42857, 6), (6, 10)\} \tag{7.38}$$

by integrating using the different cases from Figure 7.8.

Time Interval: $[0.1, 0.\overline{4}]$. Here case (c) holds for query point q_2 over this time interval. Hence the integral for query point q_2 and $t \in [.1, .\overline{4}]$ in each dimension is

given as:

$$\begin{aligned} p_c &= c\int_{8.5}^{10}\int_{5}^{10} 2(0.7x_0 - 2.9)dx_1dx_0 + \int_{5}^{8.5}\int_{-t(x_0-8.5)+5}^{10} 2(0.7x_0 - 2.9)dx_1dx_0 \\ &= 117.5 - 17.354\bar{6}t \end{aligned} \tag{7.39}$$

Case (g) holds for query point q_1 and thus the integral for query point q_1 and $t \in (.1, .\overline{4})$ in each dimension is given as:

$$\begin{aligned} p_g &= c\int_{5}^{10}\int_{-t(x_0-9.5)+8}^{10} 2(0.7x_0 - 2.9)dx_1dx_0 \\ &= 47.0 - 32.41\bar{6}t \end{aligned} \tag{7.40}$$

Hence the integral of the region is:

$$\begin{aligned} p &= c\,(p_c - p_g)^3 \\ &= 2.106 \times 10^{-3}t^3 + 2.957 \times 10^{-2}t^2 + 0.138t + 0.216 \end{aligned} \tag{7.41}$$

Evaluating p at the start and end of the time interval we have $p(0.1) \approx 0.23$ and $p(0.\overline{4}) = 0.28$. Figure 7.10 shows p in the time interval. Clearly p is increasing and consequently we have a maximum at the end point $t = 0.\overline{4}$. Since there are 10 points we must multiply $p(0.\overline{4})$ by 10 to get the approximation for the time interval as:

$$MaxCount_{0.1 \le t \le 0.\overline{4}} \approx 2.8. \tag{7.42}$$

Since we can not have partial points, we can round this result to 3.

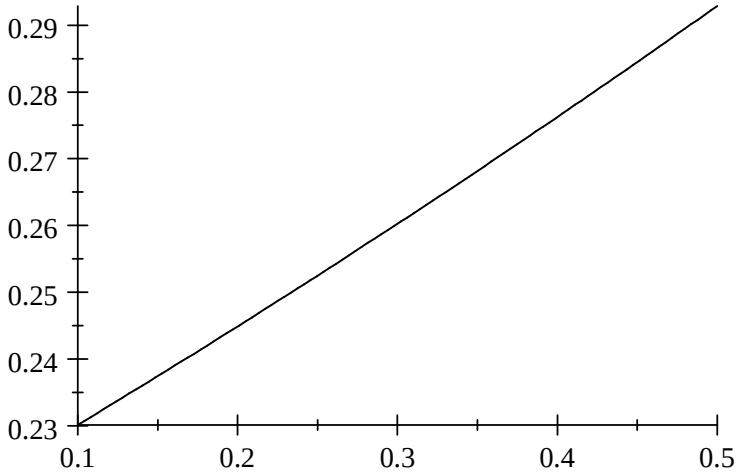


Figure 7.10. Graph of p, $0.1 \leq t \leq 0.\overline{4}$.

Time Interval $[0.\overline{4}, 1.428]$. Note that case (c) holds for query point q_2 over this time interval, hence p_c is given in Equation (7.39). Case (b) holds for q_1 over this interval. Therefore,

$$\begin{aligned} p_b &= \int_{\frac{-(10-8)}{t}+9.5}^{10} \int_{-t(x_0-9.5)+8}^{10} (0.7x_0 - 2.9)(2) dx_1 dx_0 \\ &= 0.995\overline{3}t + \frac{15}{t} - \frac{1.8\overline{6}}{t^2} + 7.85. \end{aligned} \tag{7.43}$$

Hence, the integral over the region is

$$\begin{aligned} p &= c\,(p_c - p_b)^3 \\ &\approx \frac{8.902 \times 10^{-2}}{t^2} - \frac{0.355}{t} - 0.417t + 6.827 \times 10^{-2} t^2 - \frac{1.355 \times 10^{-2}}{t^3} \\ &\quad - 3.808 \times 10^{-3} t^3 + \frac{1.483 \times 10^{-3}}{t^4} - \frac{9.665 \times 10^{-5}}{t^5} + \frac{4.009 \times 10^{-6}}{t^6} + 0.925. \end{aligned} \tag{7.44}$$

Checking the values gives $p(0.\overline{4}) \approx 0.28$ and $p(1.428) \approx 0.248$. Neither of these produce values larger than our current maximum. The graph of p in Figure 7.11 shows that there is no interior maximum.

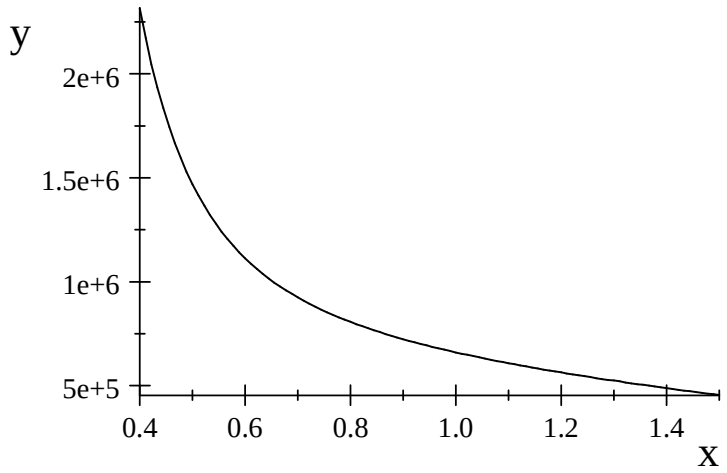


Figure 7.11. Graph of p, $0.\overline{4} \leq t \leq 1.428$.

Time Interval $[1.42857, 6]$. Note that case (b) holds for query point q_1 over this time interval, hence p_b is given in Equation (7.43). Case (f) holds for q_2 over this interval. Therefore,

$$\begin{aligned} p_f &= \int_5^{10} \int_{\frac{-(x_1-5)}{t}+8.5}^{10} (0.7x_0 - 2.9)(2)dx_0dx_1 \\ &= 53.625 + \frac{76.25}{t} - \frac{29.166}{t^2}. \end{aligned} \tag{7.45}$$

Hence, the integral is given by:

$$\begin{aligned} p &= c\,(p_f - p_b)^3 \\ &= \frac{0.235}{t} - 3.746 \times 10^{-3} t + \frac{0.217}{t^2} + 8.395 \times 10^{-5} t^2 - \frac{0.143}{t^3} - 6.088 \times 10^{-7} t^3 \\ &\quad - \frac{0.126}{t^4} + \frac{8.442 \times 10^{-2}}{t^5} - \frac{1.254 \times 10^{-2}}{t^6} + 4.875 \times 10^{-2}. \end{aligned} \tag{7.46}$$

As shown in Figure 7.12, we again have a decreasing function which has no interior maximums. Evaluating p at the end points gives $p(1.428) \approx 0.248$ and $p(6) \approx 0.073$. Therefore we keep our current maximum.

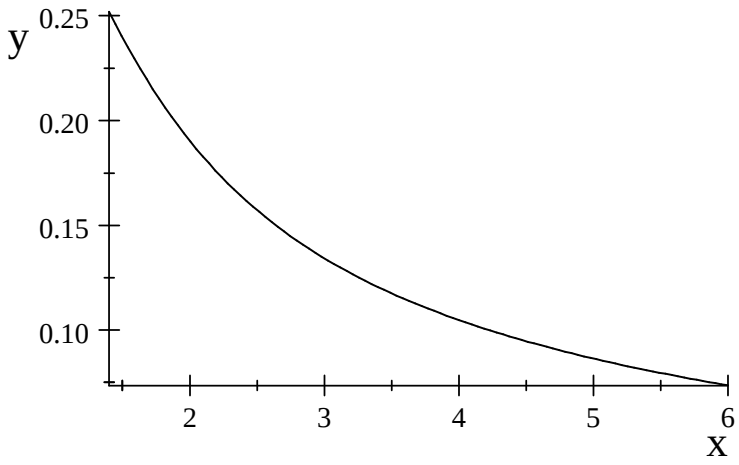


Figure 7.12. Graph of p, $1.428 \leq t \leq 6$.

Time Interval $[6, 10]$ Case (f) holds for both query points. Equation (7.45) gives the integral for q_2. The integral for q_1 is given as:

$$\begin{aligned} p_{f(q_1)} &= \int_5^{10} \int_{\frac{-(x_1-8)}{t}+9.5}^{10} (0.7x_0 - 2.9)(2)dx_0dx_1 \\ &= 19.625 - \frac{8.167}{t^2} - \frac{18.75}{t} \end{aligned} \tag{7.47}$$

Therefore, the integral is given by:

$$\begin{aligned} p &= c(p_{f(q_2)} - p_{f(q_1)})^3 \\ &\approx 2.42 \times 10^{-2} + \frac{0.203}{t} + \frac{0.523}{t^2} + \frac{0.278}{t^3} - \frac{0.323}{t^4} + \frac{7.748 \times 10^{-2}}{t^5} - \frac{5.709 \times 10^{-3}}{t^6}. \end{aligned} \tag{7.48}$$

As shown in Figure 7.13, we again have a decreasing function that has no interior

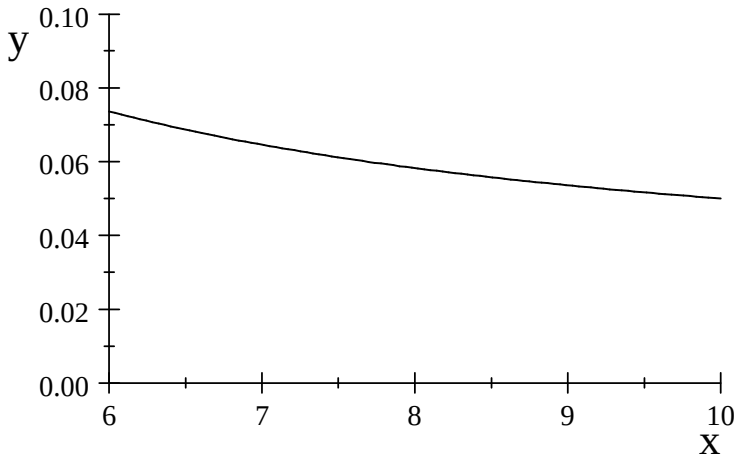


Figure 7.13. Graph of p, $6 \leq t \leq 10$.

maximums. Evaluating p at the end points gives $p(6) \approx 0.0736$ and $p(10) \approx 0.05$. Therefore, we keep our current maximum.

Max Count and Time: From the above, it follows that MaxCount has an approximate value of 3 at time $t = 0.\overline{4}$.

7.3. Dynamic MaxCount Algorithm

MaxCount$(H, Q_1, Q_2, t^{[}, t^{]})$
input: A set of buckets H built by the index structure presented, query points $Q_1(t)$ and $Q_2(t)$ and a query time interval $(t^{[}, t^{]})$.
output: The estimated MaxCount value.

```
01.  TimeIntervals ← ∅                                                    O(1)
02.  for i ← 0 to |H| − 1                                                 O(B)
03.      CrossTimes ← CalculateCrossTimes(Q1, Q2, t[, t], Hi)             O(1)
04.      for j ← 1 to |CrossTimes| − 1                                    O(1)
05.          Union(TimeIntervals, TimeInterval(tj−1, tj)                  O(1)
06.      end for
07.  end for

08.  TimeIntervals = BucketSort(TimeIntervals)                            O(B)
09.  IndexTimeIntervals = Merge(TimeIntervals)                            O(B)
10.  for each IndexTimeInterval ∈ IndexTimeIntervals                      O(B)
11.      calculate(MaxCount, MaxTime, IndexTimeInterval)                  O(1)
12.  end for

13.  return (MaxCount, MaxTime)
```

The algorithm to compute MaxCount with each line labeled with its running time is given above. Line 01 initiates a set of bucket time-interval objects to be empty. Line 03 returns a list of ordered times when a line through Q_1 or Q_2 crosses a bucket corner vertex. Line 05 turns this list into a set of $TimeInterval$ objects and adds them to the set of $TimeIntervals$. We list this "for each" loop as $O(1)$ because it consists of a constant number of calculations bounded by the number of vertices in the bucket. Line 08 uses the linear time sorting algorithm BucketSort to sort the bucket time intervals. Line 09 creates the time-partition order and index bucket

time intervals from the bucket time intervals in $O(B)$. An additional pass adds the bucket time intervals to the appropriate index time-intervals in $O(B)$. Lines 10-12 perform the MaxCount calculation discussed above.

In order to use the linear time BucketSort algorithm, we need the following definition and lemmas.

Definition 7.7 (Time-Interval Ordering)**.** *We define the lexicographical ordering* $\prec$ *of two* time intervals A *and* B *as follows:*

$$A.l < B.l \quad \Rightarrow \quad A \prec B \tag{7.49}$$

$$A.l = B.l \quad \wedge \quad A.u < B.u \quad \Rightarrow \quad A \prec B \tag{7.50}$$

$$A.l = B.l \quad \wedge \quad A.u = B.u \quad \Rightarrow \quad A = B \tag{7.51}$$

The distribution of time interval objects created in Line 08 of the MaxCount algorithm may not be uniform across the query time interval $T = [t^{\lfloor}, t^{\rfloor}]$. However, we can still prove the following.

Lemma 7.3. *If the distribution of buckets is uniform, then the distribution of bucket time-interval objects can be uniformly distributed within the sorting buckets of the bucket sort.*

Proof. Consider the relationship between successive slopes measured as the angles between lines through a query point Q with slopes $s_i = -t_i$ and $s_{i+1} = -t_{i+1}$. Suppose $\triangle t = 1$ with $t_0 = 0$ and $t_1 = 1$, then the angle between the two lines is $\triangle s = \frac{\pi}{4}$. The solid lines in Figure 7.14 show that half of the bucket corner vertices

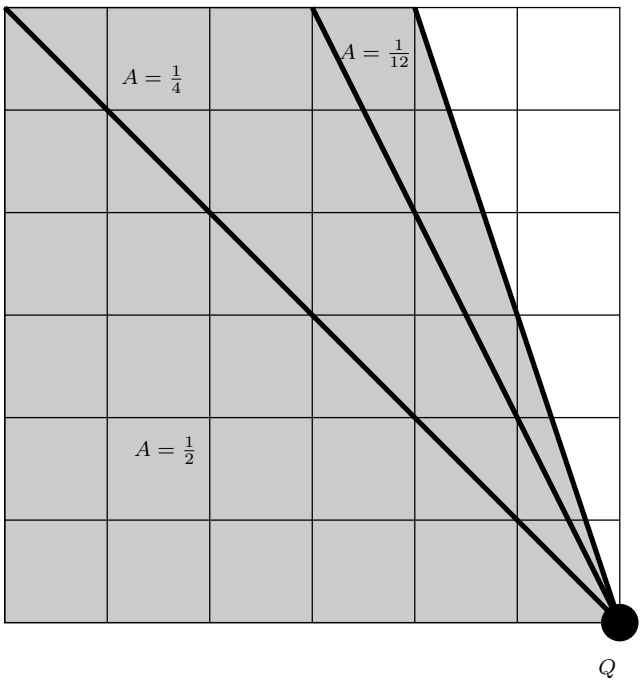


Figure 7.14. Areas of successive slopes.

are swept by the line sweeping through Q between $s_0 = 0$ and $s_1 = -1$. Consider a query time interval $[0, 10]$. Half of the corner vertices, and thus half of the time intervals, are between time $t = 0$ and $t = 1$. Thus, we conclude that the time interval objects created by sweeping will not be uniformly distributed throughout the query time interval.

Let Q' be the midpoint between Q_1 and Q_2. Let $S = \{t_1, ...t_k\}$ where $t_1 = t^{\lfloor}$, $t_k = t^{\rfloor}$ and $t_{i+1} - t_i = L$ for some positive constant L and $1 \leq i \leq k-1$. Let D_B be a bucket that contains the space in the 6-dimensional index. Model the normalized bucket function for D_B as a constant $F = 1$. Thus p, the bucket probability, from Equation (7.32) becomes the hyper-volume of the space swept by the line through Q'. By Lemma 7.2, we can find the area for a specific time interval in S in

constant time. The percentage of sorting buckets, $posb_i$, needed in any time interval $T_i = [t_i, t_{i+1}] \in S$ within the query time interval is given by:

$$posb_i = \frac{p(t_{i+1}) - p(t_i)}{p(t^]) - p(t^[)} \tag{7.52}$$

Let N be the number of sorting buckets. Then, the number of sorting buckets, $nosb_i$, assigned to interval i is given by:

$$nosb_i = N \cdot posb_i \tag{7.53}$$

If $nosb_i < 1$ we can combine it with $nosb_{i+1}$. If the query time interval is very large, then we may need to include multiple time intervals from S to get one sorting bucket. Thus, we create more sorting buckets (with smaller time intervals) in areas where the expected number of bucket time intervals is large. Conversely, we create fewer sorting buckets (with larger time intervals) in areas where the expected number of bucket time intervals is small. Hence we model each sorting bucket so that its time interval length directly relates to the percentage of bucket time intervals that are assigned to it. Thus, we conclude that we will uniformly distribute the time interval objects across all sorting buckets. □

Lemma 7.4. *Insertion of any bucket time-interval object T_O into the proper sorting bucket can be done in $O(1)$ time.*

Proof. The distribution of sorting buckets is determined by k time intervals in Lemma 7.3. Call these *sorting time interval objects* where each object contains: the lower bound l, the upper bound u, the number of sorting buckets assigned to this

interval b_s, the length of the time interval for the sorting bucket w and an array B_p containing pointers to these sorting buckets. Let A be the array of sorting time interval objects, and L be the length of each time interval where the time intervals are as in Lemma 7.3. Then, finding the correct sorting bucket for T_O requires two calculations:

$$SortingTimeInterval = A\left[\left\lfloor \frac{T_O.l}{L} \right\rfloor\right] \tag{7.54}$$

$$SortingBucket = B_p\left[\left\lfloor \frac{T_O.l - SortingTimeInterval.l}{w} \right\rfloor\right]. \tag{7.55}$$

Each of these calculations requires constant time, hence T_O can be inserted into the proper sorting bucket in $O(1)$ time. □

Using the above two lemmas, we can prove the following.

