

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

1996

Semi-join strategies for total cost minimization in distributed query processing.

William Todd Bealor
University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Bealor, William Todd, "Semi-join strategies for total cost minimization in distributed query processing." (1996). *Electronic Theses and Dissertations*. 3426.
<https://scholar.uwindsor.ca/etd/3426>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.



National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your fee - votre contribution

Our fee - notre contribution

NOTICE

AVIS

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

If pages are missing, contact the university which granted the degree.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

**Semi-join Strategies for
Total Cost Minimization in
Distributed Query Processing**

by

William T. Bealor

A Thesis

**Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in Partial
Fulfillment of the Requirements for the
Degree of Master of Science at the
University of Windsor**

**Windsor, Ontario, Canada
1995**



National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file / Votre référence

Our file / Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-10975-5

Canada

Name William T. Keeler

Dissertation Abstracts International and Masters Abstracts International are arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation or thesis. Enter the corresponding four-digit code in the spaces provided.

Computer Science

SUBJECT TERM

0984

UMI

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS	
Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Journalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465
EDUCATION	
General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0272
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0725
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0523

Psychology	0525
Reading	0535
Religious	0527
Sciences	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romanic	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	0578
General	0578

Ancient	0579
Medieval	0581
Modern	0582
Black	0328
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	0317
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760
EARTH SCIENCES	
Biogeochemistry	0425
Geochemistry	0996

Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleocology	0426
Paleontology	0418
Paleozoology	0985
Palynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0749
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575

Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences	
Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Science	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463
Applied Sciences	
Applied Mechanics	0346
Computer Science	0984

Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994
PSYCHOLOGY	
General	0621
Behavioral	0384
Clinical	0622
Developmental	0620
Experimental	0623
Industrial	0624
Personality	0625
Physiological	0989
Psychobiology	0349
Psychometrics	0632
Social	0451


William T. Bealor 1995
© All Rights Reserved

APPROVED BY:



Dr. J. M. Morrissey, School of Computer Science

Supervisor



Dr. S. Bandyopadhyay, School of Computer Science

Departmental Reader



Dr. R. Pinto, Department of Philosophy

External Reader

Abstract

A new static heuristic, called Algorithm W is presented as an efficient method for reducing the total volume of data transmitted over the network during distributed query processing. It uses the concepts of profit, marginal profit and gain to construct small, highly selective reducers using cost-effective semi-join sequences. In most cases the heuristic has a complexity of $O(nm)$. A limitation of static strategies, such as Algorithm W is that they rely on accurate estimates to perform properly. The presence of estimation errors may lead to sub-optimal solutions. A solution to this problem is the use of a dynamic strategy (Boderick, 1985; Boderick, Pyra *et al.*, 1989) in which the schedule of operations is monitored and corrected if the performance deteriorates. A purely dynamic heuristic, Algorithm DW is proposed which uses up to date information eliminating the need for schedule monitoring. It is shown that the overheads incurred by using exact information are minimal with respect to the overall total cost. A benchmark database is proposed upon which the empirical performance of the heuristics can be measured. Algorithm W is evaluated against the AHY General (total time) algorithm (Apers, Hevner, Yao, 1983) to investigate whether improvements are possible. The performance of the proposed heuristics are evaluated to test the hypothesis that a dynamic strategy using better estimates will produce improved schedules.

To my mother and father...

Acknowledgements

This work could not have been accomplished without the help of so many people. I would like to thank Dr. Morrissey for the ideas behind the heuristics along with the concept of marginal profit and the related proofs. Thanks to Dr. Bandyopadhyay and Dr. Pinto for their comments on my thesis. I appreciated all of the comments and discussions with all of my colleagues in the database research group. I would also like to thank a couple of good friends, Mounia whose friendship and support helped me get through some tough times and Steve for all of his technical support as well as being a good friend. Lastly, I would like to thank my parents for all of their support and understanding over these last few years.

Contents

Abstract	iv
Acknowledgements	vi
List of Figures	ix
List of Tables	x
Chapter 1 Introduction	I
1.1 THESIS ORGANIZATION	4
Chapter 2 Background	6
2.1 QUERY OPTIMIZATION OBJECTIVES	6
2.2 ESTIMATION	7
2.3 SEMI-JOINS AS REDUCERS	11
2.4 STATIC STRATEGIES	14
2.4.1 Two and Three Phased Approaches	15
2.5 DYNAMIC STRATEGIES	17
Chapter 3 Assumptions and Definitions	20
3.1 GAINFUL NON-PROFITABLE SEMIJOINS	27
Chapter 4 The Heuristics	30
4.1 APERS-HEVNER-YAO (AHY) ALGORITHMS	30
4.1.1 Algorithm SERIAL	31
4.1.2 Algorithm GENERAL	32
4.1.3 Complexity Analysis of Algorithm GENERAL (total time)	35

4.2	ALGORITHM W	35
4.2.1	Cost-effectiveness Analysis of Algorithm W	37
4.2.2	Complexity Analysis of Algorithm W	40
4.3	A COMPARATIVE EXAMPLE	41
4.4	ALGORITHM DW	51
4.4.1	Description of Algorithm DW	52
4.4.2	Outline of Algorithm DW	53
Chapter 5	Evaluation	55
5.1	METHODOLOGY	55
5.1.1	The Test Queries	56
5.1.2	The Test Database	57
5.2	EXPERIMENTAL RESULTS	59
5.2.1	Relevance of Results	60
5.3	CONCLUSIONS	61
5.3.1	Algorithm W versus AHY (total cost)	61
5.3.2	Algorithm W versus Algorithm DW	65
Chapter 6	Conclusions and Future Work	70
6.1	FUTURE WORK	71
	Selected Bibliography	73
	Appendix A: Connectivity	76
	Appendix B: WWW Availability	78
	Appendix C: Result Summaries	80
	Vita Auctoris	89

List of Figures

Figure 2.1	Sample query graph.	9
Figure 2.2	The possible negative reduction effect of a relational join.	12
Figure 2.3	Illustration of semijoins.	13
Figure 5.1	W – AHY cost comparisons for 100% connectivity. . .	62
Figure 5.2	W – AHY cost comparisons for 75% connectivity. . . .	63
Figure 5.3	W – AHY cost comparisons for 50% connectivity. . . .	64
Figure 5.4	W – DW cost comparisons for 100% connectivity. . . .	67
Figure 5.5	W – DW cost comparisons for 75% connectivity.	68
Figure 5.6	W – DW cost comparisons for 50% connectivity.	69

List of Tables

Table 3.1	A statistical representation of a query.	27
Table 4.1	Query statistics for the comparative example.	42
Table 5.1	Descriptions of individual runs.	60
Table 5.2	Statistical relevance of differences between Algorithm DW and Algorithm W.	61
Table C.1	Uniform distribution with approx. 100% connectivity. . .	81
Table C.2	Random distribution with approx. 100% connectivity. . .	82
Table C.3	Uniform distribution with approx. 75% connectivity. . .	83
Table C.4	Random distribution with approx. 75% connectivity. . .	84
Table C.5	Uniform distribution with approx. 50% connectivity. . .	85
Table C.6	Random distribution with approx. 50% connectivity. . .	86
Table C.7	Percentage increase in the response time for Algorithm DW over W.	87
Table C.8	Overhead as a percentage of the total cost in Algorithm DW.	88

Chapter 1

Introduction

With the proliferation and continued advancement of telecommunication technology, it is not surprising to see the development of decentralized information systems. Distributed Database Management Systems have certain advantages over traditional Centralized Database Management Systems in that they achieve the advantages of performance, reliability, availability and modularity that are inherent in distributed systems [CP84, OV91, Teo92]. By definition, a distributed database is a collection of multiple, logically interrelated databases distributed over a computer network. Each site within the network consists of an autonomous database capable of processing local applications as well as distributed applications which require access to data from several different sites via a communication network [CP84, OV91].

The use of a relational query enables users to specify a description of the required data without having to know the physical location of the data. The retrieval of data from various sites in a distributed database is referred to as distributed query processing. It is evident that the performance of a Distributed

Database Management System is critically dependant upon the capability of the query optimizer to derive efficient query processing strategies [ES80]. In addition to increasing the response time for a query, the use of an ineffective processing strategy may cause performance deterioration over the entire distributed database [OS88].

The most common approach to distributed query optimization has been the use of a static three phased approach that utilizes either join or semi-join operators as the primary reducing operator. Of those algorithms employing semi-joins as the primary reducing operator, the algorithms developed by Apers, Hevner and Yao [AHY83] are regarded as the best heuristics to date. Other less common methods are based on the use of improvement algorithms [CL84], the pipelining of the semi-join process [RK91], adaptive selection of execution strategies [TI90] as well as the use of one-shot fixed precision semi-joins [WLC91]. For specific types of distributed databases, the use of specialized semi-joins have been proposed as the primary reducing operator [PC90, CL90].

Clearly, in the case of static algorithms, any realistic query optimization hinges on the accurate estimation of the cardinality of intermediate results and final results of a query [Gra89, Loh89]. It is argued in [Loh89] that the two major assumptions (the uniform distribution of data and attribute independence) in the probabilistic models used by nearly every optimizer are flawed, resulting in estimation errors.

In addition, little work has been done in the validation of the optimization algorithms [Sel89]. In most cases algorithms are compared in theory without

considering the effect of using real data. Previous work in benchmarking database systems [BDT83], suggests that it should not be unreasonable to validate an algorithm's performance against a realistic database.

While it is commonly agreed upon that real world data does not conform to the uniform distribution and attribute independence assumptions, to our knowledge no one has examined whether or not these assumptions have an effect on the performance of distributed query processing. Clearly, validation would answer this question. In addition, it would also provide insight into the accuracy of the estimation techniques.

With respect to the development of new heuristics, citations indicate that the Apers-Hevner-Yao (AHY) Algorithms are considered to be the best heuristics proposed to handle general queries. Heuristics proposed after the AHY algorithms are typically designed around specific hybrid operators, architectures, network topologies, etc. Are the AHY algorithms the best that can be achieved ?

In this thesis the following questions are examined:

- Can improvements be made in semi-join based query optimization heuristics ?
- Are the assumptions of uniform data distribution and attribute independence valid for real world data ?
- Will the use of current information available to a dynamic heuristic provide better performance than its static counterpart ?

To address these questions a benchmark database has been developed on which

the performance of the AHY General (total time) algorithm will be compared with two proposed heuristics namely, Algorithms W (static) and DW (dynamic).

1.1 THESIS ORGANIZATION

This work is organized into six chapters with chapter 1 constituting the introduction to this thesis. In chapter 2 the relevant background material for this thesis is reviewed. This review includes discussions on the goal of distributed query processing, estimation techniques as well as static and dynamic query processing strategies.

In Chapter 3 all of the notations and definitions that are used throughout this work are presented. In particular, theorems, proofs and lemmas for numerous concepts employed by the proposed heuristics are presented.

Chapter 4 presents detailed descriptions of the three heuristics. A comparative example between Algorithm W and AHY is provided to clarify how each heuristic is executed. This example also serves to illuminate the differences that exist between the two heuristics.

In Chapter 5 the evaluation methodology is discussed. Particular attention is paid to the design of the benchmark database and the queries with which the heuristics are evaluated with. The remainder of this chapter is used to present the experimental results along with a discussion of the conclusions that can be inferred from the results.

Lastly, chapter 6 provides a summary of the conclusions attained from the work that this thesis represents along with some plans for future work.

In this chapter background information on the distributed query optimization problem is presented. In Section 1, the overall objectives of distributed query optimization are examined. Section 2, details the use of semi-joins as the primary reducing operation. Lastly, in Sections 3 and 4 respectively we present and discuss the use of static and dynamic strategies for distributed query optimization are discussed.

2.1 QUERY OPTIMIZATION OBJECTIVES

Most research with the exception of [ESW78] has concentrated on the optimization of a particular class of queries commonly known as Select-Project-Join queries¹ [CP84, OV91]. The formulation of an optimized execution strategy is based on the minimization of some objective cost function or functions. Some commonly used objective functions [Bod85, BR88b] are the:

- dollar cost due to network usage

¹ Also referred to as conjunctive normal form queries.

- dollar cost due to local CPU usage
- combined dollar cost of both local CPU processing and network usage
- delay due to local CPU processing
- delay due to network data transfers
- combined delay due to local CPU processing and network data transfers
- volume of data processed by all information processors
- volume of data transferred over the network
- total size of partial results²

Central to most optimization strategies is the use of either a *total cost model* or *response time model*. In the total cost model the objective is to minimize the overall costs that are incurred in processing the query. Most approaches assume the total cost to be the amount of data transferred. Due to the nature of distributed databases, it is possible that some or all of the processing required for a given query can be executed in parallel. The response time model is based on this supposition; seeking to minimize the elapsed time for query execution.

2.2 ESTIMATION

The fundamental goal of an optimization algorithm is the formulation of a query execution strategy that is optimal. Unfortunately, the formulation of an optimal solution can only be accomplished by performing an exhaustive search of all possible execution strategies. The complexity of such an enumeration has been

² A partial result refers to the size of a relation after the application of a relational operation.

shown to be NP-hard [WC93], making any such algorithms too computationally expensive to implement. Hence, heuristic algorithms are employed to quickly formulate near-optimal strategies.

The performance of static query optimization algorithms is heavily dependant upon the estimation technique used to evaluate the sizes of partial results. Some of the commonly used algorithms are as follows.

To estimate the expected number of tuples in a relation resulting from an arbitrary number of join operations, Chen and Yu [CY90] propose the following theorem.

Theorem: Let $G = (V, E)$ be a join query graph and $G_R = (V_R, E_R)$ is a connected subgraph of G . Let R_1, R_2, \dots, R_p be the relations corresponding to nodes V_R and A_1, A_2, \dots, A_q be the distinct attributes associated with edges in E_R .

Let n_i be the number of different nodes (relations) that edges with attribute A_i are incident to. Suppose R^* is the relation resulting from the join operations between R_1, R_2, \dots, R_p and m is the expected number of tuples in R^* . Then

$$m = \frac{\prod_{i=1}^p |R_i|}{\prod_{j=1}^q |A_j|^{n_j-1}}$$

For the query graph illustrated in Figure 2.1, the expected number of tuples in the resulting relation is estimated as

$$\frac{\prod_{i=1}^4 |R_i|}{|A||B|^3|C||D|}$$

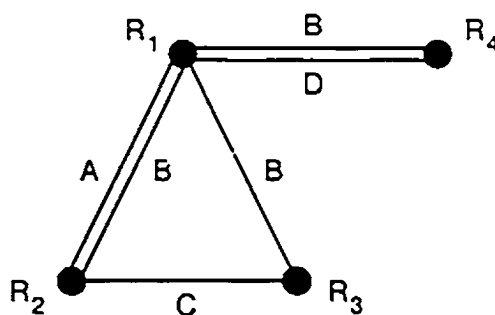


Figure 2.1 Sample query graph.