Theorem 7.2. *The running time of the* MaxCount *algorithm is* $O(B)$ *where* B *is the number of buckets.*

Proof. Let H be the set of buckets where each bucket B_i contains the normalized trend function F_i. Let Q_1 and Q_2 be the query points and $[t^{[}, t^{]}]$ be the query time interval. (Lines 01-07): Calculating the time intervals takes $O(B)$ time because the cross times for each bucket can be calculated in constant time. (Line 08): By Lemmas 7.3 and 7.4, we have an approximately even distribution of time interval objects within the sorting buckets where we can insert an object in constant time. This result fulfills the requirements of the BucketSort, Cormen et al. (2001), which allows the intervals to be sorted in $O(B)$ time. (Lines 09-12): Calculate the MaxCount and time for each time interval in constant time using Lemma 7.2.

These lines takes $O(B)$ time because there are $O(B)$ time intervals. Finding the global MaxCount and time requires retaining the maximum time and count at line 11. Returning the MaxCount and time takes $O(1)$ time. Thus, the running time is given by $O(B) + O(B) + O(B) + O(1) = O(B)$. □

7.4. An Exact MaxCount Algorithm

ExactMaxCount$(D, Q_1, Q_2, t^{[}, t^{]})$
input: D is the database of points. The query is made up of a hyper-rectangle Q defined by points Q_1 and Q_2 and the time interval $T = [t^{[}, t^{]}]$
output: The exact MaxCount and time at which it occurs.

```
01.  Times ← ∅ //of CrossTime objects                                   O(1)
02.  for each point p_i ∈ D                                             O(N)
03.      if p_i ∈ Q during T                                            O(1)
04.          EntryTime ← CalculateEntryTime(p_i, Q, T)                  O(1)
05.          ExitTime ← CalculateExitTime(p_i, Q, T)                    O(1)
06.          if EntryTime ∈ Times                                       O(1)
07.              Times.GET(EntryTime).Count++                           O(1)
08.          else
09.              Times.ADD(newCrossTime(EntryTime))                     O(1)
10.          end if
11.          if ExitTime ∈ Times                                        O(1)
12.              Times.GET(ExitTime).Count--                            O(1)
13.          else
14.              Times.ADD(newCrossTime(ExitTime))                      O(1)
15.          end if
16.  end for
17.  SORT(Times)                                                        O(n log n)
18.  TRAVERSE(Times, time, Max-Count) //tracking time                   O(N)
                                       //and MaxCount
19.  return (time,MaxCount)                                            O(1)
```

The above algorithm finds the exact MaxCount values. It is easy to see that the running time is given by:

$$O(N) + O(n \log n) \tag{7.56}$$

where N is the number of points in the database and n represents the result size of the query.

It is possible to slightly improve the above algorithm. First, divide the index space into k subspaces and maintain separate partial databases for each. Assign processes on individual systems to each database to calculate the MaxCount query and return the time intervals to a central process. Merging the time interval lists into a global time interval list saves time on the sorting part of the algorithm. The running time for each of k partial databases would be close to $O(\frac{n}{k} \log \frac{n}{k})$. This result is an approximate value because we do not guarantee an even split between partial databases. Placing buckets for each partial database in a Tree structure may be reasonable and could cut down the average running time to $O(\log N + n \log n/k)$. Implementation and analysis for this particular approach is left as future work.

CHAPTER 8

Threshold Aggregation Operators

The ThresholdRange algorithm shown below and described in Definition 5.2 relates to MaxCount in the way we calculate the aggregation. We maintain a running count to find time intervals that exceed the threshold value M. If we set the threshold value near the MaxCount value ($M \rightarrow$ MaxCount), ThresholdRange finds a small interval containing the MaxCount. We demonstrate this in the results, Chapter 10.

ThresholdRange$(H, Q_1, Q_2, t^{\lfloor}, t^{\rfloor}, M)$
input: A set of buckets H build by the index structure presented, query points $Q_1(t)$ and $Q_2(t)$, a query time interval $[t^{\lfloor}, t^{\rfloor}]$, and M is the threshold value
output: The estimated set of time intervals where R contains more than M points.

01 - 08 are the same as the MaxCount algorithm.

09.	$TimeIntervals \leftarrow \emptyset$	$O(1)$
10.	**for each** $TimeInterval \in TimePartitionOrder$	$O(B)$
11.	$CMaxCount \leftarrow$ calculate(MaxCount, $MaxTime, TimeInterval$)	$O(1)$
12.	**if** $CMaxCount > M$	$O(1)$
13.	$TimeIntervals \leftarrow TimeIntervals \bigcup TimeInterval$	$O(1)$
14.	**end if**	
15.	**end for**	
16.	Merge($TimeIntervals$)	$O(B)$
17.	**return** $TimeIntervals$	

ThresholdRange uses the same techniques up to Line 08, and then collects different information from each $TimeInterval$ starting in Line 10. This leads to the following Theorem.

Theorem 8.3. *The estimated* ThresholdRange *query runs in* $O(B)$ *time.*

Proof. The ThresholdRange algorithm differs from the MaxCount algorithm only in lines 09-17. Lines 11-14 run in $O(1)$ time. Line 10 executes lines 11-13 $O(B)$ times. In line 16, Merge($TimeIntervals$) is a linear walk of the time intervals that joins adjacent time intervals T_a and T_b when $T_a \bigcup T_b$ would form a continuous time interval. The calculation is trivially $O(1)$ time for joining the adjacent intervals. Hence, we conclude by Theorem 7.2 that the ThresholdRange runs in $O(B)$ time. □

We give the following three operators based on ThresholdRange and conclude that none of the changes to the algorithm affect the running time of the ThresholdRange algorithm.

ThresholdCount:

By adding a line between 14 and 15 in the ThresholdRange algorithm that counts the merged time intervals, we can return the count of time intervals during the query time interval where congestion occurs. This count of time intervals gives a measure of variation in congestion. That is, if we have lots of time intervals, we expect that we have a large number of pockets of congestion. Since ThresholdCount does not give information relative to the entire time interval, it may need to be examined in light of the total time above the threshold.

ThresholdSum:

By summing the times instead of using the $\bigcup$ operator in line 13 of the ThresholdRange algorithm, we can return the total congestion time during the query time interval. This total gives a measure of the severity of congestion that may be compared to the length of query time.

ThresholdAverage:

By adding a line between lines 14 and 15 in the ThresholdRange algorithm that finds average length of the merged time intervals, we can return the average length of time each congestion will last. This average gives a different measure of the severity of each congestion.

We could calculate other operators such as the standard deviation of the time intervals or many other complicated statistics on the distribution of time intervals. However the five operators we define mirror the standard aggregation operators available in relational databases.

CHAPTER 9

CountRange

The CountRange algorithm is an adaptation of MaxCount in that it is the Count portion of the MaxCount query. Using the equations for the cases described in Figure 7.8, we calculate the CountRange as follows:

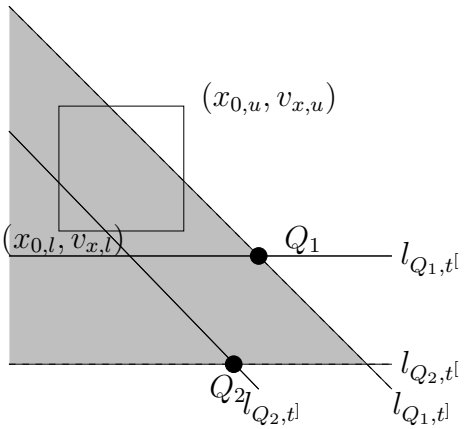


Figure 9.1. CountRange Q_1 at $t^]$ to Q_2 at $t^[$.

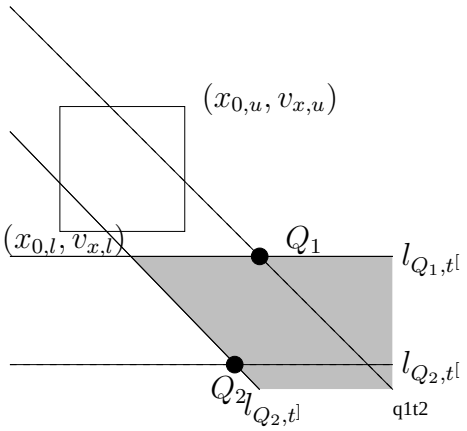


Figure 9.2. CountRange Q_1 at $t^[$ to Q_2 at $t^]$.

For each bucket we determine if the bucket is completely in or completely out of the query space. First we find the beginning and ending time intervals. For each time interval, we get the associated function $\triangle p$ given in Equation (7.20) and its components. The components $\triangle p$ given in Equation (7.19) define the area above a line through Q_1 and Q_2 at times $t^[$ and $t^]$. Figures 9.1 and 9.2 show these four lines.

Figure 9.1 shows the shaded area defined by:

$$\triangle \overleftarrow{p} = p_{Q_2,t[} - p_{Q_1,t]}. \tag{9.1}$$

Figure 9.2 shows the shaded area:

$$\triangle \overrightarrow{p} = p_{Q_2,t]} - p_{Q_1,t[}. \tag{9.2}$$

If $\triangle \overleftarrow{p}$ or $\triangle \overrightarrow{p}$ for bucket i is equal to the count of the bucket, then bucket i is completely contained in the query. If $\triangle \overleftarrow{p}$ and $\triangle \overrightarrow{p}$ for bucket i are equal to 0, then bucket i is not contained in the query. If neither of these is true, we approximate the count for bucket i as the $\max(\triangle \overleftarrow{p}, \triangle \overrightarrow{p})$. That is, we calculate the number of points in bucket i that contribute to the CountRange as:

$$count_i = \begin{cases} b_i & \text{if} \quad \triangle \overleftarrow{p} = b_i \vee \triangle \overrightarrow{p} = b_i \\ 0 & \text{if} \quad \triangle \overleftarrow{p} = \triangle \overrightarrow{p} = 0 \\ \max(\triangle \overleftarrow{p}, \triangle \overrightarrow{p}) & \text{Otherwise} \end{cases} \tag{9.3}$$

This calculation requires that we keep the single dimension equations for Q_1 and Q_2 available and not discard them after finding $\triangle p$ (see Equation (7.20)).

Hence, we have the following algorithm for CountRange:

CountRange$(H, Q_1, Q_2, t^{[}, t^{]})$
input: A set of buckets H built by the index structure presented, query points $Q_1(t)$ and $Q_2(t)$ and a query time interval $(t^{[}, t^{]})$.
output: the estimated CountRange.

1.	$Count \leftarrow 0$	$O(1)$
2.	**for each** *bucket* $B_i \in D$	$O(B)$
3.	Calculate$(\triangle \overleftarrow{p}, \triangle \overrightarrow{p})$ //using Equations (9.1)-(9.2)	$O(1)$
4.	Calculate$(count_i)$ //using Equation (9.3)	$O(1)$
5.	$Count \leftarrow Count + count_i$	$O(1)$
6.	**end for**	
7.	**return** $Count$	$O(1)$

Theorem 9.4. *The* CountRange *query runs in* $O(B)$ *time.*

Proof. Consider two different data structures for our buckets: HashTables and R-trees. In the case of indexing using an R-tree, the worst case requires that we examine all buckets used in generating CountRange. It is possible that this list could include all B buckets giving a worst case of $O(B)$. In the case of using a HashTable, we must examine all B buckets. By Lemma 6.2, and because Equations (7.33) and (9.3) are calculated in constant time, each bucket can be examined to determine the count that contributes to the CountRange query in constant time. Therefore, the algorithm runs in $O(B)$ time. □

We note that CountRange is a simplification of the MaxCount operator in that we do not examine every time interval. Further we have a slightly different form of $\triangle p$ from Equation (7.20) to find the count.

CHAPTER 10

Experimental Results

10.1. Methods and Measures

We collected data in runs of several queries that were selected from a set of randomly generated queries. The selection process weeded out most similar queries and kept a set that represents narrow queries, wide queries, near corner or edge queries, and outside queries. Throughout our experiments, we did not see significant accuracy fluctuation due to any of these types of queries.

Each experimental run consists of running all of the queries at several different decreasing bucket sizes on a single data set. We made experimental runs against data sets ranging from 10,000 points to 1,500,000 points[1].

In the following experimental analysis, we measure the percentage error of the estimation algorithm relative to the exact-count algorithm as follows:

$$Error_{Relative} = \frac{|Exact\ Operator - Estimated\ Operator|}{Exact\ Operator} \tag{10.1}$$

Equation (10.1) provides a useful measure if the query returns a reasonable number of points. Queries that return a small number of points indicate that we should use the exact method.

[1]Threshold aggregation runs go only to 1 million points at which we already achieve acceptable error.

For ThresholdRange, we measure the percentage of intervals given by the accurate algorithm not covered by the estimation algorithm using the operator UC for uncovered. That is, $\mathrm{UC}(a, b)$ returns the sum of the lengths of intervals in a not covered by intervals in b. We divide the result by the accurate ThresholdSum to determine the ThresholdRange error:

$$\texttt{error} = \frac{\mathrm{UC}\left(\textit{Ext.}\ \textsc{ThresholdRange}, \textit{Est.}\ \textsc{ThresholdRange}\right)}{\textit{Ext.}\ \textsc{ThresholdSum}} \tag{10.2}$$

We also measure the percentage of intervals given by the estimate algorithm not covered by the exact algorithm. We divide the result by the estimated ThresholdSum to determine the ThresholdRange excess-error.

$$\texttt{excess-error} = \frac{\mathrm{UC}\left(\textit{Est.}\ \textsc{ThresholdRange} \backslash \textit{Ext.}\ \textsc{ThresholdRange}\right)}{\textit{Est.}\ \textsc{ThresholdSum}} \tag{10.3}$$

We performed all the data runs on a Athlon 2000 with 1 GB of RAM. During each of the queries the program does not contact the server tier and, thus, minimizes the impact of running a server on the same computer. The program pre-loads all data into data structures so that even the exact algorithms do not contact the server tier.

10.2. Data Generation

Data for the experiments was randomly generated around several cluster centers. The i^{th} point generated for the database is located near a randomly selected cluster at a distance between 0 and d, where d is proportional to i. This method is similar to

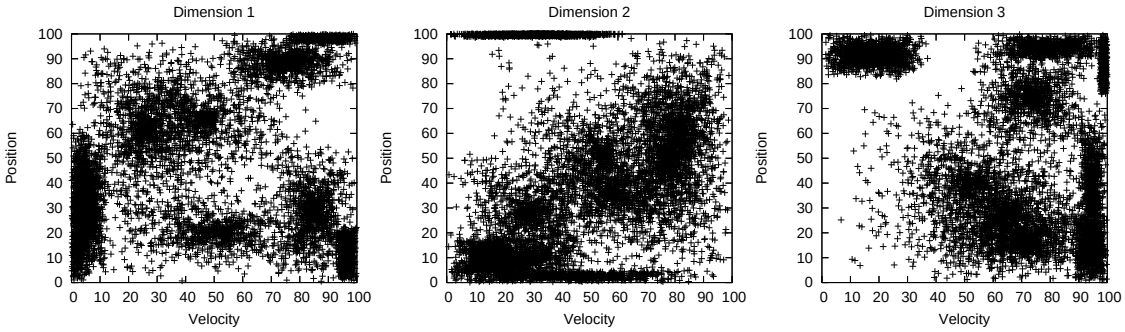


Figure 10.1. Sample data view.

the Ziggurat (Marsaglia & Tsang 2000) method of generating gaussian (or normal) distributions used in the GSTD (Theodoridis et al. 1999) and G-TERD (Tzouramanis et al. 2002) spatiotemporal data generators (Nascimento et al. 2003). However, our method does not generate strictly Gaussian distributions since the distributions may stretch and compress along an axis. Our goal was to generate a cluster that represents a source location and velocity that has most elements starting near a center point and decreasing as one moves to a boundary for the cluster. This method models source regions where the objects all head about the same direction. A secondary goal was to make certain that clusters were random in size and shape. The program is also capable of approximating a Zipf distribution used in (Choi & Chung 2002, Revesz & Chen 2003, Tao, Sun & Papadias 2003). However, a single Zipf distribution does not test the adaptability of our algorithm well. I.e. our algorithm is capable of modeling a Zipf distribution and as such we could use a single bucket. Figure 10.1 shows a sample of a data set with points projected onto the three views. The clusters look even more random, because they can overlay one another. When one looks at these, they nearly resemble the lights of a city from the air.

Along with a single Zipf distribution, we also note that a randomly generated uniform-distribution is not a good distribution to use for these types of experiments. Uniform distributions do not test the ability of the algorithm to adapt. In fact from early experiments in (Anderson & Revesz 2005) we have found that using such a distribution gives great (though meaningless) results. The problem resolves to a system capable (and willing to) model a uniform distribution finding a nearly perfect uniform distribution to model. Hence these results are neither realistic, nor meaningful.

10.3. Parameter Effects

The index space ranges from 0 to 100 in each dimension. The **number of points** in the different data sets ranges from $10,000$ to $1,500,000$. The following parameters were used in creating the index and finding the MaxCount.