Estimating the size of partial results after the application of a semi-join operation is somewhat different than that of the join operation. The two most common algorithms, outlined in [Yu85] are as follows.

Suppose that we have two single attribute relations, R_1 and R_2 . Suppose further that the values of the common join-attribute, say A are uniformly and independently distributed on both relations. If the semi-join $R_2 \bowtie_{A} R_1$ is executed, the size of R'_1 can be estimated as $S(R_1) \times \rho_{2a}$, where ρ_{2a} is the selectivity of attribute A of relation R_2 . The selectivity of the reduced attribute A of relation R_1 is estimated by $\rho'_{1a} = \rho_{1a} \times \rho_{2a}$.

If the reducing relation is the result of a sequence of semi-joins, the incoming selectivity for this schedule is the product of the selectivities of all of the attributes in the schedule. The only restriction is that if there are multiple occurrences of an attribute in the schedule then only one instance of its selectivity is used.

Estimation is somewhat different when dealing with multi-attribute relations. Suppose R_2 is defined as above but R_1 is now a relation with two join-attributes A and B . After the application of the semi-join $R_2 \bowtie_{A} R_1$, the cardinality of R_1

is easily estimated as $|R_1| \times \rho_{2a}$. R_1 may subsequently be used to reduce some other relation, say R_3 , by performing the semi-join $R_1 \bowtie_B R_3$ over attribute B . To obtain an accurate estimate of the size of R_3 after the semi-join it is necessary to estimate the cardinality of attribute B of R_1 after the execution of $R_2 \bowtie_A R_1$. Yu [Yu85] shows that this estimation problem is related to the problem: "Given n balls with m different colours. What is the expected number of colours if t balls are randomly selected from the n balls". In the semi-join problem the correspondences are: n balls being the cardinality of R_1 prior to reduction; m colours being the cardinality of the B values in R_1 ; the t selected balls correspond to the cardinality of R_1 after the application of the semi-join. The expected number of colours of the t selected balls is

$$m \times \left[1 - \prod_{i=1}^t \left(\frac{n((m-1/m)) - i + 1}{n - i + 1} \right) \right]$$

It is important to note that while t is a parameter in the ball-colour problem, the cardinality of R_1 after the application of the semi-join needs to be estimated. For this reason the formula, if evaluated in its present form is expensive computationally and may cause overflow or underflow for large values of t . The following function presented in [BGW⁺81, Yu85] provides an estimation to the above formula after t has been estimated.

$$\begin{cases} m, & \text{if } t > 2m \\ \frac{(t+m)}{3}, & \text{if } 2m > t > m/2 \\ t, & \text{if } (m/2 > t) \end{cases}$$

The following example illustrates the use of this estimation formula.

Example: Suppose that R_1 and R_2 are the same as above and R_3 is a single attribute relation with attribute B . If R_1 is reduced by R_2 and R_3 using semi-joins,

$R_2 \bowtie_A R_1 \bowtie_B R_3$, the cardinality of R_1 can be estimated as $|R_1| \times \rho_1 \times \rho_2$, where ρ_3 is the selectivity of R_3 under attribute B . Because attributes A and B are independent of each other, the expected number of distinct values of A and B are estimated using the approximation formula described above.

2.3 SEMI-JOINS AS REDUCERS

Initially, distributed query optimization focused on the use of the relational join as the primary reducing operation. While simplistic in its execution, there exists the possibility that the size of the relation resulting from a join may exceed the sizes of the relations that participated in the join. This particular condition is illustrated in Figure 2.2.

In contrast, the semi-join operation is guaranteed to monotonically reduce the size of a relation, with the worst case being no reduction. In addition, the properties of semi-joins permit their computation with less intersite data transfers than for joins. If required, the reduction effect of a relational join may be obtained through the application of one semi-join and one join as defined by the equivalence relation

$$R_i \bowtie R_j \equiv (R_i \bowtie R_j; R'_i \bowtie R_i)$$

Based on the properties of semi-joins, the number of tuples in the result of the application of a semi-join, say $R_2 \bowtie R_1$ will be in the range $1 \leq |R'_1| \leq |R_1|$. Variance in the reduction effect of a semi-join is illustrated in the examples presented in Figure 2.3. Because the reduction effects of a semi-join

R_1	
A	B
a1	b1
a2	b1
a2	b2
a3	b2
a4	b2

R_2	
B	C
b1	c3
b1	c2
b2	c4
b3	c1
b4	c5

$R_1 \bowtie R_2$		
A	B	C
a1	b1	c3
a1	b1	c2
a2	b1	c3
a2	b1	c2
a2	b2	c4
a3	b2	c4
a4	b2	c4

Figure 2.2 The possible negative reduction effect of a relational join.

are asymmetric³, it is necessary to consider both applications of the semi-join in order to determine which application produces the greatest reduction. Clearly the use of a semi-join such that $S(R_i) \simeq S(R'_i)$ (as illustrated in Figure 2.3(b)), will not be *cost effective*. In this respect, optimization strategies based on semi-joins consider the use of *beneficial semi-joins* only.

A beneficial semi-join refers to a semi-join in which the benefit of performing the semi-join exceeds the cost of executing it. In practise the benefit is considered

³ The semi-join $R_i \bowtie R_j$ is not equivalent to the semi-join $R_j \bowtie R_i$.

R_1		R_2	
A	B	B	C
a1	b1	b1	c1
a2	b1	b2	c2
a2	b3	b5	c1
a2	b4	b5	c2
a3	b5	b5	c3
		b5	c4
		b5	c5

$R_2 \bowtie_b R_1$		$R_1 \bowtie_b R_2$	
A	B	B	C
a1	b1	b1	c1
a2	b1	b5	c1
a3	b5	b5	c2
		b5	c3
		b5	c4
		b5	c5

(a)
(b)

Figure 2.3 Illustration of semijoins.

to be the data that the semi-join eliminates. Benefit is formally defined as

$$\begin{aligned}
 B(R_i \bowtie_{d_{ia}} R_j) &= S(R_j) - S(R_j) \times \rho(d_{ia}) \\
 &= S(R_j) \times (1 - \rho(d_{ia}))
 \end{aligned}$$

where $\rho(d_{ia})$ is the selectivity of the attribute of R_i that is used to reduce R_j .

The cost associated with a semi-join refers to the cost of projecting the joining attribute from the reducing relation and transmitting it to the relation to be reduced. In general (with the exception of [VV84, HWY85, CL87, YGC88,

PLH89, AM91]), the cost of projecting the reducing attribute is considered to be negligible in comparison to the transmission cost. Assuming a fixed transmission cost between sites, the cost of a semi-join is computed using the following function:

$$C(R_i \bowtie_a R_j) = C_0 + C_1 \times S(d_{ia})$$

where the coefficients C_0 and C_1 are fixed constants representing the start-up cost for a transmission and the fixed cost per unit of data transmitted respectively. $S(d_{ia})$ represents the size of the projected attribute d_{ia} .

2.4 STATIC STRATEGIES

As previously stated the distributed query optimization is an NP-hard problem [WC93]. Hence, numerous heuristic algorithms have been proposed for constructing “near optimal” query execution schedules [BGW⁺81, AHY83, Yu85, KR87]. The term “near optimal” is used loosely in the sense that measuring the performance of a particular heuristic requires the optimal execution schedule to be known. In [Bod85] an A* tree is used to determine the optimal execution schedules for 30 different queries. Clearly if thousands of queries are being tested, it is unrealistic to attempt to compute the optimal solution for each query. For this reason the performance of a heuristic algorithm is typically described in terms of an improvement made over another existing algorithm. As a result, it is not evidently clear how “close to optimal” a solution for a particular strategy may be.

2.4.1 Two and Three Phased Approaches

The traditional approach to distributed query optimization has been the use of a *three phased approach* [BGW⁺81, AHY83, Yu85, KR87], consisting of the following three phases:

Phase 1 *Initial local processing.* Tuples and attributes which are irrelevant with respect to the query are filtered out by appropriate selection, projection, and join operations at the local site prior to any data transmission. This has the effect of reducing the amount of data transmitted over the network.

Phase 2 *Semi-join preprocessing.* After local processing, semi-joins are used to further reduce the size of relations. Based on the equi-join clauses in the query's qualification, a semi-join schedule (or sequence) is constructed. This schedule is subsequently used for the semi-join preprocessing.

Phase 3 *Final processing.* The resulting relations after the semi-join preprocessing phase are transmitted to the assembly site (usually the query site) where all of the relations are joined to form the result of the query. If it is the case that the assembly site is not the query site, the final result must be subsequently transmitted to the query site.

While not explicitly stated, if redundant relations are permitted, the identification of the required sites is performed within the local processing phase.

Alternatively, there are a few heuristic algorithms that are based upon a two

phased approach consisting of the following phases:

Phase 1 Determine the sequence of relational operations which minimizes the total size of the partial results.

Phase 2 Apply a polynomial time algorithm to find the optimal network site locations for executing the sequence of relational operations.

The rationale behind the two phased approach is that it essentially decomposes the optimization problem into two easier problems which may be solved more efficiently. Results in [BR88a] indicate that this approach yields beneficial results when both CPU and data transmission costs are incorporated into the objective cost function.

One of the first optimization algorithms (based on the use of semi-joins) to be proposed and implemented was the SDD-1 optimization algorithm, developed for the SDD-1 distributed database system [BGW⁺81]. Designed under the assumption that the transmission of data was the slowest component in query processing, the objective of the algorithm was to process a query with a minimum amount of intersite data transfers. It is important to point out that a reduction in the amount of intersite data transfers has the additional advantage of reducing the network load.

Being essentially an iterative hill-climbing algorithm, the SDD-1 algorithm always selects the most profitable reduction that is immediately at hand. The major disadvantage of this approach lies in its inability to backtrack and consider other execution strategies which may produce better solutions.

Another set of heuristics, proposed by Apers, Hevner and Yao [AHY83] was developed to handle simple as well as general types of queries. It is evident from citations in numerous articles that the Apers-Hevner-Yao (AHY) algorithms are considered to be milestones within the field of distributed query optimization.

Using the three phased framework that is common to many heuristics, a new static heuristic, Algorithm W is proposed. Using semi-joins and the concept of marginal profit, Algorithm W attempts to minimize the overall total cost of executing a query. A complete description of Algorithm W is presented in chapter 4.

2.5 DYNAMIC STRATEGIES

Static query optimizers rely heavily on various techniques for estimating the sizes of partial results, selectivities and other parameters pertaining to the distributed environment. It is recognized that a strategy based on inaccurate estimations may be far from optimal [ES80]. Any estimation errors in static strategies will be propagated and compounded during the execution of the schedule. Two alternative approaches exist for avoiding this problem.

The first approach relies on various dynamic query execution techniques to alleviate the problem. A dynamic query execution has the advantage that a strategy may be modified if it is found that it is not proceeding as planned. To determine whether a strategy is proceeding as planned requires information regarding the progress of the strategy to be gathered by one or more processors. The collection

of information on the current progress of a strategy is commonly referred to as *monitoring*. Based on this monitoring there is some decision making process which decides whether the current strategy being executed should be aborted and a new strategy proposed for the portion of the strategy that "has yet to be processed". If the current strategy is to be aborted then some form of corrective action will be needed to form and initiate a new strategy in order to complete the query. Various methods of monitoring and corrective action are discussed in [BRJ89]

Irrespective of the method of monitoring, the decision to correct can be made using methods based on either frequent *reformulation* or the use of preestablished threshold values. A description of both approaches is outlined below.

Reformulation. Whenever a new partial result is formed, the unexecuted portion of the query is reformulated using the most up-to-date information available. A correction is appropriate if the new reformulation has a lower cost than that of the current strategy.

Threshold. When a strategy is formulated, additional information is included to support the decision making process. For each parameter⁴ used in the formulation, two threshold values, V_{low} and V_{high} are constructed. A strategy is corrected if the actual value of a parameter falls outside the range of the associated threshold values.

⁴ For total cost estimation the parameter is the size of partial results.

In any given query there exist some partial results whose estimates are more critical than others. A threshold method proposed in [BR88b] deals with this problem through the use of a *Critical Path Network*. In this situation, threshold values are only constructed for partial results that are considered to be critical to the overall execution of the query. If any critical threshold value is exceeded, it is known that the strategy will be delayed and should be subsequently aborted and corrected.

The second approach is to provide more accurate estimates. However, it is noted that applications supporting the computation of accurate estimates are typically expensive in terms of size and upkeep of the required statistical information [BRJ89].

In this thesis a purely dynamic heuristic called Algorithm DW is proposed, which computes the execution strategy for a query on the fly using up to date information on the participating relations. A dynamic execution of this form does not require any schedule monitoring. In addition, it is believed that the overhead associated with maintaining up to date information will not constitute a significant portion of the overall total cost. A detailed description of Algorithm DW is given in Chapter 4.

Chapter 3

Assumptions and Definitions

For this thesis a distributed database management system is considered to be a collection of independent databases connected via a point-to-point network. Queries executed by the distributed database management system are taken from the select-project-join (SPJ) class of queries⁵. Each relation is located at a different site and has at least one attribute, other than any join attributes, that is required at the query site. For each relation it is assumed that the attribute values are uniformly distributed and that attributes are independent of one another. The cost of executing a query is considered to be a linear function relative to the total amount of data that is transferred across the network. It is also assumed that the local processing costs are negligible with respect to the data transmission costs.

As semijoins are the basis for all of the algorithms discussed in this thesis, this operation is outlined first. Suppose a query requires two relations say R_1 and R_2 to be joined, that is execute $R_1 \bowtie R_2$. A straight forward approach is to ship both relations to the query site and perform the join there. Alternatively, semijoins

⁵ Alternately referred to as conjunctive normal form queries.

may be used to reduce one or both of the relations prior to being shipped to the query site. A semi-join from relation R_1 to R_2 , denoted $R_1 \bowtie R_2$, is executed in the following manner:

1. Project R_1 over the common join attribute to get $R_1[j]$.
2. Ship $R_1[j]$ to the site of R_2 .
3. Execute $R_1[j] \bowtie R_2$.

Using the semi-join, the size of R_2 is reduced by eliminating those tuples which will not occur in the relation $R_1 \bowtie R_2$. A carefully chosen sequence of semi-joins can significantly reduce the sizes of the relations before they are shipped to the query site, thus reducing the total amount of data transferred across the network.

The following are defined for each relation R_i , $i = 1, 2, \dots, m$:

A_i	number of distinct attributes in relation R_i where $A_i > 1$.
$S(R_i)$	size (in bytes or any suitable measure) of R_i .
$ R_i $	the cardinality of relation R_i .

For each attribute⁶, d_{ij} , $j = 1, 2, \dots, A_i$ of R_i the following are defined:

$D(d_{ij})$	the domain of possible values for attribute d_{ij} .
$ D(d_{ij}) $	the cardinality of $D(d_{ij})$, that is the number of distinct values that make up the domain for d_{ij} .