Size of Buckets: The size of the buckets determines the number of possible buckets in the index. In the experiments, buckets divide the space up such that there are 5 to 20 divisions in each dimension[2]. These divisions equate to bucket sizes ranging from 5 to 20 units wide in each dimension. Relative to our previous work (Anderson 2006), this algorithm puts much more space into each bucket creating bigger buckets.

Query Location: Locating the query near the lower or upper corners affects relative accuracy because the query returns very few points. Queries in this region

[2]Some MaxCount runs included up to 40 divisions increasing accuracy, but not enough to warrant the extra running time.

are not interesting because they rarely involve many points and represent a query region that moves away from points in the database or barely moves at all. The small number of points returned indicates use of the exact algorithms.

Query Types: In (Anderson 2006), we considered queries with several different characteristics: dense, sparse, and Euclidean distance as it related to bucket size. By modeling the skew in buckets, we minimize the effect of these characteristics to the point that they did not impact the query error. Queries where the distance between the query points was small appeared to do as well as wider queries *providing they returned a reasonable number of points.* This result is a clear improvement over previous work that assumed uniform density within a bucket.

Cluster Points: Index space saturation determines the number of buckets necessary for the index. The number of cluster points does not appear to affect error as much as the space saturation. Further, we do not consider a larger number of cluster points reasonable since the index space approaches a uniform distribution as the number of cluster points increases. Gaps introduce difficult areas to model when they are not uniform. And once again we reiterate, uniform distributions are not useful. In our experiments cluster points number between 10 and 50.

Histogram Divisions: Increasing histogram divisions to $s > 5$ had no affect on the accuracy. This result is not unexpected because histograms are used to define a trend function relative to trend functions on other axes. Increasing the histogram divisions has a tendency to flatten the lines. However, normalization flattens the trend function while maintaining the relationships between trends and hence this

behavior is easily explained. Thus, increasing histogram divisions only increases the running time without increasing accuracy.

Threshold Value: The threshold value determines the accuracy when set to low values compared to the number of points in the database. As expected, these extreme point values produce accurate estimations. High values also follow this trend.

Time Endpoints: When dealing with either small time end points or small buckets, the method is susceptible to rounding error. In particular, Equation (7.33) contains both t^6 and $\frac{1}{t^6}$ terms. For very small values, on the order of 1×10^{-54} for 64-bit doubles, these calculations are extremely sensitive and care must be given to guard against rounding error. Those errors showed in two ways. First, by a direct warning programmed into the solution, and second, by a series of fairly stable time values for the MaxCount followed by unstable variations when increasing the number of buckets. At some point, smaller bucket sizes increase the likelihood of errors in both time and count values. Also smaller buckets contain fewer points, which impacts the size of the constants in Equation (7.33). Hence, as the bucket size becomes smaller in successive runs, the existence of instability in the time values after a series of stable values predicts that an accurate MaxCount may be found in the previous larger bucket size. *Throughout our experiments, this condition was an excellent predictor of an accurate* MaxCount.

The experiments demonstrated that 6-dimensional space compounds the problem when creating small buckets. Creating an index with unit buckets would result in the possibility of having 1×10^{12} buckets. Clearly this number is unrealistic for

common moving object applications where we may be dealing with million(s) of objects. In practice the number of buckets needed to reach acceptable error levels was between $78,000$ and $227,000$ buckets. These numbers reflect the ability to reach error levels under 5% and were roughly related to the saturation of the space by the points. It should be clear that a higher saturation of the space by points would require a larger number of buckets. Figure 10.2 shows that we had a roughly linear increase in the number of buckets for an exponential increase in the space. This pleasant surprise indicates that for unsaturated data sets, the exponential explosion of space is manageable.

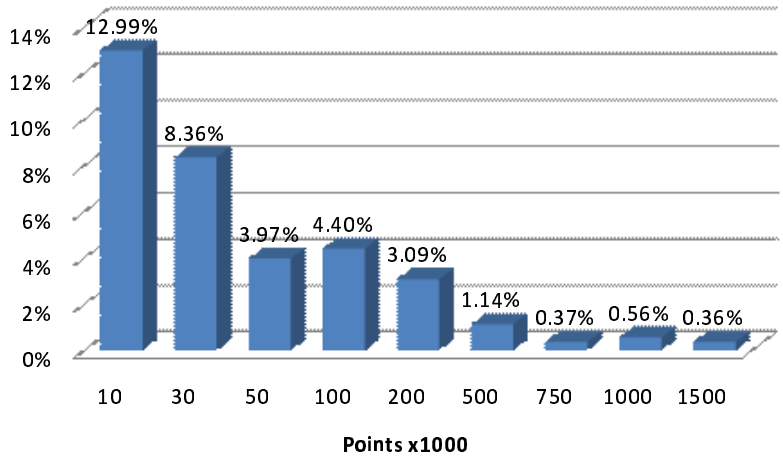


Figure 10.2. Ratio of buckets to points.

10.4. Running Time Observations

Figure 10.3 shows the average ratio of the exact MaxCount running time to the estimated MaxCount running time as a function of the number of points in the database. This result shows a nearly exponential growth when comparing the

values between 10,000 and 1,000,000. The leveling off occurs because the number of points returned by the query of 1 million points nearly equals the number of points returned by the query of 1.5 million points. This result precisely matches our running-time analysis of the exact and estimation algorithms.

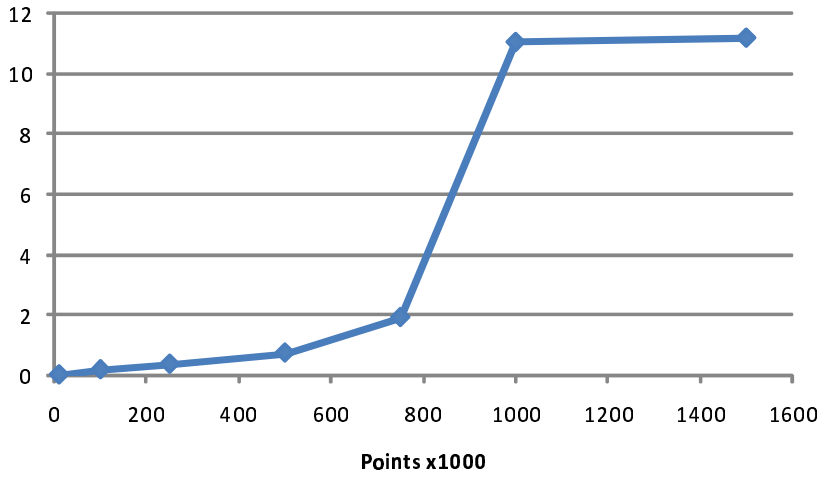


Figure 10.3. Ratio of exact running time to estimated running time.

A natural question is when to use the exact versus the estimated methods. In runs with a small number of points that need to be processed, the exact and estimation methods run about equally fast. However, when the result size reaches values greater than $40,000$ (our experiments returned sets as large as 331,491), the estimation algorithms run up to 35 times faster than the exact algorithms. Further, we note that the error is less predictable at smaller results sizes. Hence for small databases or in queries that return small result sets, efficiency and accuracy both indicate using the exact method. However, for large data sets greater than or equal to 1 million points, the estimation method out-performs by far the exact method.

10.5. Operator Observations

As expected, we noticed that each operator runs in about the same time as MaxCount. Only error values seemed to be different when studying different types of aggregation (e.g., when studying overlap error in ThresholdRange versus count error in MaxCount). Never-the-less, we have similarities between the results. Almost all the figures in this section look like a view of mountains from a valley. That is what we expected to see and the lower and flatter the terrain the better. Buckets increase from back to front and point set sizes increase from left to right.

10.5.1. MaxCount

Figure 10.4 shows that increasing the number of buckets to the indicated values dramatically decreases the MaxCount error. As the number of points increases we also see a decrease in the error. Note that for larger buckets (e.g. smaller values on the "Buckets per Dimension axis"), the error decreases at a slightly faster rate.

The exact MaxCount provided the values against which our estimation algorithm was tested for accuracy. Since the method does not rely on buckets, and has zero error, we note only that on queries with small result sizes, this method performs as well, or better than the estimation algorithm.

10.5.2. ThresholdRange

Figures 10.5 and 10.6 give the ThresholdRange error and ThresholdRange excess error respectively for $T = 10$. ThresholdRange error gives the percentage of the exact intervals not covered by the estimation value, and ThresholdRange

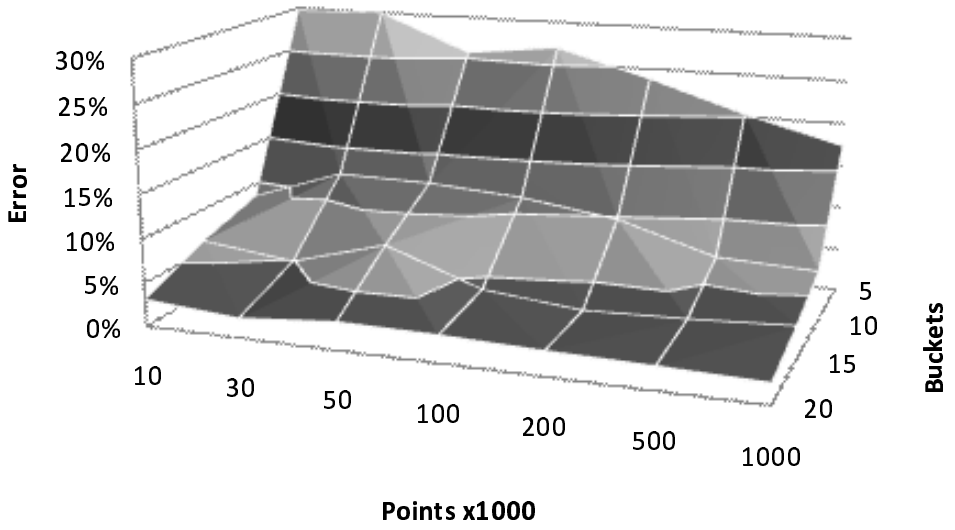


Figure 10.4. MaxCount error.

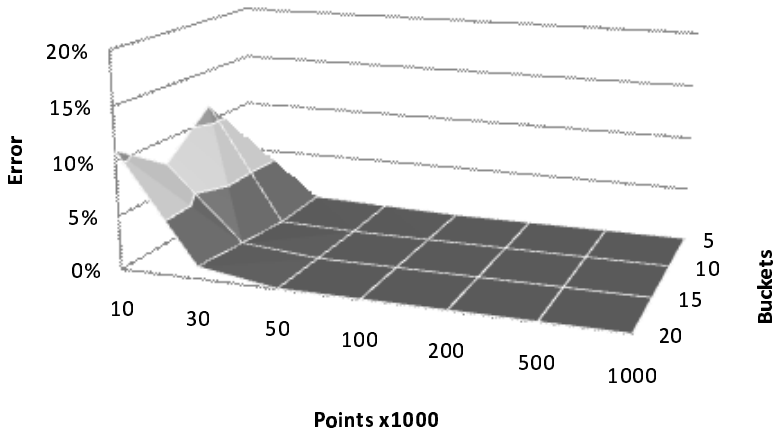


Figure 10.5. ThresholdRange error.

excess error gives the percentage of the estimation not covering the exact. These figures show that our method acts conservatively in covering more than is needed.

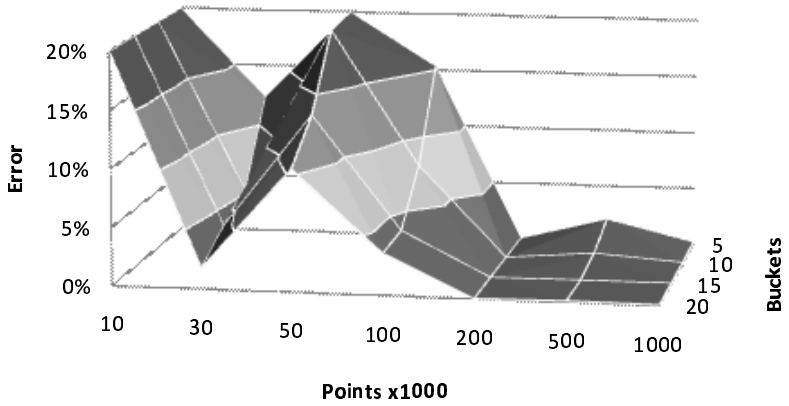


Figure 10.6. ThresholdRange error.

However, at larger point-set sizes, we still achieve under 5% error. Figure 10.5 shows 0% error caused by the point count staying above 10% in data sets containing more than 30,000 points. Figure 10.6 shows that we covered at least 10% more time in the query time interval than needed until we reach larger point sets. Still, we showed improvement with more buckets.

At $T = 1000$, we see 0% error until we reach point sets of 500,000 and greater. Figure 10.7 shows excellent results with buckets above 10. Also, Figure 10.8 shows that the excess error drops to near 0% as well.

Figures 10.9 and 10.10 show what happens when we find an interval near the MaxCount value. The two figures show the consequences of the estimation intervals being offset from the exact intervals by small amounts. The error decreases with more buckets.

The rest of our results are shown in Figures C.1 to C.6 in Appendix C.

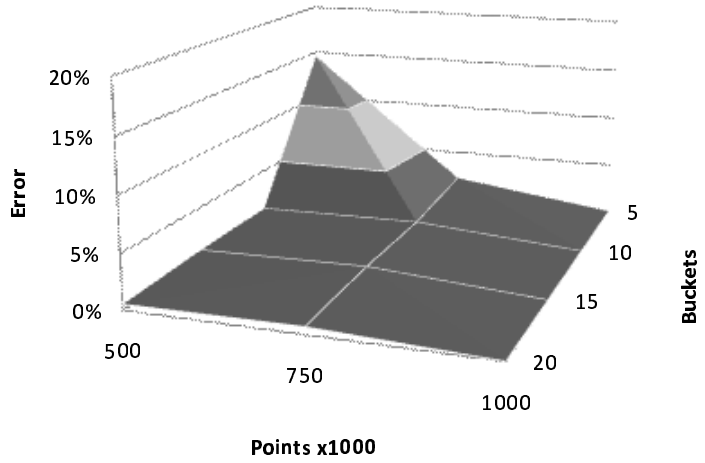


Figure 10.7. ThresholdRange error, T=1000.

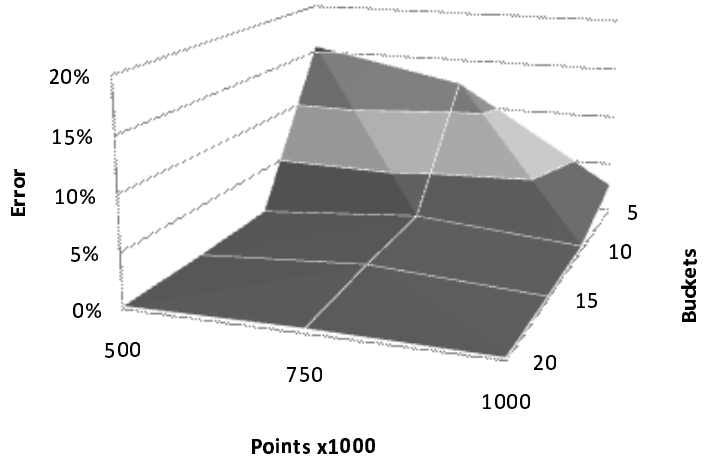


Figure 10.8. ThresholdRange excess error, T=1000.

10.5.3. ThresholdCount

This operator is the only operator that does not have relative error measurements. Instead we report the average number of intervals the estimation method differs from the exact method. As you can see, we differ by two from the correct number.

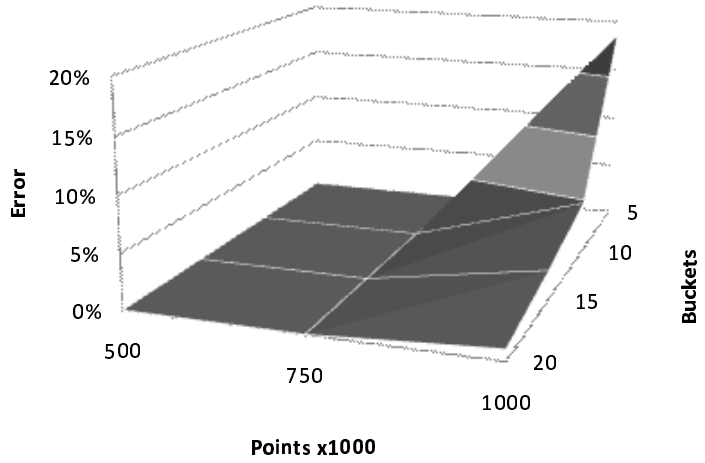


Figure 10.9. ThresholdRange error, T=100000.

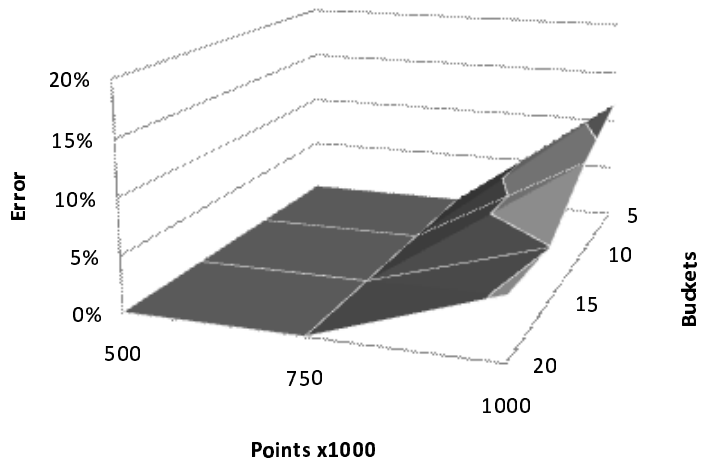


Figure 10.10. ThresholdRange excess error, T=100000.