⁶ The denotation d_{ij} refers to the j th join-attribute of relation R_i .

- $|d_{ij}|$ the cardinality of relation R_i projected over attribute d_{ij} , that is the number of distinct values in attribute d_{ij} .
- $S(d_{ij})$ size of attribute d_{ij} .
- $\rho(d_{ij})$ selectivity of attribute d_{ij} , where selectivity is defined as

$$\frac{|d_{ij}|}{|D(d_{ij})|}$$

With the execution of any semi-join there is some degree of overhead. The execution of the semi-join $d_{ij} \bowtie d_{kj}$ incurs a cost that is proportional to the amount of data (size of d_{ij}) that is transmitted from the site of R_i to the site of relation R_k . The cost is defined as

$$C(d_{ij} \bowtie d_{kj}) = C_0 + [C_1 \times S(d_{ij})]$$

where C_0 and C_1 are fixed constants. In all of the following definitions and examples in this thesis it is assumed that $C_0 = 0$ and $C_1 = 1$.

The benefit associated with the execution of a semi-join is equal the amount of data that will not be needed in the final result and hence does not need to be transmitted to the final result site. The benefit is defined as

$$\begin{aligned} B(d_{ij} \bowtie d_{kj}) &= S(R_k) - (S(R_k) \times \rho(d_{ij})) \\ &= S(R_k) \times (1 - \rho(d_{ij})) \end{aligned}$$

A semi-join is termed **profitable** if the benefit outweighs the cost. Profit is defined as

$$P(d_{ij} \bowtie d_{kj}) = B(d_{ij} \bowtie d_{kj}) - C(d_{ij} \bowtie d_{kj})$$

A **reducer** is defined as any attribute which can be used to reduce any other attribute (or relation). For the distributed query optimization, the identification of those reducers which are inexpensive to use and are good at reducing other relations (i.e. small attributes with high selectivities) is of key interest. Each of the proposed heuristics presented in this thesis attempts to construct reducers for each attribute in a cost effective manner. Each reducer is built using a sequence of semijoins $d_{aj} \bowtie d_{bj} \bowtie d_{cj} \bowtie \dots \bowtie d_{mj}$ such that $S(d_{aj}) \leq S(d_{bj}) \leq S(d_{cj}) \leq \dots \leq S(d_{mj})$. The final attribute to be reduced is considered to be the reducer and is denoted d_{mj}^* . To estimate the cost and benefit of using a reducer d_{mj}^* , some provision for estimating its selectivity is required. The selectivity of a reducer with respect to any relation occurring in its construction sequence is defined as

- 1 with respect to R_m , since d_{mj}^* has no reduction effect on the relation in which it is contained.
- the product of selectivities of all of the attributes which occur after d_{ij} in the sequence, since R_i has already been reduced by those attributes which precede it in the sequence.

In the case of relations which have a common-join attribute but do not appear in the construction sequence, the selectivity of the reducer with respect to the relation is simply the product of all of the selectivities of the attributes in the sequence. The selectivity of a reducer d_{mj}^* w.r.t. the relation R_i is formally

defined as

$$\rho(d_{mj}^*) = \begin{cases} 1. & i = m \\ \prod_{x=i+1}^m \rho(d_{xj}). & i < m \\ \prod_{x=u}^m \rho(d_{xj}). & \text{otherwise} \end{cases}$$

The cost associated with the application a reducer is the cost of transmitting it to the site of the relation that is to be reduced, which is simply $S(d_{mj}^*)$. To estimate the benefit of applying the reducer it is necessary to estimate the reduction effects the reducer will have on the relation if the reducer were to be used. The estimated size of R_i after the semi-join $d_{mj}^* \bowtie R_i$ is computed as

$$S'(R_i) \times \rho(d_{mj}^*)$$

where $S'(R_i)$ is the estimated size of R_i after all semi-joins preceding it in the sequence have been performed and $\rho(d_{mj}^*)$ is the estimated selectivity of the reducer w.r.t. R_i as defined above. The benefit of the semi-join $d_{mj}^* \bowtie R_i$ is subsequently defined as

$$B(d_{mj}^* \bowtie R_i) = S'(R_i) \times (1 - \rho(d_{mj}^*))$$

In some cases semi-joins may not be profitable, however their use may increase the profitability of subsequent semi-joins. These semi-joins are identified by examining their estimated **marginal profit**. For example, consider the semi-join $d_{xj}^* \bowtie d_{yj}$. Put simply, the marginal profit is the “extra” profit we acquire by using d_{yj}^* as the reducer rather than d_{xj}^* . For each relation that can be reduced using d_{xj}^* (and d_{yj}^*) there may exist some “extra” profit. The marginal profit is

therefore considered to be the sum of these “extra” profits. However, the following facts must be considered when computing the marginal profit:

- There is no profit in the semi-join $d_{yj}^* \times R_y$ since the attribute belongs to the relation.
- There is no profit in the semi-join $d_{yj}^* \times R_i$, if the cost outweighs the benefit of the semi-join.
- If the case where the semi-join $d_{xj}^* \times R_i$ is not profitable but $d_{yj}^* \times R_i$ is profitable, the marginal profit is simply the profit of the semi-join $d_{yj}^* \times R_i$.
- In the case of R_x we have $P(d_{xj}^* \times R_x) = 0$, therefore the marginal profit is $P(d_{yj}^* \times R_x)$.

For all other cases the marginal profit w.r.t. R_i is defined as

$$\begin{aligned} MP_{R_i}(d_{xj}^* \times d_{yj}) &= P(d_{yj}^* \times R_i) - P(d_{xj}^* \times R_i) \\ &= S(R_i) \times (\rho(d_{xj}^*) - \rho(d_{yj}^*)) + S(d_{xj}^*) - S(d_{yj}^*) \end{aligned}$$

The marginal profit with respect to R_i can be summarized as

$$MP_{R_i} = \begin{cases} 0, & \text{if } i = y \\ 0, & \text{if } P(d_{xj}^* \times R_i) \leq 0 \text{ and } P(d_{yj}^* \times R_i) \leq 0 \\ P(d_{yj}^* \times R_i), & \text{if } P(d_{xj}^* \times R_i) \leq 0 \\ P(d_{yj}^* \times R_i), & \text{if } i = x \\ P(d_{yj}^* \times R_i) - P(d_{xj}^* \times R_i), & \text{otherwise} \end{cases}$$

The total marginal profit is obtained by summing all positive marginal profits:

$$MP(d_{xj}^* \times d_{yj}) = \sum_{i=1}^m MP_{R_i}(d_{xj}^* \times d_{yj}) \text{ s.t. } MP_{R_i} > 0$$

The **gain** of a semi-join is defined to be the sum of its profit and marginal profit:

$$G(d_{xj}^* \bowtie d_{yj}) = P(d_{xj}^* \bowtie d_{yj}) + MP(d_{xj}^* \bowtie d_{yj})$$

A semi-join is **cost effective** if its gain is positive. In the proposed heuristics, the construction of reducers is based solely on the use of cost effective semi-joins. In the case where a semi-join is profitable however, there is no marginal profit, the semi-join should not be executed since it will be at least as profitable to use the reducer to reduce the relation instead.

Theorem 3.1. If, as part of the schedule to construct the reducer d_{mj}^* , there occurs a semi-join $d_{xj}^* \bowtie d_{yj}$ such that $P(d_{xj}^* \bowtie d_{yj}) > 0$ and $MP(d_{xj}^* \bowtie d_{yj}) = 0$ then it is at least as profitable to execute $d_{mj}^* \bowtie d_{yj}$ instead, where d_{mj}^* is the reducer.

Proof. When constructing the reducer d_{mj}^* the semi-joins are executed in the following order

$$d_{aj} \bowtie d_{bj} \bowtie d_{cj} \bowtie \cdots \bowtie d_{xj} \bowtie d_{yj} \cdots \bowtie d_{mj}$$

Therefore we have $S(d_{xj}^*) \geq S(d_{yj}^*)$ and $C(d_{xj}^* \bowtie d_{yj}) \geq C(d_{mj}^* \bowtie d_{yj})$. Because $S(d_{xj}^*) \geq S(d_{yj}^*) \Rightarrow \rho(d_{xj}^*) \geq \rho(d_{mj}^*)$ it is clear that $B(d_{xj}^* \bowtie d_{yj}) \leq B(d_{mj}^* \bowtie d_{yj})$.

$$\text{Therefore } P(d_{mj}^* \bowtie d_{yj}) \geq P(d_{xj}^* \bowtie d_{yj}). \quad \square$$

Corollary 3.1. Profitability is not a sufficient condition for performing a semi-join during the construction of a reducer.

Corollary 3.2. Semi-joins with no marginal profit should not be performed.

Corollary 3.3. A semi-join should be performed if the marginal profit exceeds the cost. For example, consider the semi-join $d_{xj}^* \bowtie d_{yj}$ which is part of the reducer construction sequence. If there exists relations R_i , $i \neq y$ such that

$$\sum_{i=a}^m S(R_i) \times (\rho(d_{xj}^*) - \rho(d_{yj}^*)) - S(d_{yj}^*) > 0$$

then $G(d_{xj}^* \bowtie d_{yj}^*) > 0$, therefore the semi-join should be performed.

3.1 GAINFUL NON-PROFITABLE SEMIJOINS

In this section it is shown that while a semi-join may not be immediately profitable, it may be gainful if the marginal profit is sufficiently large. Gainful semi-joins should therefore be executed because the overall goal is to maximize the reduction effect. In this particular case the reduction effect is due to the increased selectivity that is propagated from the non-profitable semi-join to later semi-joins. For example⁷, the data in Table 3.1 represents a query after all local

Relation	$S(R_i)$	$S(d_{ij})$	$\rho(d_{ij})$
R1	1000	500	0.5
R2	800	600	0.6
R3	3000	700	0.7
R4	5000	800	0.8

Table 3.1 A statistical representation of a query.

⁷ Note: for simplicity, in this example we assume that each tuple or attribute value constitutes one unit of data transmission cost.

processing has been carried out. All that remains is to join the relations and ship them to some (other) result site.

The semi-join $d_{11}^* \bowtie d_{12}$ is clearly not profitable since the cost is 500 units and the benefit is only 400 units. However, this semi-join is gainful because the marginal profit is substantial:

$$\begin{aligned} MP_{R_4}(d_{11}^* \bowtie d_{21}) &= S(R_4) \times (\rho(d_{11}^*) - \rho(d_{21}^*)) + S(d_{11}^*) - S(d_{21}^*) \\ &= (5000 \times 0.2) + 500 - 300 \\ &= 1200 \end{aligned}$$

Clearly, the marginal profit is greater than the cost of the semi-join, hence this is a gainful semi-join. Obviously, d_{21} should be used to reduce R_4 however, this fact needs to be identified when the semi-join $d_{11}^* \bowtie d_{12}$ is considered. The calculation of marginal profit and gain provides the information necessary to make an appropriate decision.

Lemma 3.1. The semi-join $d_{xj}^* \bowtie d_{yj}$ is gainful but not profitable if there exist relations R_i , $i \notin \{x, y\}$ such that

- a) $S(d_{xj}^*) > S(R_y) \times (1 - \rho(d_{xj}^*))$ and
- b) $\sum_{i=a}^m [S(R_i) \times (\rho(d_{xj}^*) - \rho(d_{yj}^*))] - S(d_{yj}^*) > 0$

Condition (a) follows from the definition of $P(d_{xj}^* \bowtie d_{yj})$. If condition (b) holds then there exist one or more relations R_i , $i \notin \{x, y\}$ for which

$$\sum_{i=a}^m MP_{R_i}(d_{xj}^* \bowtie d_{yj}) > C(d_{xj}^* \bowtie d_{yj})$$

This implies that $G(d_{xj}^* \times d_{yj}) > 0$ hence, the cost of the semi-join outweighs any immediate benefit but the marginal profit is greater than the cost indicating that it must be gainful.

Corollary 3.4. A necessary condition for adding a semi-join to the schedule for reducer construction is that the marginal profit must be greater than the cost of the semi-join.

Corollary 3.5. If a relation can be found where the marginal profit of the semi-join exceeds its cost, it is not necessary to examine an other relations.

In this chapter the details of the three heuristics implemented in this thesis namely, Algorithms AHY General (total time), W and DW are presented. For this thesis Algorithm AHY is used as a benchmark with upon which the performance of the proposed heuristics W and DW can be based. The reasoning behind this selection is twofold. First, the AHY algorithm is considered by many to be the best general query optimizer proposed. Secondly, there is extensive literature describing the execution of the heuristic.

In section 1 the details of the AHY algorithms are presented. In the following section the description of the proposed heuristic, Algorithm W is given. Section 3 presents a comparative example of the two heuristics to illustrate their respective use and relative differences. Lastly, the proposed dynamic heuristic, Algorithm W is described.

4.1 APERS-HEVNER-YAO (AHY) ALGORITHMS

A collect of algorithms for optimizing a special class of simple *queries*⁸

⁸ A simple query refers to a query that has only one common join attribute.

has been introduced and investigated in [AHY83]. These algorithms namely Algorithm SERIAL and Algorithm PARALLEL attempt to minimize the total time and response time respectively. In each algorithm, semi-joins are used to reduce the size of relations by deleting those tuples which will not play a role in the final join.

Extending Algorithms SERIAL and PARALLEL, Apers, Hevner and Yao present Algorithm GENERAL which is capable of optimizing *general* queries. A general query is characterized by relations which contain more than one common join attribute.

As the focus of this thesis is query optimization with respect to total cost, discussions on the AHY algorithms will be limited to Algorithm SERIAL and Algorithm GENERAL (Total Cost).

4.1.1 Algorithm SERIAL

Algorithm SERIAL works as follows:

Step 1: Order relations R_i such that $S(R_1) \leq S(R_2) \leq \dots \leq S(R_n)$.

Step 2: If no relations are at the result node, then select strategy

$$R_1 \rightarrow R_2 \rightarrow \dots \rightarrow R_n \rightarrow \text{result node}$$

or else if R_r is a relation at the result node, then there are two strategies;

a) $R_1 \rightarrow R_2 \rightarrow \dots \rightarrow R_r \rightarrow \dots \rightarrow R_n \rightarrow R_r$ or

b) $R_1 \rightarrow R_2 \rightarrow \dots \rightarrow R_{r-1} \rightarrow R_{r+1} \rightarrow \dots \rightarrow R_n \rightarrow R_r$

After studying the literature it appears as though Algorithm SERIAL initially ranks the relations in terms of size under the assumption that the cost of transmitting a relation is directly proportional to the size of the relation. If this is indeed the case, it is obvious that the use of a non-linear cost model in conjunction with Algorithm SERIAL would result in sub-optimal schedules.