Figure 10.11 shows the average error at $T = 10$ where the errors are small. Figure 10.12 ($T = 1000$) looks much worse, but in reality we are still below 2 intervals off. We also note that the estimation may split or combine an interval

incorrectly when the intervals are very close together without greatly affecting the error of other operators. Given this possibility, the results are excellent.

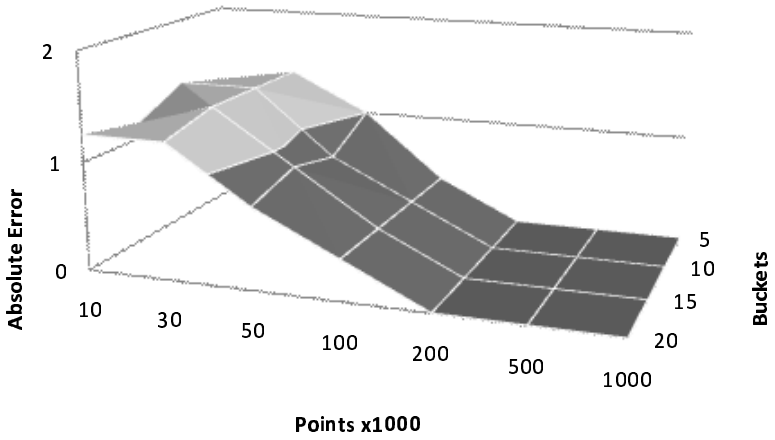


Figure 10.11. ThresholdCount error, T=10.

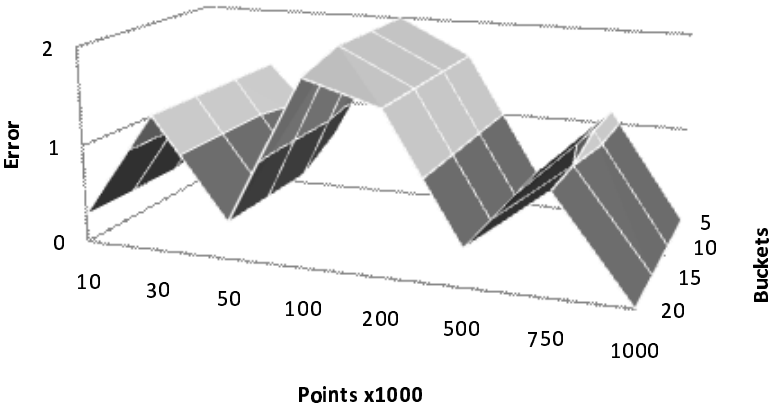


Figure 10.12. ThresholdCount error, T=100.

The rest of the results are shown in Figures C.7-C.10 in Appendix C.

10.5.4. ThresholdSum

ThresholdSum gives the total time above the threshold T. As one can see in Figure 10.13, at higher bucket counts we have excellent error rates at $T = 10$. We didn't always expect great results at this threshold level across all data sets, but ThresholdSum gives this result consistantly all the way across.

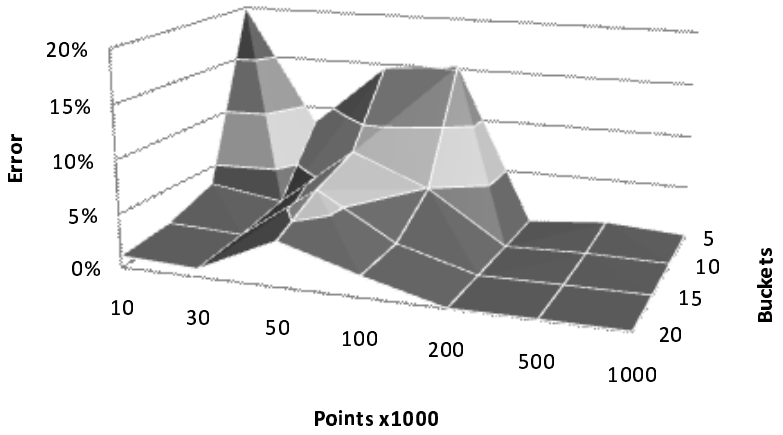


Figure 10.13. ThresholdSum error, T=10.

We do note that when the threshold approaches MaxCount, we see extremely good accuracy as shown in Figure 10.14.

The rest of the results are shown in Figures C.11-C.14 in Appendix C.

10.5.5. ThresholdAverage

ThresholdAverage gives the average length of each time interval. Figure 10.15 shows the now familiar mountains descending below 5% error at 20 buckets for $T = 10$. The Figure also shows that even though a few of the data sets tended to

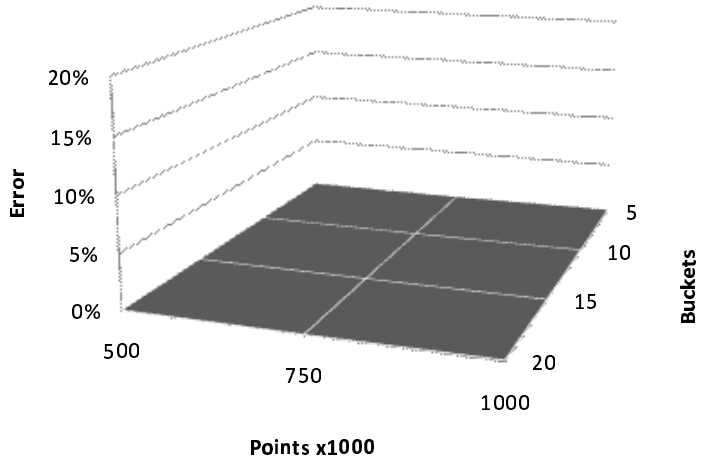


Figure 10.14. ThresholdSum error, T=100000.

have good results at 5 and 10 buckets, these results are not guaranteed in general. In Figure 10.16, the error reaches a plateau below 5% with only small bumps in the data. The rest of the threshold values are shown in Figures C.15-C.18 of Appendix C.

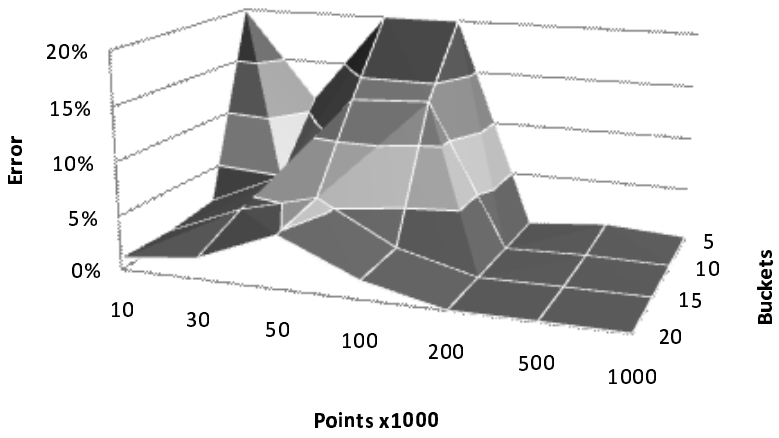


Figure 10.15. ThresholdAverage error, T=10.

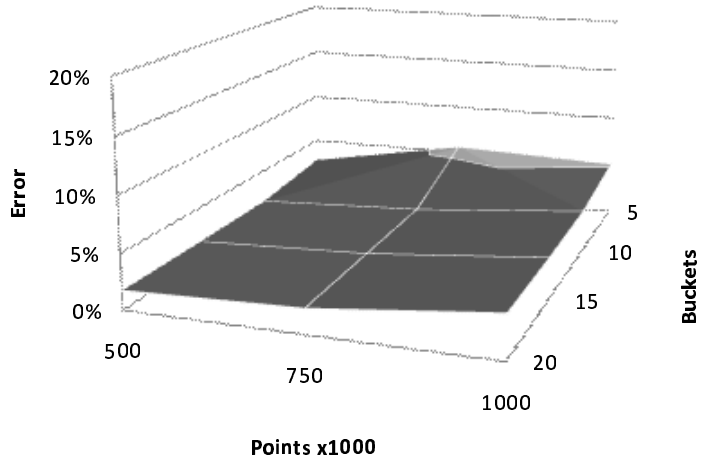


Figure 10.16. ThresholdAverage error, T=1000.

10.5.6. CountRange

Other CountRange algorithms have achieved error values between 2% and 3%. Using our method we conjecture that we could reduce the error because our method of approximation, although much more complicated, theoretically adapts to skewed distributions better than other methods. Figure 10.17 shows that we achieved errors under 2% for 20 buckets across all the data sets, and in some cases, under 1%.

Count range also performs about the same speed as the threshold operators due to its similar implementation.

10.6. Conclusions and Future Work

We implemented and compared two new MaxCount algorithms. The estimated MaxCount was shown to be fast and accurate while still allowing fast constant time updates. No other algorithm has these features to date. We showed

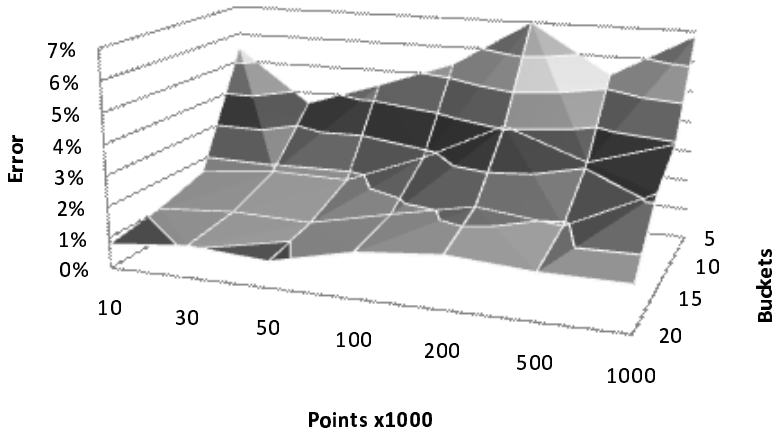


Figure 10.17. CountRange error.

that ThresholdRange, ThresholdCount, ThresholdSum, ThresholdAverage, and CountRange are related to MaxCount and can be evaluated using similar techniques and that we achieve error values under 5% in these operations. We gave an empirical threshold for choosing between the exact and estimated algorithms. We discussed the issues related to higher dimensions and note that all sweeping algorithms have this problem. We also note that using our technique it is possible to decompose the problem and run it in a multiprocessor or grid environment where the database is divided into smaller databases.

Future work may include decreasing the running time by finding other techniques because there does not appear to be a clear method for decreasing the running time of sweeping methods. One could also consider implementing and comparing these techniques in a grid computing environment.

APPENDIX A

Integral Details for Cases

For each case given in Section 7.2 we give the form for each case with no justification although the setup of each integral should be clear. Mathematica provided the evaluation of the integrals used in the implementation. In this section we give the details of each integral evaluation. Because of the size we present one per page.

Case A:

$$\int_{x1L}^{-(x2u-x2q)/t+x1q} \int_{-t(x1-x1q)+x2q}^{x2u} (b1+a1x1)(b2+a2x2)\; dx2\; dx1 =$$

$$\begin{aligned}
t^2 \quad & \left(\frac{a2\ b1\ x1L^3}{6} + \frac{a1\ a2\ x1L^4}{8} - \frac{a2\ b1\ x1L^2\ x1q}{2} - \frac{a1\ a2\ x1L^3\ x1q}{3} + \right. \\
& \left. \frac{a2\ b1\ x1L\ x1q^2}{2} + \frac{a1\ a2\ x1L^2\ x1q^2}{4} - \frac{a2\ b1\ x1q^3}{6} - \frac{a1\ a2\ x1q^4}{24} \right) + \\
& b1\ b2\ x1L\ x2q + \frac{a1\ b2\ x1L^2\ x2q}{2} - b1\ b2\ x1q\ x2q - \frac{a1\ b2\ x1q^2\ x2q}{2} + \\
& \frac{a2\ b1\ x1L\ x2q^2}{2} + \frac{a1\ a2\ x1L^2\ x2q^2}{4} - \frac{a2\ b1\ x1q\ x2q^2}{2} - \frac{a1\ a2\ x1q^2\ x2q^2}{4} + \\
t \quad & \left(\frac{-\left(b1\ b2\ x1L^2\right)}{2} - \frac{a1\ b2\ x1L^3}{3} + b1\ b2\ x1L\ x1q + \frac{a1\ b2\ x1L^2\ x1q}{2} - \frac{b1\ b2\ x1q^2}{2} \right. \\
& \left. - \frac{a1\ b2\ x1q^3}{6} - \frac{a2\ b1\ x1L^2\ x2q}{2} - \frac{a1\ a2\ x1L^3\ x2q}{3} + a2\ b1\ x1L\ x1q\ x2q + \right. \\
& \left. \frac{a1\ a2\ x1L^2\ x1q\ x2q}{2} - \frac{a2\ b1\ x1q^2\ x2q}{2} - \frac{a1\ a2\ x1q^3\ x2q}{6} \right) - \\
& b1\ b2\ x1L\ x2u - \frac{a1\ b2\ x1L^2\ x2u}{2} + b1\ b2\ x1q\ x2u + \frac{a1\ b2\ x1q^2\ x2u}{2} - \\
& \frac{a2\ b1\ x1L\ x2u^2}{2} - \frac{a1\ a2\ x1L^2\ x2u^2}{4} + \frac{a2\ b1\ x1q\ x2u^2}{2} + \frac{a1\ a2\ x1q^2\ x2u^2}{4} + \\
\frac{1}{t} \quad & \left(\frac{-\left(b1\ b2\ x2q^2\right)}{2} - \frac{a1\ b2\ x1q\ x2q^2}{2} - \frac{a2\ b1\ x2q^3}{6} - \frac{a1\ a2\ x1q\ x2q^3}{6} + \right. \\
& \left. b1\ b2\ x2q\ x2u + a1\ b2\ x1q\ x2q\ x2u - \frac{b1\ b2\ x2u^2}{2} - \frac{a1\ b2\ x1q\ x2u^2}{2} + \right. \\
& \left. \frac{a2\ b1\ x2q\ x2u^2}{2} + \frac{a1\ a2\ x1q\ x2q\ x2u^2}{2} - \frac{a2\ b1\ x2u^3}{3} - \frac{a1\ a2\ x1q\ x2u^3}{3} \right) + \\
\frac{1}{t^2} \quad & \left(\frac{-\left(a1\ b2\ x2q^3\right)}{6} - \frac{a1\ a2\ x2q^4}{24} + \frac{a1\ b2\ x2q^2\ x2u}{2} - \frac{a1\ b2\ x2q\ x2u^2}{2} + \right. \\
& \left. \frac{a1\ a2\ x2q^2\ x2u^2}{4} + \frac{a1\ b2\ x2u^3}{6} - \frac{a1\ a2\ x2q\ x2u^3}{3} + \frac{a1\ a2\ x2u^4}{8} \right)
\end{aligned}$$

Case B:

$$\int_{-t(x_1-x_{1,q})+x_{2,q}}^{x_{2,u}} \int_{x_{1,L}}^{-(x_{2,u}-x_{2,q})/t+x_{1q},} (a_1x_1+b_1)(a_2x_2+b_2)dx_2dx_1 =$$

$$\begin{aligned}
t^2 \quad & \left(\frac{a2\,b1\,x1q^3}{6} + \frac{a1\,a2\,x1q^4}{24} - \frac{a2\,b1\,x1q^2\,x1u}{2} + \frac{a2\,b1\,x1q\,x1u^2}{2} - + \right. \\
& \left. \frac{a1\,a2\,x1q^2\,x1u^2}{4} - \frac{a2\,b1\,x1u^3}{6} + \frac{a1\,a2\,x1q\,x1u^3}{3} - \frac{a1\,a2\,x1u^4}{8} \right) + \\
t \quad & \left(\frac{b1\,b2\,x1q^2}{2} + \frac{a1\,b2\,x1q^3}{6} - b1\,b2\,x1q\,x1u + \right. \\
& \frac{b1\,b2\,x1u^2}{2} - \frac{a1\,b2\,x1q\,x1u^2}{2} + \\
& \frac{a1\,b2\,x1u^3}{3} + \frac{a2\,b1\,x1q^2\,x2q}{2} + \frac{a1\,a2\,x1q^3\,x2q}{6} - a2\,b1\,x1q\,x1u\,x2q + \\
& \left. \frac{a2\,b1\,x1u^2\,x2q}{2} - \frac{a1\,a2\,x1q\,x1u^2\,x2q}{2} + \frac{a1\,a2\,x1u^3\,x2q}{3} \right) + \\
& b1\,b2\,x1q\,x2q + \frac{a1\,b2\,x1q^2\,x2q}{2} - b1\,b2\,x1u\,x2q - \frac{a1\,b2\,x1u^2\,x2q}{2} + \\
& \frac{a2\,b1\,x1q\,x2q^2}{2} + \frac{a1\,a2\,x1q^2\,x2q^2}{4} - \frac{a2\,b1\,x1u\,x2q^2}{2} - \frac{a1\,a2\,x1u^2\,x2q^2}{4} + \\
& - b1\,b2\,x1q\,x2u - \frac{a1\,b2\,x1q^2\,x2u}{2} + b1\,b2\,x1u\,x2u + \frac{a1\,b2\,x1u^2\,x2u}{2} - \\
& \frac{a2\,b1\,x1q\,x2u^2}{2} - \frac{a1\,a2\,x1q^2\,x2u^2}{4} + \frac{a2\,b1\,x1u\,x2u^2}{2} + \frac{a1\,a2\,x1u^2\,x2u^2}{4} + \\
\frac{1}{t} \quad & \left(b1\,b2\,x2q\ (x2q-x2u) + a1\,b2\,x1q\,x2q\ (x2q-x2u) + \frac{a2\,b1\,x2q^2\,(x2q-x2u)}{2} + \right. \\
& \frac{a1\,a2\,x1q\,x2q^2\,(x2q-x2u)}{2} - \frac{b1\,b2\,(x2q-x2u)^2}{2} - \frac{a1\,b2\,x1q\,(x2q-x2u)^2}{2} - \\
& \frac{a2\,b1\,x2q\,(x2q-x2u)^2}{2} - \frac{a1\,a2\,x1q\,x2q\,(x2q-x2u)^2}{2} + \frac{a2\,b1\,(x2q-x2u)^3}{6} + \\
& \frac{a1\,a2\,x1q\,(x2q-x2u)^3}{6} - b1\,b2\ (x2q-x2u)\ x2u - + \\
& \left. a1\,b2\,x1q\ (x2q-x2u)\ x2u - \frac{a2\,b1\,(x2q-x2u)\,x2u^2}{2} - \frac{a1\,a2\,x1q\,(x2q-x2u)\,x2u^2}{2} \right) \\
\frac{1}{t^2} \quad & \left(\frac{a1\,b2\,x2q\,(x2q-x2u)^2}{2} + \frac{a1\,a2\,x2q^2\,(x2q-x2u)^2}{4} - \frac{a1\,b2\,(x2q-x2u)^3}{3} - \right. \\
& \frac{a1\,a2\,x2q\,(x2q-x2u)^3}{3} + \frac{a1\,a2\,(x2q-x2u)^4}{8} - \\
& \left. \frac{a1\,b2\,(x2q-x2u)^2\,x2u}{2} - \frac{a1\,a2\,(x2q-x2u)^2\,x2u^2}{4} \right)
\end{aligned}$$