4.1.2 Algorithm GENERAL

The overall strategy employed by Algorithm GENERAL is to decompose a general query into a collection of simple queries. These simple queries are subsequently processed using Algorithm SERIAL for total cost optimization. The resulting schedules are examined and integrated to form an optimized schedule representing the general query. A detailed description of Algorithm GENERAL is given as follows (summarized from [AHY83]):

Step 1: *Do all initial local processing.*

Step 2: *Generate candidate schedules.* Isolate each of the σ common join attributes, and consider each to define a *simple query* with an undefined result node. Apply Algorithm SERIAL to each simple query. This results in one schedule per simple query. From these schedules, the candidate schedules for each common join attribute are extracted. Consider the common join attribute d_{ij} . Its candidate schedule is identical to the schedule produced by Algorithm SERIAL, applied to the simple query in which d_{ij} occurs, up to the transmission of d_{ij} . All transmissions after that are deleted from the schedule.

Step 3: *Integrate the candidate schedules.* For each relation R_i , the candidate schedules are integrated to form a processing schedule for R_i . To minimize total cost, schedule integration is performed using either procedure **Total** or procedure **COLLECTIVE**. Outlines for procedures **TOTAL** and **COLLECTIVE** follow. It should be pointed out that procedure **TOTAL** does not consider the existence of redundant data transmissions in separate relation schedules. Therefore, strategies derived using procedure **TOTAL** may not be optimal.

Step 4: *Remove schedule redundancies.* Eliminate relation schedules for relations which have been transmitted in the schedule for another relation.

The following are outlines for procedures **TOTAL** and **COLLECTIVE**.

Procedure TOTAL

Step 1: *Adding candidate schedules.* For each relation R_i and each candidate schedule $CSCH_l$, perform the following. If a schedule contains a transmission of a joining attribute of R_i , say d_{ij} , then create another candidate schedule identical to $CSCH_l$ except that the transmission of d_{ij} is deleted.

Step 2: *Select the best candidate schedule* For each relation R_i and each common join attribute d_{ij} , $j = 1, 2, \dots, \sigma$, select the candidate schedule which minimizes the total time (cost) for transmitting R_i . Only joining attributes which can be joined with d_{ij} are considered. $BEST_{ij}$ denotes the best candidate schedule for relation R_i and joining attribute d_{ij} .

Step 3: Candidate schedule ordering. For each relation R_i , candidate schedules $BEST_{ij}$ are ordered on joining attributes d_{ij} , $j = 1, 2, \dots, \sigma$, so that

$$ART_{i1} + C(S(R_1) \times SLT_{i1}) \leq \dots \leq ART_{i\sigma} + C(S(R_i) \times SLT_{i\sigma})$$

Schedules involving joining attributes not in R_i are disregarded. ART_{ij} denotes the arrival time (cost) of the $BEST_{ij}$ schedule. SLT_{ij} denotes the accumulated selectivity of the $BEST_{ij}$ schedule into R_i .

Step 4: Schedule integration. For each $BEST_{ij}$ in ascending order of j , construct the integrated schedule to R_i which consists of the parallel transmissions of candidate schedule $BEST_{ij}$ and all schedules $BEST_{ik}$ where $k < j$. Select the integrated schedule that results in the minimum total time (cost) value

$$TOTT_i = \sum_{k=1}^j \left[ART_{ik} + C \left(S(R_i) \times \prod_{k=1}^j SLT_{ik} \right) \right].$$

Procedure COLLECTIVE

Step 1: Select candidate schedule. For each relation R_i and joining attribute d_{ij} , $j = 1, 2, \dots, \sigma$, select the minimum cost candidate schedule that contains the transmission of all components of attribute j with selectivities less than 1.

Step 2: Build processing strategy. For each relation R_i , define the schedule to be the parallel transmissions of all d_{ij} candidate schedules to R_i .

Step 3: Test strategy variations. Using a removal heuristic, construct new strategies by removing the most costly data transmission. The total time cost of the new strategy is compared with that of the old strategy, with the less costly

strategy being maintained. Testing continues until no further cost benefit can be obtained.

An illustrative example of AHY General (total time) is presented in section 4.3.

4.1.3 Complexity Analysis of Algorithm GENERAL (total time)

If it is assumed that a general query requires data from m relations, and that all m relations are joined on σ joining attributes. In step 2, Algorithm SERIAL is applied to each simple query. Because the joining attributes must be ordered by size the complexity is $O(\sigma m \log_2 m)$.

The complexity of procedure TOTAL is $O(\sigma m^2)$. In step 1, no more than $O(\sigma m)$ candidate schedules are added. For each relation the procedure must subsequently determine the $BEST_{ij}$ schedule among the $O(\sigma m)$ candidate schedules. Hence, the complexity of step 2 is $O(\sigma m^2)$. Therefore, the complexity for an arbitrary general distributed query, Algorithm GENERAL (total time) has a processing complexity no worse than $O(\sigma m^2)$.

4.2 ALGORITHM W

In this section a proposed static heuristic (Algorithm W) that attempts to minimize total cost is described. The algorithm is characterized by two distinct phases; first, semi-join schedules for constructing each reducer are formed using a cost/benefit analysis which is based on the estimated attribute selectivities and the

sizes of partial results. In the second phase the schedule is executed. A detailed description of Algorithm W follows.

Step 1: *Determine schedules for the construction of reducers.* For each join-attribute j , establish a schedule for the construction of reducer d_{mj}^* ⁹.

a) Order the attributes by increasing size such that

$$S(d_{aj}) \leq S(d_{bj}) \leq \dots \leq S(d_{mj})$$

b) Next, evaluate the semi-joins in order beginning with $d_{aj} \bowtie d_{bj}$. The semi-join is appended to the schedule for constructing the reducer if

i. It is both profitable and marginally profitable. In other words

$$P(d_{aj} \bowtie d_{bj}) > 0 \text{ and } MP(d_{aj}^* \bowtie d_{bj}) > 0 \text{ or}$$

ii. It is not profitable but is gainful. That is, we have $P(d_{aj} \bowtie d_{bj}) \leq 0$ but

$$G(d_{aj}^* \bowtie d_{bj}) > 0.$$

If the semi-join is appended the next semi-join for consideration is $d_{bj}^* \bowtie d_{cj}$ otherwise $d_{aj}^* \bowtie d_{cj}$ is considered. This process is repeated until all attributes have been considered. The final attribute to be reduced is the reducer.

Step 2: *Reducer selection and application review of unused semi-joins.* In this step the reduction effects of the construction and use of each reducer on all applicable relations are considered. That is,

a) The reducers are ordered by increasing size.

⁹ It is important to note that each schedule is constructed independently and no semi-joins are actually executed in this step.

- b) For each reducer in turn, estimate the reduction effects of constructing and applying it. Profitable semi-joins are appended to the final schedule.

Step 3: Review of unused semi-joins. After Step 2 it may be the case that a number of reducers have not been used as they are not profitable. In this case, check to see if there are any remaining profitable semi-joins for that particular join-attribute. This reevaluation process is carried out as follows:

- a) sort the attributes by increasing size
 b) evaluate each semi-join in turn, appending profitable semi-joins to the final schedule. Note, the marginal profit is not considered in this step.

Step 4: Execute the schedule. In this step the reducers are constructed and shipped to the designated sites to be used as reducers. The reduced relations are subsequently shipped to the query site where the answer is assembled.

An illustrative example of Algorithm W is given in section 4.3.

4.2.1 Cost-effectiveness Analysis of Algorithm W

The term “cost-effective” in this sense refers to the construction of good reducers with a minimum cost overhead associated with their construction. Under this context, the heuristic constructs and applies the reducers in the most cost-effective manner.