Case C:

$$\int_{x2L}^{x2u-(x2-x2q)/t+x1q}\int_{x1L}^{} (a_1x_1+b_1)(a_2x_2+b_2)dx_1dx_2 =$$

$$b1\ b2\ x1L\ x2L + \frac{a1\ b2\ x1L^2\ x2L}{2} - b1\ b2\ x1\ qx2L - \frac{a1\ b2\ x1q^2\ x2L}{2} +$$
$$\frac{a2\ b1\ x1L\ x2L^2}{2} + \frac{a1\ a2\ x1L^2\ x2L^2}{4} - \frac{a2\ b1\ x1qx2L^2}{2} - \frac{a1\ a2\ x1q^2\ x2L^2}{4} -$$
$$b1\ b2\ x1L\ x2u - \frac{a1\ b2\ x1L^2\ x2u}{2} + b1\ b2\ x1q\ x2u + \frac{a1\ b2x1q^2\ x2u}{2} -$$
$$\frac{a2\ b1\ x1L\ x2u^2}{2} - \frac{a1\ a2\ x1L^2\ x2u^2}{4} + \frac{a2\ b1\ x1q\ x2u^2}{2} + \frac{a1\ a2\ x1q^2\ x2u^2}{4} +$$
$$\frac{1}{t}\left(\frac{b1\ b2\ x2L^2}{2} + \frac{a1\ b2\ x1q\ x2L^2}{2} + \frac{a2\ b1\ x2L^3}{3} + \frac{a1\ a2\ x1q\ x2L^3}{3}\right.$$
$$-b1\ b2\ x2L\ x2q - a1\ b2\ x1q\ x2L\ x2q$$
$$-\frac{a2\ b1\ x2L^2\ x2q}{2} - \frac{a1\ a2\ x1q\ x2L^2\ x2q}{2} +$$
$$b1\ b2\ x2q\ x2u + a1\ b2\ x1q\ x2q\ x2u - \frac{b1\ b2\ x2u^2}{2} - \frac{a1\ b2\ x1q\ x2u^2}{2} +$$
$$\left.\frac{a2\ b1\ x2q\ x2u^2}{2} + \frac{a1\ a2\ x1q\ x2q\ x2u^2}{2} - \frac{a2\ b1\ x2u^3}{3} - \frac{a1\ a2\ x1q\ x2u^3}{3}\right) +$$
$$\frac{1}{t^2}\left(\frac{-\left(a1\ b2\ x2L^3\right)}{6} - \frac{a1\ a2\ x2L^4}{8} + \frac{a1\ b2\ x2L^2\ x2q}{2} + \frac{a1\ a2\ x2L^3\ x2q}{3} -\right.$$
$$\frac{a1\ b2\ x2L\ x2q^2}{2} - \frac{a1\ a2\ x2L^2\ x2q^2}{4} + \frac{a1\ b2\ x2q^2\ x2u}{2} - \frac{a1\ b2\ x2q\ x2u^2}{2} +$$
$$\left.\frac{a1\ a2\ x2q^2\ x2u^2}{4} + \frac{a1\ b2\ x2u^3}{6} - \frac{a1\ a2\ x2qx2u^3}{3} + \frac{a1\ a2\ x2u^4}{8}\right)$$

Case D:

$$\int_{x2L-(x2-x2q)/t+x1q}^{x2u}\int^{x1u}(a_1x_1+b_1)(a_2x_2+b_2)dx_1dx_2 =$$

$$\begin{aligned}
&b1\ b2\ x1q\ x2L + \tfrac{a1\ b2\ x1q^2\ x2L}{2} - b1\ b2\ x1u\ x2L - \tfrac{a1\ b2\ x1u^2\ x2L}{2} + \\
&\tfrac{a2\ b1\ x1q\ x2L^2}{2} + \tfrac{a1\ a2\ x1q^2\ x2L^2}{4} - \tfrac{a2\ b1\ x1u\ x2L^2}{2} - \tfrac{a1\ a2\ x1u^2\ x2L^2}{4} - \\
&b1\ b2\ x1q\ x2u - \tfrac{a1\ b2\ x1q^2\ x2u}{2} + b1\ b2\ x1u\ x2u + \tfrac{a1\ b2\ x1u^2\ x2u}{2} - \\
&\tfrac{a2\ b1\ x1q\ x2u^2}{2} - \tfrac{a1\ a2\ x1q^2\ x2u^2}{4} + \tfrac{a2\ b1\ x1u\ x2u^2}{2} + \tfrac{a1\ a2\ x1u^2\ x2u^2}{4} + \\
\tfrac{1}{t}\ &\left(\tfrac{-\left(b1\ b2\ x2L^2\right)}{2} - \tfrac{a1\ b2\ x1q\ x2L^2}{2} - \tfrac{a2\ b1\ x2L^3}{3} - \tfrac{a1\ a2\ x1q\ x2L^3}{3} + \right. \\
&b1\ b2\ x2L\ x2q + a1\ b2\ x1q\ x2L\ x2q + \tfrac{a2\ b1\ x2L^2\ x2q}{2} + \tfrac{a1\ a2\ x1q\ x2L^2\ x2q}{2} \\
&-b1\ b2\ x2q\ x2u - a1\ b2\ x1q\ x2q\ x2u + \tfrac{b1\ b2\ x2u^2}{2} + \tfrac{a1\ b2\ x1q\ x2u^2}{2} - \\
&\left.\tfrac{a2\ b1\ x2q\ x2u^2}{2} - \tfrac{a1\ a2\ x1q\ x2q\ x2u^2}{2} + \tfrac{a2\ b1\ x2u^3}{3} + \tfrac{a1\ a2\ x1q\ x2u^3}{3}\right) + \\
\tfrac{1}{t^2}\ &\left(\tfrac{a1\ b2\ x2L^3}{6} + \tfrac{a1\ a2\ x2L^4}{8} - \tfrac{a1\ b2\ x2L^2\ x2q}{2} - \tfrac{a1\ a2\ x2L^3\ x2q}{3} + \right. \\
&\tfrac{a1\ b2\ x2L\ x2q^2}{2} + \tfrac{a1\ a2\ x2L^2\ x2q^2}{4} - \tfrac{a1\ b2\ x2q^2\ x2u}{2} + \tfrac{a1\ b2\ x2q\ x2u^2}{2} - \\
&\left.\tfrac{a1\ a2\ x2q^2\ x2u^2}{4} - \tfrac{a1\ b2\ x2u^3}{6} + \tfrac{a1\ a2\ x2q\ x2u^3}{3} - \tfrac{a1\ a2\ x2u^4}{8}\right)
\end{aligned}$$

Case E:

$$\int_{x1L}^{-(x2L-x2q)/t+x1q} \int_{-t(x1-x1q)+x2q}^{x2u} (a_1x_1+b_1)(a_2x_2+b_2)dx_2dx_1 =$$

$$\begin{aligned}
t^2 \quad & \left(\frac{a2\ b1\ x1L^3}{6} + \frac{a1\ a2\ x1L^4}{8} - \frac{a2\ b1\ x1L^2\ x1q}{2} - \frac{a1\ a2\ x1L^3\ x1q}{3} + \right. \\
& \left. \frac{a2\ b1\ x1L\ x1q^2}{2} + \frac{a1\ a2\ x1L^2\ x1q^2}{4} - \frac{a2\ b1\ x1q^3}{6} - \frac{a1\ a2\ x1q^4}{24}\right) + \\
t \quad & \left(\frac{-\left(b1\ b2\ x1L^2\right)}{2} - \frac{a1\ b2\ x1L^3}{3} + b1\ b2\ x1L\ x1q + \frac{a1\ b2\ x1L^2\ x1q}{2} - \right. \\
& \frac{b1\ b2\ x1q^2}{2} - \frac{a1\ b2\ x1q^3}{6} - \frac{a2\ b1\ x1L^2\ x2q}{2} - \frac{a1\ a2\ x1L^3\ x2q}{3} + \\
& \left. a2\ b1\ x1L\ x1q\ x2q + \frac{a1\ a2\ x1L^2\ x1q\ x2q}{2} - \frac{a2\ b1\ x1q^2\ x2q}{2} - \frac{a1\ a2\ x1q^3\ x2q}{6}\right) \\
& -b1\ b2\ x1L\ x2u - \frac{a1\ b2\ x1L^2\ x2u}{2} + b1\ b2\ x1q\ x2u + \frac{a1\ b2\ x1q^2\ x2u}{2} - \\
& \frac{a2\ b1\ x1L\ x2u^2}{2} - \frac{a1\ a2\ x1L^2\ x2u^2}{4} + \frac{a2\ b1\ x1q\ x2u^2}{2} + \frac{a1\ a2\ x1q^2\ x2u^2}{4} - \\
& b1\ x1q\ \left(-\left(b2\ x2L\right) - \frac{a2\ x2L^2}{2} + b2\ x2u + \frac{a2\ x2u^2}{2}\right) + \\
& b1\ b2\ x1L\ x2q + \frac{a1\ b2\ x1L^2\ x2q}{2} - b1\ b2\ x1q\ x2q - \frac{a1\ b2\ x1q^2\ x2q}{2} + \\
& \frac{a2\ b1\ x1L\ x2q^2}{2} + \frac{a1\ a2\ x1L^2\ x2q^2}{4} - \frac{a2\ b1\ x1q\ x2q^2}{2} - \frac{a1\ a2\ x1q^2\ x2q^2}{4} - \\
& \frac{a1\ x1q^2\ \left(-(b2\ x2L) - \frac{a2\ x2L^2}{2} + b2\ x2u + \frac{a2\ x2u^2}{2}\right)}{2} + b1\ x1u\ \left(-\left(b2\ x2L\right)\right. \\
& \left. + -\frac{a2\ x2L^2}{2} + b2\ x2u + \frac{a2\ x2u^2}{2}\right) \frac{a1\ x1u^2\ \left(-(b2\ x2L) - \frac{a2\ x2L^2}{2} + b2\ x2u + \frac{a2\ x2u^2}{2}\right)}{2} + \\
\frac{1}{t} \quad & \left(-\left(b1\ b2\ x2q\ \left(-x2L+x2q\right)\right) - a1\ b2\ x1q\ x2q\ \left(-x2L+x2q\right) - \right. \\
& \frac{a2\ b1\ x2q^2\ \left(-x2L+x2q\right)}{2} - \frac{a1\ a2\ x1q\ x2q^2\ \left(-x2L+x2q\right)}{2} + \frac{b1\ b2\ \left(-x2L+x2q\right)^2}{2} + \\
& \frac{a1\ b2\ x1q\ \left(-x2L+x2q\right)^2}{2} + \frac{a2\ b1\ x2q\ \left(-x2L+x2q\right)^2}{2} + \frac{a1\ a2\ x1q\ x2q\ \left(-x2L+x2q\right)^2}{2} - \\
& \frac{a2\ b1\ \left(-x2L+x2q\right)^3}{6} - \frac{a1\ a2\ x1q\ \left(-x2L+x2q\right)^3}{6} + b1\ b2\ \left(-x2L+x2q\right)\ x2u + \\
& a1\ b2\ x1q\ \left(-x2L+x2q\right)\ x2u + \\
& \frac{a2\ b1\ \left(-x2L+x2q\right)\ x2u^2}{2} + \frac{a1\ a2\ x1q\ \left(-x2L+x2q\right)\ x2u^2}{2} - \\
& b1\ \left(-x2L+x2q\right)\ \left(-\left(b2\ x2L\right) - \frac{a2\ x2L^2}{2} + b2\ x2u + \frac{a2\ x2u^2}{2}\right) - \\
& \left. a1\ x1q\ \left(-x2L+x2q\right)\ \left(-\left(b2\ x2L\right) - \frac{a2\ x2L^2}{2} + b2\ x2u + \frac{a2\ x2u^2}{2}\right)\right) + \\
\frac{1}{t^2} \quad & \left(\frac{-\left(a1\ b2\ x2q\ \left(-x2L+x2q\right)^2\right)}{2} - \frac{a1\ a2\ x2q^2\ \left(-x2L+x2q\right)^2}{4} + \frac{a1\ b2\ \left(-x2L+x2q\right)^3}{3} + \right. \\
& \frac{a1\ a2\ x2q\ \left(-x2L+x2q\right)^3}{3} - \frac{a1\ a2\ \left(-x2L+x2q\right)^4}{8} + \frac{a1\ b2\ \left(-x2L+x2q\right)^2\ x2u}{2} + \\
& \left. \frac{a1\ a2\ \left(-x2L+x2q\right)^2\ x2u^2}{4} - \frac{a1\ \left(-x2L+x2q\right)^2\ \left(-(b2\ x2L) - \frac{a2\ x2L^2}{2} + b2\ x2u + \frac{a2\ x2u^2}{2}\right)}{2}\right)
\end{aligned}$$

Case F:

$$\int_{-(x2L-x2q)/t+x1q-t(x1-x1q)+x2q}^{x1u}\int^{x2u}(a_1x_1+b_1)(a_2x_2+b_2)dx_2dx_1 =$$

$$\begin{aligned}
t^2\ & \left(\frac{a2\ b1\ x1q^3}{6}+\frac{a1\ a2\ x1q^4}{24}-\frac{a2\ b1\ x1q^2\ x1u}{2}+\frac{a2\ b1\ x1q\ x1u^2}{2}-\right.\\
&\left.\frac{a1\ a2\ x1q^2\ x1u^2}{4}-\frac{a2\ b1\ x1u^3}{6}+\frac{a1\ a2\ x1q\ x1u^3}{3}-\frac{a1\ a2\ x1u^4}{8}\right)+\\
t\ & \left(\frac{b1\ b2\ x1q^2}{2}+\frac{a1\ b2\ x1q^3}{6}-b1\ b2\ x1q\ x1u+\frac{b1\ b2\ x1u^2}{2}-\right.\\
&\frac{a1\ b2\ x1q\ x1u^2}{2}+\frac{a1\ b2\ x1u^3}{3}+\frac{a2\ b1\ x1q^2\ x2q}{2}+\frac{a1\ a2\ x1q^3\ x2q}{6}\\
&\left.-a2\ b1\ x1q\ x1u\ x2q+\frac{a2\ b1\ x1u^2\ x2q}{2}-\frac{a1\ a2\ x1q\ x1u^2\ x2q}{2}+\frac{a1\ a2\ x1u^3\ x2q}{3}\right)\\
&+b1\ b2\ x1q\ x2q+\frac{a1\ b2\ x1q^2\ x2q}{2}-b1\ b2\ x1u\ x2q-\frac{a1\ b2\ x1u^2\ x2q}{2}+\\
&\frac{a2\ b1\ x1q\ x2q^2}{2}+\frac{a1\ a2\ x1q^2\ x2q^2}{4}-\frac{a2\ b1\ x1u\ x2q^2}{2}-\frac{a1\ a2\ x1u^2\ x2q^2}{4}-\\
&b1\ b2\ x1q\ x2u-\frac{a1\ b2\ x1q^2\ x2u}{2}+b1\ b2\ x1u\ x2u+\frac{a1\ b2\ x1u^2\ x2u}{2}-\\
&\frac{a2\ b1\ x1q\ x2u^2}{2}-\frac{a1\ a2\ x1q^2\ x2u^2}{4}+\frac{a2\ b1\ x1u\ x2u^2}{2}+\frac{a1\ a2\ x1u^2\ x2u^2}{4}-\\
&b1\ x1L\ \left(-(b2\ x2L)-\frac{a2\ x2L^2}{2}+b2\ x2u+\frac{a2\ x2u^2}{2}\right)-\\
&\frac{a1\ x1L^2\ \left(-(b2\ x2L)-\frac{a2\ x2L^2}{2}+b2\ x2u+\frac{a2\ x2u^2}{2}\right)}{2}+\\
&b1\ x1q\ \left(-(b2\ x2L)-\frac{a2\ x2L^2}{2}+b2\ x2u+\frac{a2\ x2u^2}{2}\right)+\\
&\frac{a1\ x1q^2\ \left(-(b2\ x2L)-\frac{a2\ x2L^2}{2}+b2\ x2u+\frac{a2\ x2u^2}{2}\right)}{2}+\\
\frac{1}{t}\ & (b1\ b2\ x2q\ (-x2L+x2q)+a1\ b2\ x1q\ x2q\ (-x2L+x2q)+\\
&\frac{a2\ b1\ x2q^2\ (-x2L+x2q)}{2}+\frac{a1\ a2\ x1q\ x2q^2\ (-x2L+x2q)}{2}-\frac{b1\ b2\ (-x2L+x2q)^2}{2}-\\
&\frac{a1\ b2\ x1q\ (-x2L+x2q)^2}{2}-\frac{a2\ b1\ x2q\ (-x2L+x2q)^2}{2}-\frac{a1\ a2\ x1q\ x2q\ (-x2L+x2q)^2}{2}+\\
&\frac{a2\ b1\ (-x2L+x2q)^3}{6}+\frac{a1\ a2\ x1q\ (-x2L+x2q)^3}{6}-b1\ b2\ (-x2L+x2q)\ x2u-\\
&a1\ b2\ x1q\ (-x2L+x2q)\ x2u-\frac{a2\ b1\ (-x2L+x2q)\ x2u^2}{2}-\\
&\frac{a1\ a2\ x1q\ (-x2L+x2q)\ x2u^2}{2}+\\
&+b1\ (-x2L+x2q)\ \left(-(b2\ x2L)-\frac{a2\ x2L^2}{2}+b2\ x2u+\frac{a2\ x2u^2}{2}\right)+\\
&a1\ x1q\ (-x2L+x2q)\ \left(-(b2\ x2L)-\frac{a2\ x2L^2}{2}+b2\ x2u+\frac{a2\ x2u^2}{2}\right))\\
\frac{1}{t^2}\ & \left(\frac{a1\ b2\ x2q\ (-x2L+x2q)^2}{2}+\frac{a1\ a2\ x2q^2\ (-x2L+x2q)^2}{4}-\frac{a1\ b2\ (-x2L+x2q)^3}{3}-\right.\\
&\frac{a1\ a2\ x2q\ (-x2L+x2q)^3}{3}+\frac{a1\ a2\ (-x2L+x2q)^4}{8}-\frac{a1\ b2\ (-x2L+x2q)^2\ x2u}{2}-\\
&\left.\frac{a1\ a2\ (-x2L+x2q)^2\ x2u^2}{4}+\frac{a1\ (-x2L+x2q)^2\ \left(-(b2\ x2L)-\frac{a2\ x2L^2}{2}+b2\ x2u+\frac{a2\ x2u^2}{2}\right)}{2}\right)
\end{aligned}$$

Case G:

$$\int_{x2L}^{x2U} \int_{\frac{x2-x2q}{-t}+x1q}^{x1U} (a_1x_1 + b_1)(a_2x_2 + b_2)\ dx1dx2 =$$

$$\begin{aligned}
t^2 \quad & \left(\frac{a2\ b1\ x1L^3}{6} + \frac{a1\ a2\ x1L^4}{8} - \frac{a2\ b1\ x1L^2\ x1q}{2} - \frac{a1\ a2\ x1L^3\ x1q}{3} + \right. \\
& \quad \frac{a2\ b1\ x1L\ x1q^2}{2} + \frac{a1\ a2\ x1L^2\ x1q^2}{4} - \frac{a2\ b1\ x1q^2\ x1u}{2} + \frac{a2\ b1\ x1q\ x1u^2}{2} - \\
& \quad \left. \frac{a1\ a2\ x1q^2\ x1u^2}{4} - \frac{a2\ b1\ x1u^3}{6} + \frac{a1\ a2\ x1q\ x1u^3}{3} - \frac{a1\ a2\ x1u^4}{8} \right) + \\
t \quad & \left(\frac{-\left(b1\ b2\ x1L^2\right)}{2} - \frac{a1\ b2\ x1L^3}{3} + b1\ b2\ x1L\ x1q + \frac{a1\ b2\ x1L^2\ x1q}{2} - \right. \\
& \quad b1\ b2\ x1q\ x1u + \frac{b1\ b2\ x1u^2}{2} - \frac{a1\ b2\ x1q\ x1u^2}{2} + \frac{a1\ b2\ x1u^3}{3} - \\
& \quad \frac{a2\ b1\ x1L^2\ x2q}{2} - \frac{a1\ a2\ x1L^3\ x2q}{3} + a2\ b1\ x1L\ x1q\ x2q + \frac{a1\ a2\ x1L^2\ x1q\ x2q}{2} - \\
& \quad \left. a2\ b1\ x1q\ x1u\ x2q + \frac{a2\ b1\ x1u^2\ x2q}{2} - \frac{a1\ a2\ x1q\ x1u^2\ x2q}{2} + \frac{a1\ a2\ x1u^3\ x2q}{3} \right) \\
& -b1\ b2\ x1L\ x2u - \frac{a1\ b2\ x1L^2\ x2u}{2} + b1\ b2\ x1u\ x2u + \frac{a1\ b2\ x1u^2\ x2u}{2} - \\
& \frac{a2\ b1\ x1L\ x2u^2}{2} - \frac{a1\ a2\ x1L^2\ x2u^2}{4} + \frac{a2\ b1\ x1u\ x2u^2}{2} + \frac{a1\ a2\ x1u^2\ x2u^2}{4} + \\
& b1\ b2\ x1L\ x2q + \frac{a1\ b2\ x1L^2\ x2q}{2} - b1\ b2\ x1u\ x2q - \frac{a1\ b2\ x1u^2\ x2q}{2} + \\
& \frac{a2\ b1\ x1L\ x2q^2}{2} + \frac{a1\ a2\ x1L^2\ x2q^2}{4} - \frac{a2\ b1\ x1u\ x2q^2}{2} - \frac{a1\ a2\ x1u^2\ x2q^2}{4}
\end{aligned}$$

Case H:

$$\int_{x1L}^{x1U} \int_{x2L}^{x2U} (a1x1 + b1)(a2x2 + b2)\ dx2\ dx1 =$$

$$\begin{aligned}
& \left(-\left(b1\ x1L\right) - \frac{a1\ x1L^2}{2} + b1\ x1u + \frac{a1\ x1u^2}{2} \right) \times \\
& \left(-\left(b2\ x2L\right) - \frac{a2\ x2L^2}{2} + b2\ x2u + \frac{a2\ x2u^2}{2} \right)
\end{aligned}$$

APPENDIX B

Implementation

We implemented both estimation and exact algorithms for each aggregation operator. We designed the application as a 2-tier application in C# as shown in Figure B.1.

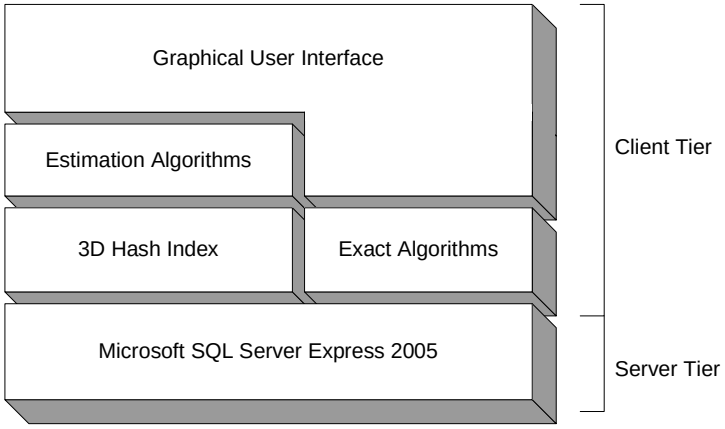


Figure B.1. Implementation architecture.

Microsoft SQL Server Express makes up the server tier and contains the point data in the form:

ID	$X1$	$X2$	$X3$	$X4$	$X5$	$X6$	$Count$

ID provides a key to the points table since we can't guarantee that every point will be unique. $Count$ allows clusters of points to appear in one place and move together. In all our experiments $Count = 1$.

Microsoft SQL server express did not provide indexing to any algorithms. Instead, we built the index on top of the database as shown in Figure B.1. The index exposes a method called GETCELLS() that may be overridden in a child object to test additional indices for future work. This method provides the selection of cells based on a query. The other methods including INSERT, DELETE and UPDATE would also need to be overridden in the inheritance structure.

The exact algorithms are simple algorithms used to check the answers of the estimation algorithms. Each estimation algorithm is implemented as a method call of the THRESHOLDESTIMATES object. Each query must build a TIMESEGMENTS list that is used in the evaluation. To optimize this process, aggregation queries are processed through the THRESHOLDESTIMATES object containing the TIMESEGMENTS that can be reused if the queries are similar. That is, the queries must have the same query points of the previous query, and at least a subset of the previous query's time interval. This query optimization significantly reduced our experimentation time. Similarly, the exact algorithms share a sorted list of points that enter and exit the query.

The graphical user interface provides three tabs shown in Figures B.2-B.5. The buttons across the top in order 1) save changes made to the database, 2) create or recreate the 3D index and 3) dump the index to the results pane under the queries tab. The "Point Set and Index Information" (Figure B.2) allows the user to specify: a) the parameters of the index and b) which "Point Set" to load and create the index from. When generating the index, the status bar at the bottom shows running time information. Here you can see hat the program created the index in 0.39 seconds.

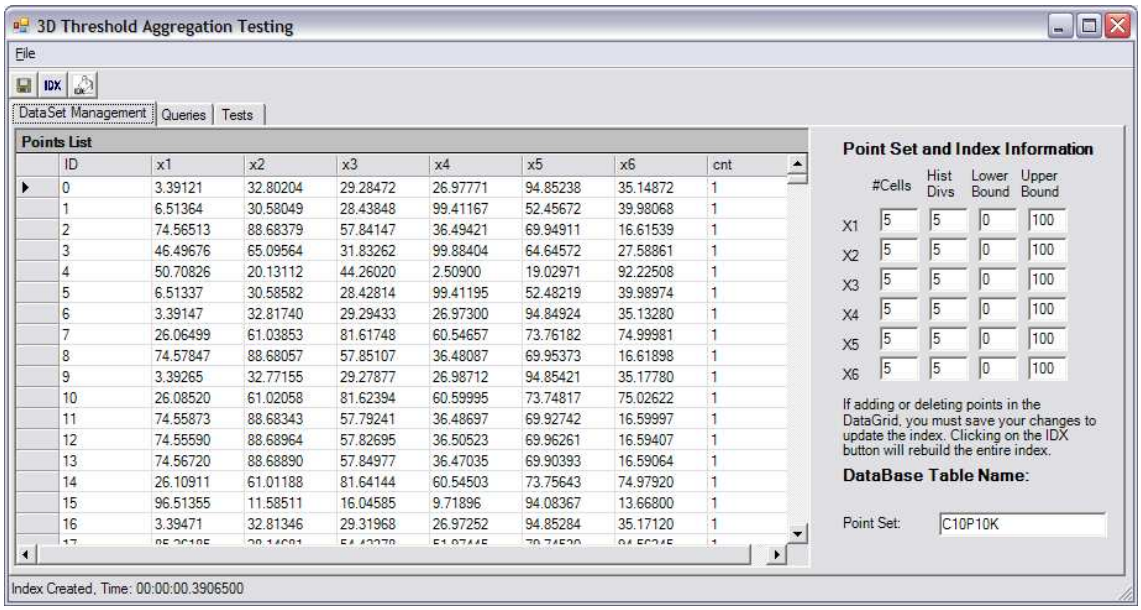


Figure B.2. Database and index parameters pane.

The query section, shown in Figure B.3, requires the user to specify a hyper-rectangle defined by "Point 1" and "Point 2", a time interval and threshold value. These points make up the parameters used in the twelve possible queries. The last two buttons at the right run a series of pre-defined queries using all the different aggregation operations against the current index.

The test pane (Figure B.4) provides an inside look into what problems we ran into and test cases for them. Each solution to the different integral cases provides almost limitless opportunity for errors. The tests for each integral case includes a laboriously hand-calculated gold-standard against which we tested the various pieces of code. Other checks examine different outputs for sane values.

We generated data for experiments in the application and stored it in the SQL database using methods triggered by the buttons shown in Figure B.5. The special

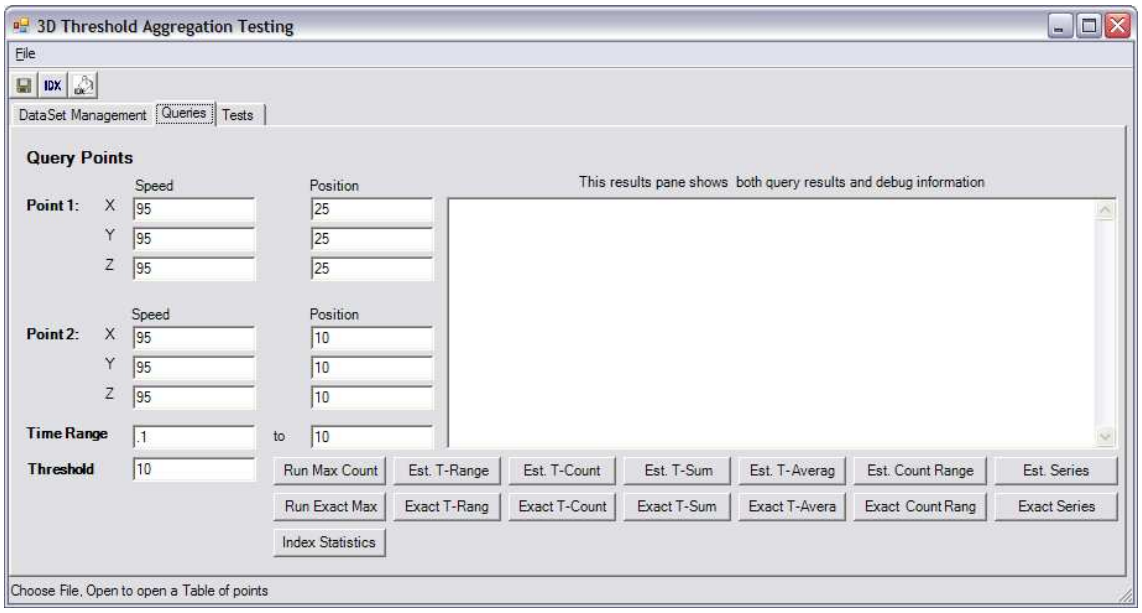


Figure B.3. Application query pane.

Data set Creator is for testing purposes only. Here the user may specify a connection string to the database [1].

The file menu (Figure B.6) includes commands to:

- Create an index for a particular data set,
- Save changes to the data set,
- Save results to a Comma Separated Values (CSV) file,
- Import CSV data, and
- Start the random data set generator form.

Output is given in CSV format in the results pane shown in Figure B.3. The format is given as: "Query Type" followed by $name, value$ pairs. For example, the exact solution to a MaxCount query is given as:

[1]If you plan on using the code you will need to change the connection string here and in the code for loading data.

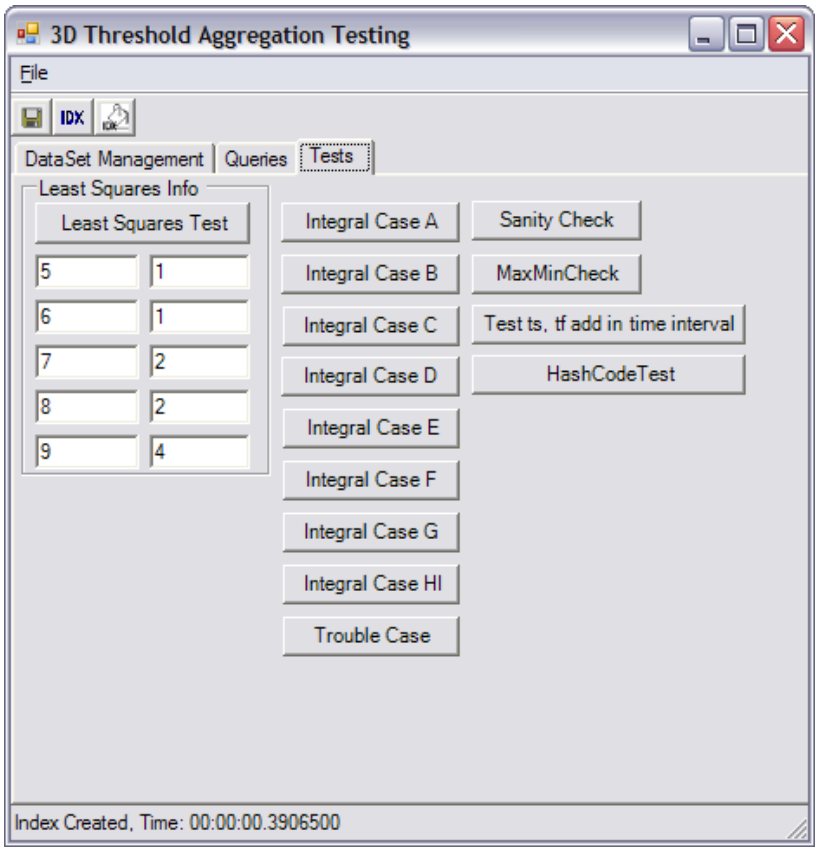


Figure B.4. Application testing pane.

MaxCount EXACT,Time,0.197428,Count,36,Skipped,9894,Used,106,rt,00:00:00.093747

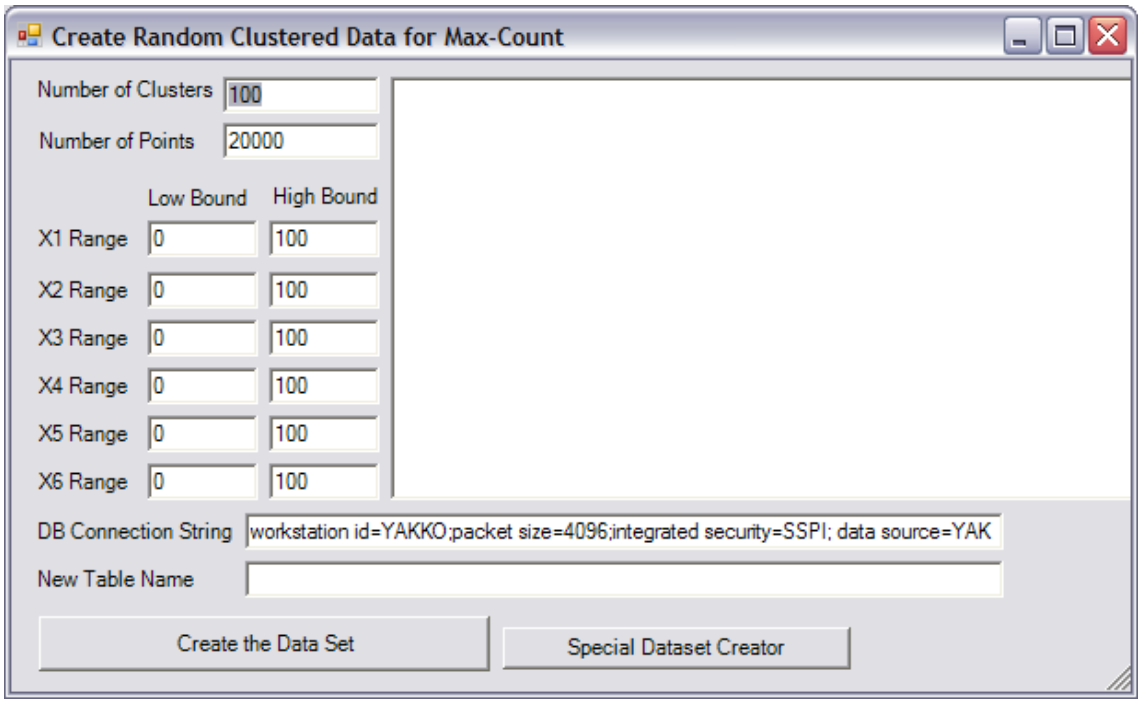


Figure B.5. Creating new random data sets.

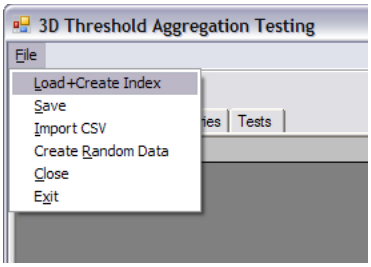


Figure B.6. File menu commands.

APPENDIX C

Additional Results

This Appendix shows results not discussed in the paper. We have a section for each type of operator discussed.

C.1. ThresholdRange Results

Figures C.1-C.6 Show the results for both ThresholdRange Error and ThresholdRange excess error defined in Equations (10.2) and (10.3).

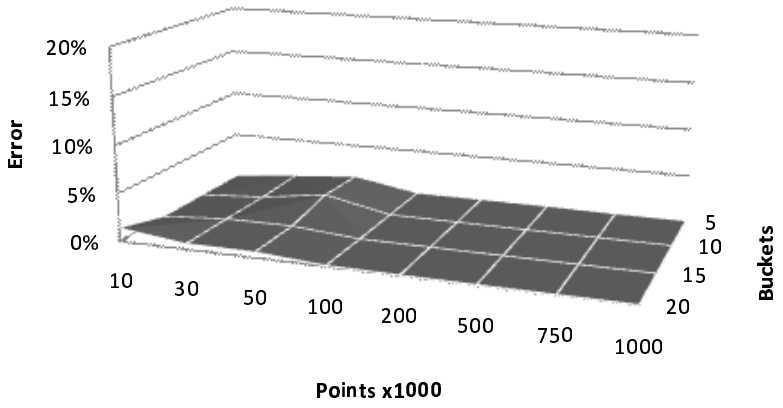


Figure C.1. ThresholdRange error, T=20.

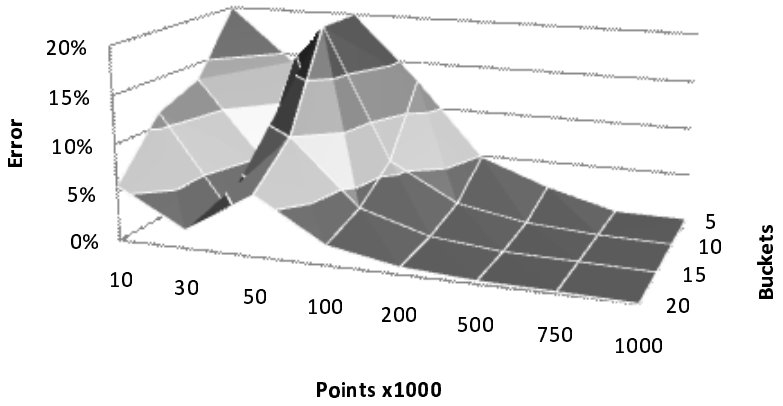


Figure C.2. ThresholdRange excess error, T=20.

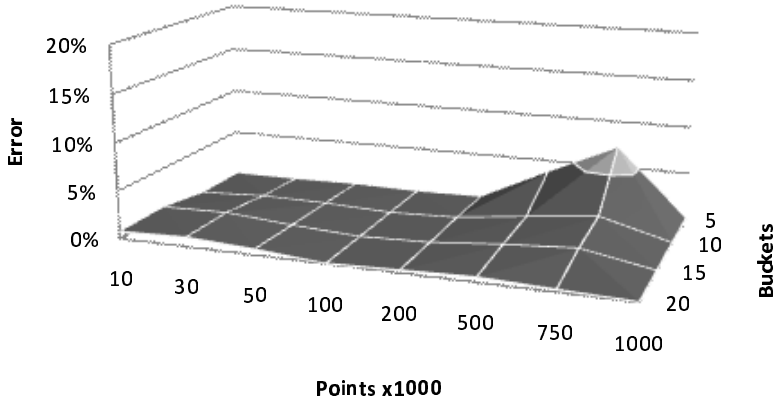


Figure C.3. ThresholdRange error, T=100.

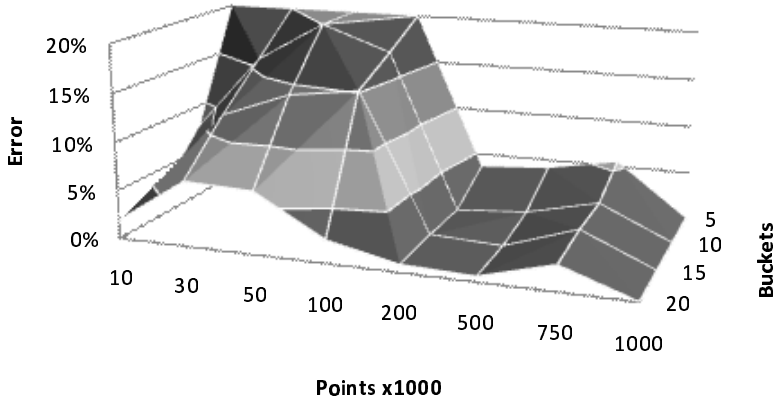


Figure C.4. ThresholdRange excess error, T=100.

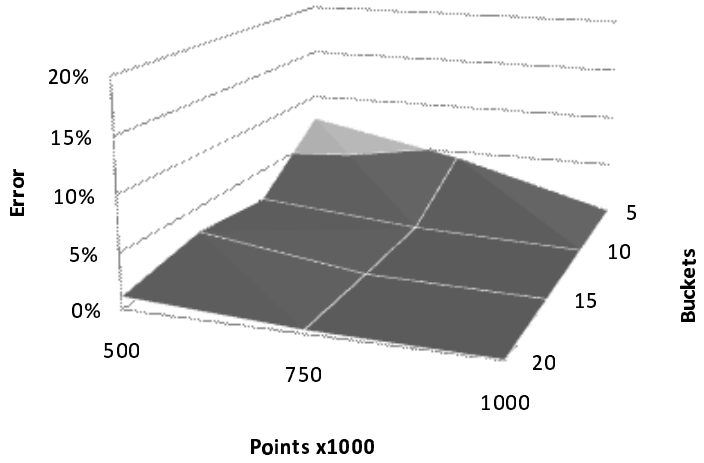


Figure C.5. ThresholdRange error, T=10000.

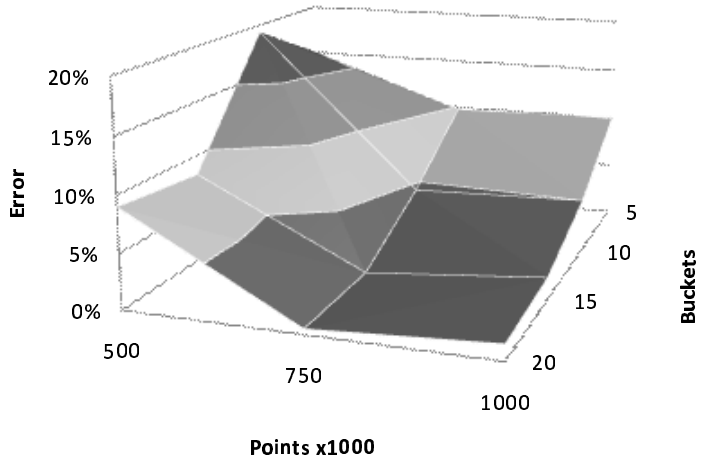


Figure C.6. ThresholdRange excess error, T=10000.

C.2. ThresholdCount Results

Figures C.7-C.10 give the results for ThresholdCount where $T = 20, 100, 10000, 100000$. All errors are relative as given in Equation (10.1).

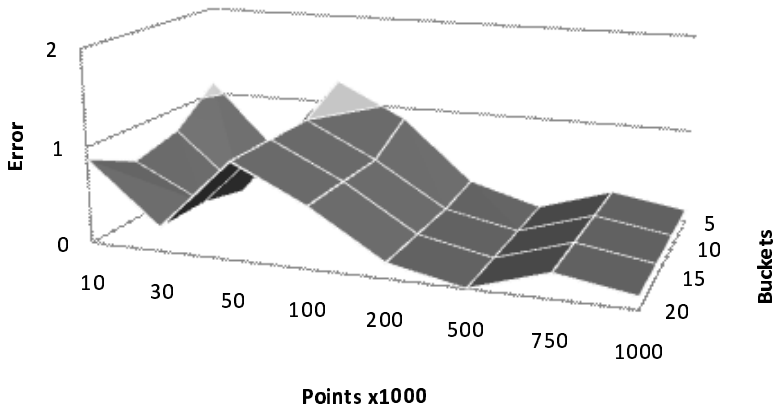


Figure C.7. ThresholdCount error, T=20.

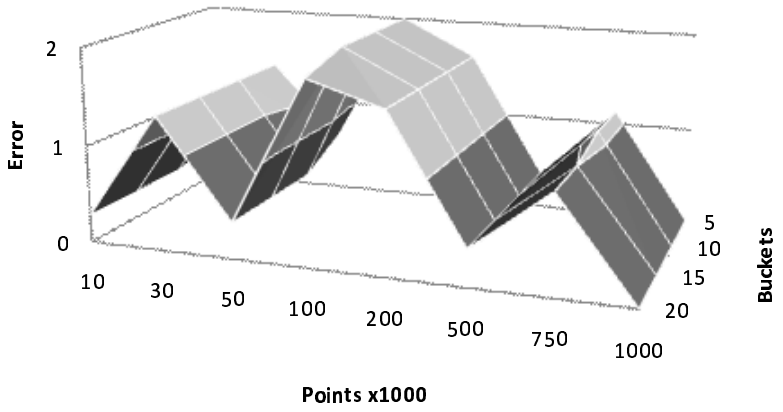


Figure C.8. ThresholdCount error, T=100.

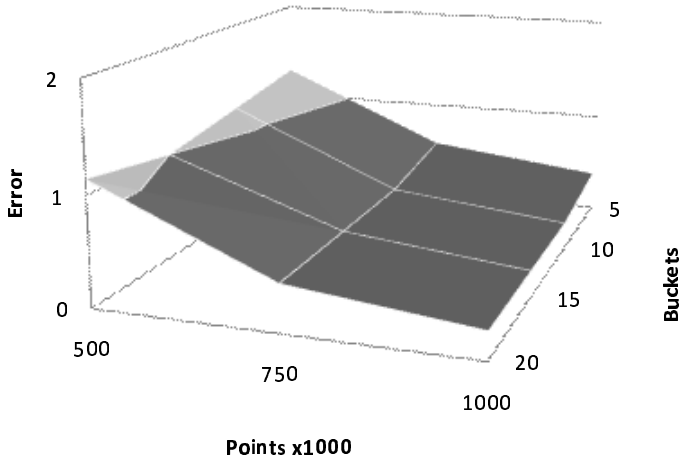


Figure C.9. ThresholdCount error, T=10000.

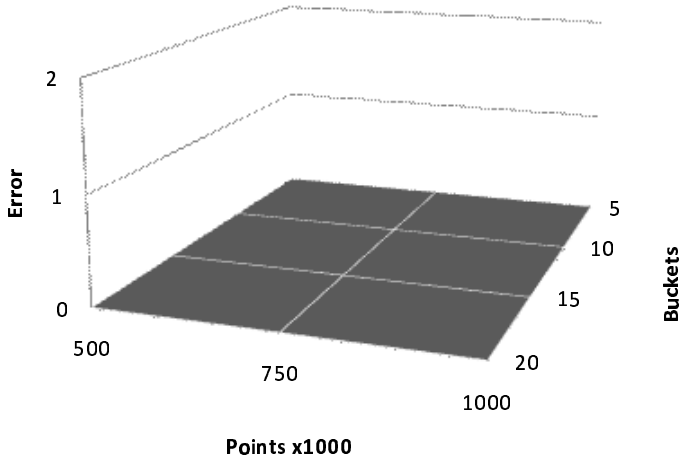


Figure C.10. ThresholdCount error, T=100000.

C.3. ThresholdSum Results

Figures C.11-C.14 give the results for ThresholdSum where $T = 20, 100, 1000, 10000$. All errors are relative as given in Equation (10.1).

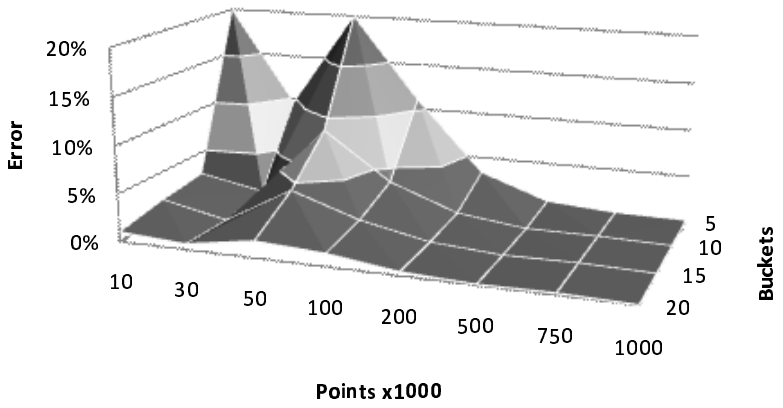


Figure C.11. ThresholdSum error, T=20.

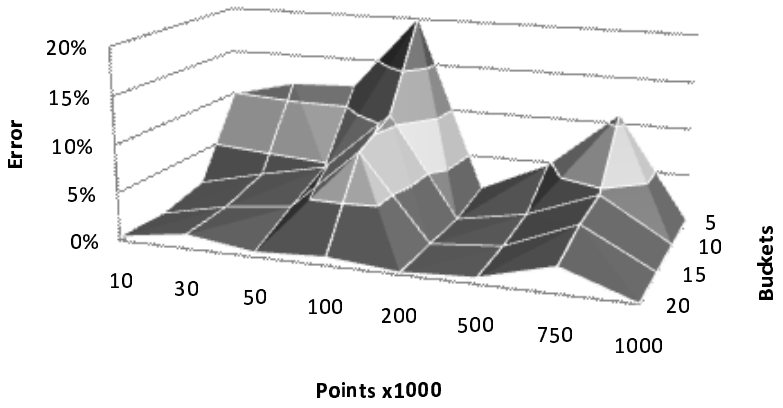


Figure C.12. ThresholdSum error, T=100.

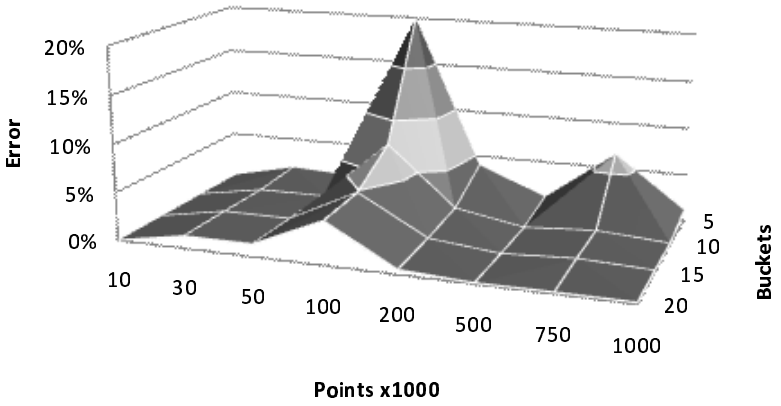


Figure C.13. ThresholdSum error, T=1000.

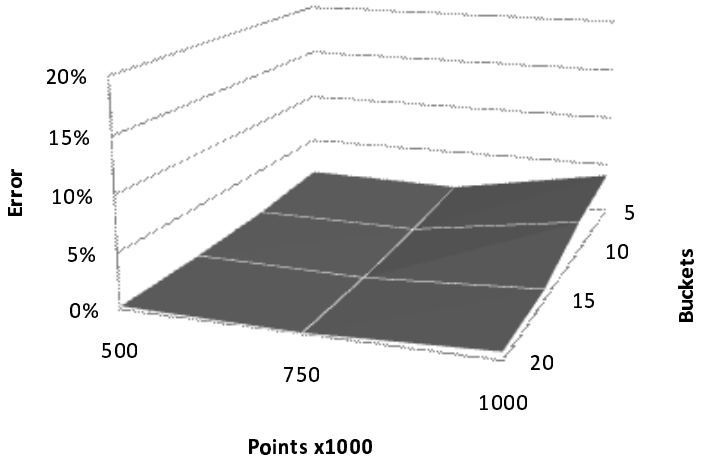


Figure C.14. ThresholdSum error, T=10000.

C.4. ThresholdAverage Results

Figures C.15-C.18 give the results for ThresholdSum where $T = 20, 100, 10000, 100000$. All errors are relative as given in Equation (10.1).

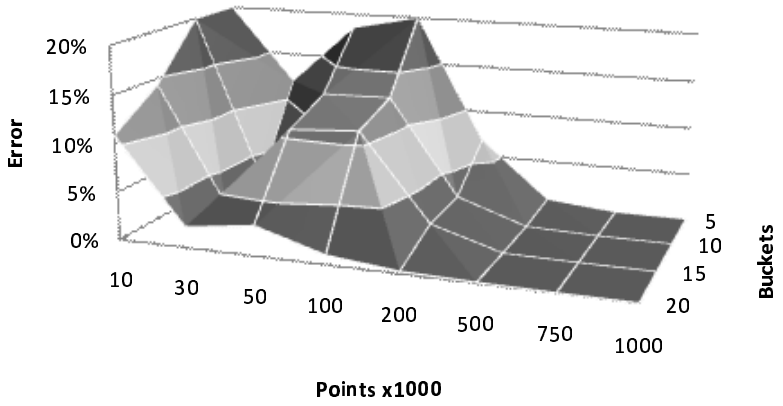


Figure C.15. ThresholdAverage error, T=20.

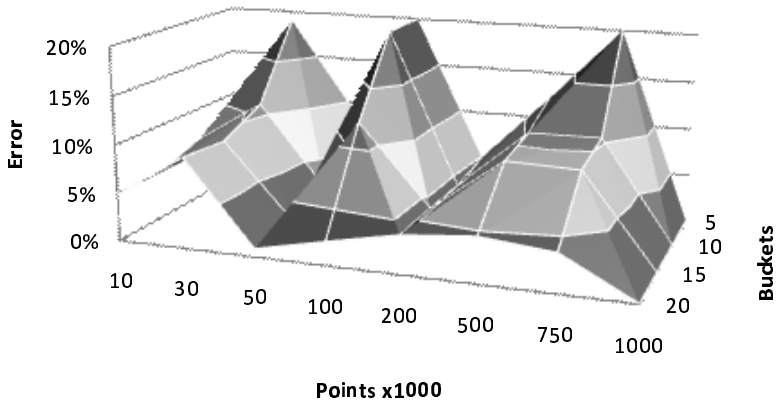


Figure C.16. ThresholdAverage error, T=100.

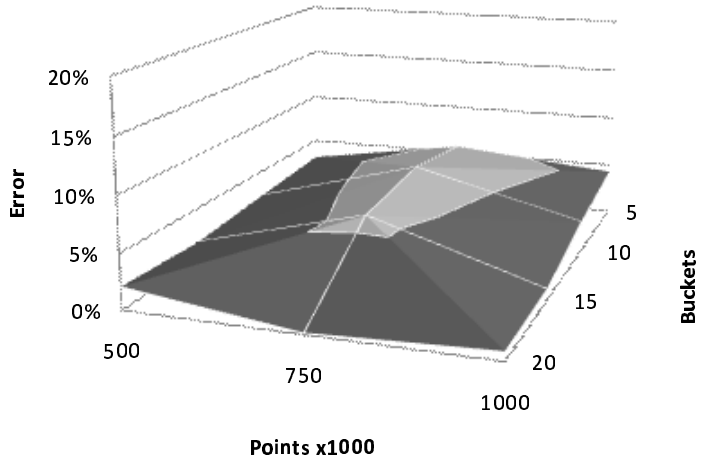


Figure C.17. ThresholdAverage error, T=10000.

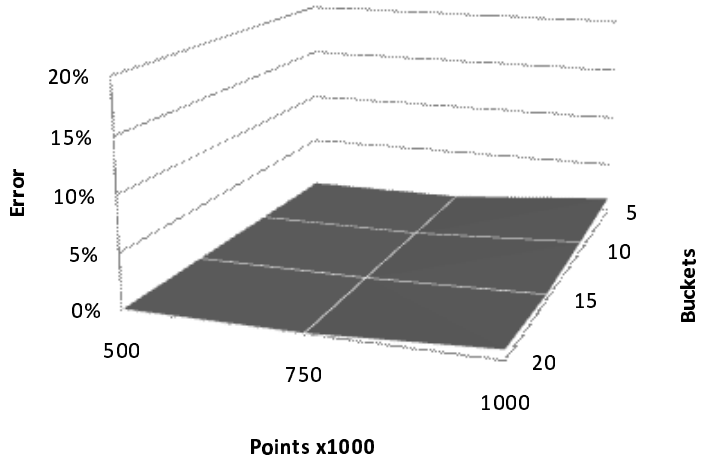


Figure C.18. ThresholdAverage error, T=100000.

References

Acharya, S., Poosala, V. & Ramaswamy, S. (1999), Selectivity estimation in spatial databases, *in* 'Proceedings of the ACM SIGMOD International Conference on Management of Data', ACM Press, New York, NY, USA, pp. 13–24.

Agarwal, P. K., Arge, L. & Erickson, J. (2003), 'Indexing moving points', *Journal of Computer and System Sciences* **66**(1), 207–243.

Anderson, S. (2003), Constraint datalog in trust management, Master's thesis, University of Nebraska, Lincoln.

Anderson, S. (2006), Aggregation estimation for 2D moving points, *in* 'Thirteenth International Symposium on Temporal Representation and Reasoning', IEEE Computer Society Press, Piscataway, NJ, USA, pp. 137–144.

Anderson, S. & Revesz, P. (2005), Verifying the incorrectness of programs and automata, *in* 'Proceedings of the 6th International Conference on Symposium on Abstraction, Reformulation and Approximation', Vol. 3607, Springer Verlag, pp. 1–13.

Anderson, S. & Revesz, P. (2007*a*), CDB-PV: A constraint database-based program verifier, *in* 'Proceedings of the 7th International Conference on Symposium on Abstraction, Reformulation and Approximation (submitted)', Springer Verlag.

Anderson, S. & Revesz, P. (2007*b*), 'Efficient aggregation of moving objects', *GeoInforatica (submitted)* .

Bagnara, R., Hill, P., Ricci, E. & Zaffanella, E. (2005), 'Precise widening operators for convex polyhedra', *Science of Computer Programming* **58**(1-2), 28–56.

Beckmann, N., Kriegel, H. P., Schneider, R. & Seeger, B. (1990), The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, *in* 'Proceedings of ACM/SIGMOD Annual Conference on Management of Data (SIGMOD)', pp. 322–331.

Bohlen, M. H., Gamper, J. & Jensen, C. S. (2006), How would you like to aggregate your temporal data?, *in* 'Thirteenth International Symposium on Temporal Representation and Reasoning', pp. 121–136.

Boigelot, B. & Wolper, P. (1994), Symbolic verification with periodic sets, *in* 'International Conference on Computer Aided Verification (CAV)', pp. 55–67.

Cai, M. & Revesz, P. (2000), Parametric R-tree: An index structure for moving objects, *in* 'Proceedings of the 10th COMAD International Conference on Management of Data', Tata McGraw-Hill, pp. 57–64.

Chen, Y. & Revesz, P. (2004), Max-count aggregation estimation for moving points, *in* 'Proceedings of the 11th International Symposium on Temporal Representation and Reasoning', pp. 103–108.

Choi, Y.-J. & Chung, C.-W. (2002), Selectivity estimation for spatio-temporal queries to moving objects, *in* 'Proceedings of the ACM SIGMOD International Conference on Management of Data', pp. 440–451.

Civilis, A., Jensen, C. S., Nenortaite, J. & Pakalnis, S. (2004), 'Efficient tracking of moving objects with precision guarantees', *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services* pp. 164–173.

Civilis, A., Jensen, C. S. & Pakalnis, S. (2005), 'Techniques for efficient road-network-based tracking of moving objects', *IEEE Transactions on Knowledge and Data Engineering* **17**(5), 698–712.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2001), *Introduction to Algorithms*, 2nd edn, MIT Press, Massachusetts.

Cousot, P. (2005), Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming, *in* 'Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)', Springer, Berlin, Paris, France, LNCS 3385, pp. 1–24.

Cousot, P. & Cousot, R. (1976), 'Static determination of dynamic properties of programs', *Proceedings of the Second International Symposium on Programming* pp. 106–130.

Cousot, P. & Cousot, R. (1977), Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints, *in* 'Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages', pp. 238–252.

Cousot, P. & Cousot, R. (1992*a*), 'Abstract interpretation frameworks.', *J. Log. Comput.* **2**(4), 511–547.

Cousot, P. & Cousot, R. (1992*b*), Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, *in* M. Bruynooghe & M. Wirsing, eds, 'Proceedings of the Fourth International Symposium on Programming Language Implementation and Logic Programming', LNCS 631, Springer-Verlag, Leuven, Belgium, pp. 269 –295.

Cousot, P. & Halbwachs, N. (1978), 'Automatic discovery of linear restraints among variables of a program', *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages* pp. 84–96.

Crhová, J., Krčál, P., Strejček, J., Šafránek, D. & Šimeček, P. (2002), Yahoda: verification tools database (http://anna.fi.muni.cz/yahoda/, Jan. 2007), *in* 'Proceedings of Tools Day', Brno FI MU, Brno, Czech Republic, pp. 99–103.

URL: *http://anna.fi.muni.cz/yahoda/*

Floyd, R. & Beigel, R. (1994), *The language of machines: an introduction to computability and formal languages*, Computer Science Press, Inc. New York, NY, USA.

Fribourg, L. & Olsén, H. (1997), 'A Decompositional Approach for Computing Least Fixed-Points of Datalog Programs with Z-Counters', *Constraints* **2**(3), 305–335.

Fribourg, L. & Richardson, J. (1996), 'Symbolic Verification with Gap-Order Constraints', *Proc. 6th Intl. Workshop on Logic Program Synthesis and Transformation (LOPSTR), LNCS* **1207**, 20–37.

Gunopulos, D., Kollios, G., Tsotras, J. & Domeniconi, C. (2005), 'Selectivity estimators for multidimensional range queries over real attributes', *The VLDB Journal* **14**(2), 137–154.

Gupta, S., Kopparty, S. & Ravishankar, C. V. (2004), Roads, codes and spatiotemporal queries., *in* 'Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems', pp. 115–124.

Guting, R. H. & Schneider, M. (2005), *Moving Objects Databases*, Morgan Kaufmann.

Guttman, A. (1984), R-trees: A dynamic index structure for spatial searching., *in* B. Yormark, ed., 'Proceedings of the ACM SIGMOD International Conference on Management of Data', ACM Press, pp. 47–57.

Hadjieleftheriou, M., Kollios, G., Gunopulos, D. & Tsotras, V. J. (2003), On-line discovery of dense areas in spatio-temporal databases, *in* 'Advances in Spatial and Temporal Databases, 8th International Symposium', Springer, pp. 306–324.

Halbwachs, N. (1979), *Détermination automatique de relations linéaires vérifiées par les variables d'un programme*, Institut National Polytechnique.

Halbwachs, N. (1993), Delay analysis in synchronous programs, *in* 'CAV '93: Proceedings of the 5th International Conference on Computer Aided Verification', Springer-Verlag, London, UK, pp. 333–346.

Jaffar, J. & Maher, M. J. (1994), 'Constraint logic programming: A survey.', *J. Log. Program.* **19/20**, 503–581.

Kerbrat, A. (1995), Reachable state space analysis of lotos specifications, *in* 'Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques VII', Chapman & Hall, Ltd., London, UK, UK, pp. 181–196.

Kilroy, C. (1997), 'Investigation: Air france 296', `http://www.airdisaster.com/investigations/af296/af296.shtml`.

Kollios, G., Gunopulos, D. & Tsotras, V. J. (1999), On indexing mobile objects, *in* 'Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems', pp. 261–272.

Kollios, G., Papadopoulos, D., Gunopulos, D. & Tsotras, J. (2005), 'Indexing mobile objects using dual transformations', *The VLDB Journal* **14**(2), 238–256.

Kuper, G. M., Libkin, L. & Paredaens, J., eds (2000), *Constraint Databases*, Springer.

Marriott, K. & Stuckey, P. (1998), *Programming with Constraints: An Introduction*, MIT Press.

Marsaglia, G. & Tsang, W. (2000), 'The ziggurat method for generating random variables', *Journal of Statistical Software* **5**(8), 1–7.

Matiyasevich, Y. V. (1993), *Hilbert's Tenth Problem*, MIT Press.

Miné, A. (2001), The octagon abstract domain, *in* 'In Proceedings Analysis, Slicing and Transformation', IEEE Press, pp. 310–319.

Mokhtar, H., Su, J. & Ibarra, O. (2002), On moving object queries: (extended abstract), *in* 'Proceedings of the 21st Symposium on Principles of Database Systems', pp. 188–198.

Nascimento, M., Pfoser, D. & Theodoridis, Y. (2003), 'Synthetic and Real Spatiotemporal Datasets', *IEEE Data Engineering Bulletin* **26**(2), 26–32.

Papadopoulos, D., Kollios, G., Gunopulos, D. & Tsotras, V. J. (2002), Indexing mobile objects on the plane, *in* 'Proceedings of the International Conference on Database and Expert Systems Applications', Aix en Provence, France.

Pelanis, M., Saltenis, S. & Jensen, C. S. (2006), 'Indexing the past, present, and anticipated future positions of moving objects', *ACM Trans. Database Syst.* **31**(1), 255–298.

Porkaew, K., Lazaridis, I. & Mehrotra, S. (2001), Querying Mobile Objects in Spatio-Temporal Databases, *in* 'Proceedings of Symposium on Spatial and Temporal Databases (SSTD)', pp. 59–78.

Revesz, P. (1998), 'Safe query languages for constraint databases.', *ACM Trans. Database Syst.* **23**(1), 58–99.

Revesz, P. (1999), Datalog programs with difference constraints, *in* '12th International Conference on Applications of Prolog', pp. 69–76.

Revesz, P. (2002), *Introduction to Constraint Databases*, Springer-Verlag.

Revesz, P. (2005), Efficient rectangle indexing algorithms based on point dominance, *in* 'Proceedings of the 12th International Symposium on Temporal Representation and Reasoning', IEEE Computer Society Press.

Revesz, P. (2007), The constraint database approach to software verification, *in* '8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation', LNCS 4349, pp. 329–345.

Revesz, P., Chen, R., Kanjamala, P., Li, Y., Liu, Y. & Wang, Y. (2000), The MLPQ/GIS constraint database system, *in* 'ACM SIGMOD International Conference on Management of Data'.

Revesz, P. & Chen, Y. (2003), Efficient aggregation over moving objects, *in* 'Proceedings of the 10th International Symposium on Temporal Representation and Reasoning, Fourth International Conference on Temporal Logic', pp. 118–127.

Rigaux, P., Scholl, M. & Voisard, A. (2001), *Spatial Databases: With Applications to GIS*, Morgan Kaufmann Publishers, San Francisco, CA, USA.

Saltenis, S., Jensen, C. S., Leutenegger, S. T. & Lopez, M. A. (2000), Indexing the positions of continuously moving objects, *in* 'Proceedings of the ACM SIGMOD International Conference on Management of Data', pp. 331–342.

Samet, H. (1990), *The design and analysis of spatial data structures*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Samet, H. (2005), *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann Publishers, San Francisco, CA.

Tao, Y. & Papadias, D. (2005), 'Historical spatio-temporal aggregation', *ACM Trans. Inf. Syst.* **23**(1), 61–102.

Tao, Y., Papadias, D. & Sun, J. (2003), The TPR*-tree: An optimized spatio-temporal access method for predictive queries, *in* 'Proceedings of the Twenty-ninth International Conference on Very Large Data Bases'.

Tao, Y., Sun, J. & Papadias, D. (2003), Selectivity estimation for predictive spatio-temporal queries, *in* 'Proceedings of the 19th International Conference on Data Engineering', pp. 417–428.

Tayeb, J., Ulusoy, Ö. & Wolfson, O. (1998), 'A quadtree-based dynamic attribute indexing method.', *Comput. J.* **41**(3), 185–200.

Theodoridis, Y., Silva, J. & Nascimento, M. (1999), 'On the Generation of Spatiotemporal Datasets', *Proc. SSD* pp. 147–164.

Trajcevski, G., Wolfson, O., Hinrichs, K. & Chamberlain, S. (2004), 'Managing uncertainty in moving objects databases', *ACM Trans. Database Syst.* **29**(3), 463–507.

Tzouramanis, T., Vassilakopoulos, M. & Manolopoulos, Y. (2002), 'On the Generation of Time-Evolving Regional Data*', *GeoInformatica* **6**(3), 207–231.

Wolfson, O. & Yin, H. (2003), Accuracy and resource consumption in tracking and location prediction, *in* 'Proceedings of the Symposium on Spatial and Temporal Databases (SSTD)', pp. 325–343.

Zhang, D., Gunopulos, D., Tsotras, V. J. & Seeger, B. (2003), 'Temporal and spatio-temporal aggregations over data streams using multiple time granularities', *Inf. Syst.* **28**(1-2), 61–84.

Zhang, D., Markowetz, A., Tsotras, V., Gunopulos, D. & Seeger, B. (2001), Efficient computation of temporal aggregates with range predicates, *in* 'Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems', pp. 237–245.