Theorem 4.1. Given relations R_1, R_2, \dots, R_m ordered such that

$$S(R_1[j]) \leq S(R_2[j]) \leq \dots \leq S(R_m[j])$$

then the sequence of semi-joins $((d_{1j} \bowtie d_{2j}) \bowtie d_{3j}) \dots \bowtie d_{mj}$ constructs a reducer (for the join-attribute j) with minimum cost. (Minimum in the sense that the data transferred between nodes to construct the reducer is kept to a minimum.)

Proof. Assume that the data is distributed such that after the application of the semi-join $d_{ij} \bowtie d_{kj}$ we have $S(d_{kj}) = S(d_{kj}) \times \rho(d_{ij})$, where $\rho(d_{ij})$ is the selectivity of the attribute d_{ij} .

Consider any x and y in the sequence above. By definition, we have $S(d_{xj}) \leq S(d_{yj})$ which implies $\rho(d_{xy}) \leq \rho(d_{yj})$. If we switch the order of x and y in the sequence we will incur an increase in the total cost, $Cost_I$ where

$$Cost_I = S(d_{yi}) - \rho(d_{xi}) \times S(d_{yi})$$

as we no longer have the reduction effects of d_{xj} . In addition, there is a decrease in the total cost,

$$Cost_D = S(d_{xj}) - \rho(d_{yj}) \times S(d_{xj})$$

since we now have the reduction effects of d_{yj} .

In the pathological case were $S(d_{xj}) = S(d_{yj})$, by definition $\rho(d_{xj}) = \rho(d_{yj})$. By inspection it is clear that $Cost_I = Cost_D$, hence the sequence is minimal in any case. For the remaining cases we want to prove that $Cost_I > Cost_D$. Let $S(d_{xj}) < S(d_{yj})$, by definition $\rho(d_{xj}) < \rho(d_{yj})$. Substituting for $Cost_I$ and $Cost_D$ gives

$$S(d_{yj}) - \rho(d_{xj}) \times S(d_{yj}) > S(d_{xj}) - \rho(d_{yj}) \times S(d_{xj})$$

Simplification reduces this equation to

$$\frac{S(d_{yj})}{S(d_{xj})} > \frac{1 - \rho(d_{yj})}{1 - \rho(d_{xj})}$$

By definition, $S(d_{xj}) < S(d_{yj})$ therefore,

$$\frac{S(d_{yj})}{S(d_{xj})} > 1$$

Similarly, by definition $\rho(d_{xj}) < \rho(d_{yj})$ therefore,

$$1 < \frac{1 - \rho(d_{yj})}{1 - \rho(d_{xj})}$$

Hence, $Cost_I > Cost_D$. Therefore, except for the noted case, if the order of any two semi-joins is swapped there is an increase in the total cost of constructing the reducer. Therefore, the sequence

$$((d_{1j} \bowtie d_{2j}) \bowtie d_{3j}) \dots \bowtie d_{mj}$$

constructs the reducer with minimal cost. \square

Theorem 4.2. Given reducers $d_{xa}^*, d_{xb}^*, \dots, d_{xn}^*$ ordered such that

$$S(d_{xa}^*) \leq S(d_{xb}^*) \leq \dots \leq S(d_{xn}^*)$$

then the cost of utilizing the reducers is minimized if they are applied in this order.

Proof. By definition the selectivity is assumed to be proportional to size. Consider using any two reducers d_{ix}^* and d_{iy}^* on any relation R_z where $S(d_{ix}^*) \leq S(d_{iy}^*)$ and $i = z$.

The following two cases must be considered:

- (Case 1) The semi-join $d_{i_x}^* \bowtie R_z$ is executed first. Then execute the semi-join $d_{i_y}^* \bowtie R_z$ only if it is profitable. The cost will therefore be either $S(d_{i_x}^*) + S(d_{i_y}^*)$ or $S(d_{i_x}^*)$.
- (Case 2) The semi-join $d_{i_y}^* \bowtie R_z$ is executed first. Then execute the semi-join $d_{i_x}^* \bowtie R_z$ only if it is profitable. The cost will therefore be either $S(d_{i_y}^*) + S(d_{i_x}^*)$ or $S(d_{i_y}^*)$.

The cost of case 1 will always be less than or equal to the cost of case 2 (and the benefit of case 1 will always be greater than or equal to the benefit of case 2). Clearly the cost is minimized if the reducers are utilized in order of increasing size. \square

4.2.2 Complexity Analysis of Algorithm W

Algorithm W as outlined produces cost-effective schedules in an efficient manner. Assume that a query requires the joining of m relations over n common-join attributes. In step 1a at most m attributes are sorted resulting in a complexity of $O(m \log m)$. This step is repeated for the n common-join attributes giving a complexity of $O(nm \log m)$.

In step 1b the profit and marginal profit are computed for $m - 1$ semi-joins. The calculation of profit is always $O(1)$ however, marginal profit is computed with respect to a variable number of other relations. In the worst case the marginal profit must be computed with respect to $m - 1$ relations, resulting in a complexity

of $O(m^2)$. Step 1b is repeated for each attribute, giving a best case complexity of $O(nm)$ and a worst case complexity of $O(nm^2)$.

In step 2a the n reducers are sorted, with complexity $O(n \log n)$. In step 2b the cost and benefit of at most $m-1$ are computed. This is repeated n times, resulting in a complexity of $O(nm)$.

In step 3 the construction sequences of unused reducers are reviewed. If α is the number of unused reducers to be reconsidered, the complexity for this step will be $O(\alpha m \log m)$. At worst case α will be $n-1$.

Thus, Algorithm W will have a best case complexity of $O(nm)$ and a worst case complexity of $O(nm^2)$. It is important to note that in most cases a positive marginal profit can be found with respect to a single relation. A positive marginal profit can usually be found by examining the largest relation not participating in the semi-join. Therefore, in most cases the expected complexity of Algorithm W is $O(nm)$.

4.3 A COMPARATIVE EXAMPLE

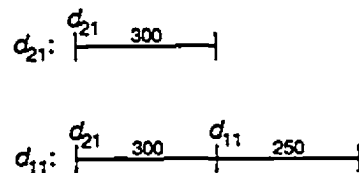
To illustrate and compare Algorithm W to Algorithm AHY General (total time), the execution of each algorithm on the sample query given in Table 4.1 are outlined in detail.

R_i	$S(R_i)$	d_{i1}		d_{i2}		d_{i3}	
		$S(d_{i1})$	$\rho(d_{i1})$	$S(d_{i2})$	$\rho(d_{i2})$	$S(d_{i3})$	$\rho(d_{i3})$
R_1	2000	500	0.83	800	0.53	600	0.60
R_2	4500	300	0.50	1000	0.67	—	—
R_3	6000	—	—	1400	0.93	800	0.80

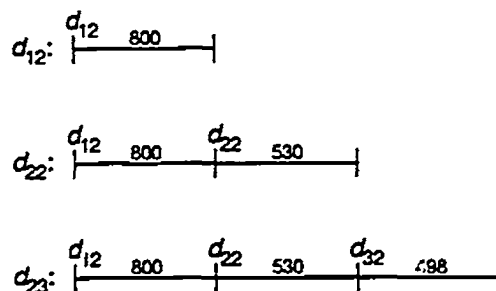
Table 4.1 Query statistics for the comparative example.

Applying Algorithm AHY General (total time) to the example, three simple queries are formed on attributes d_{i1} , d_{i2} and d_{i3} . In step 2 of Algorithm AHY General (total time), the following serial candidate schedules are formed.

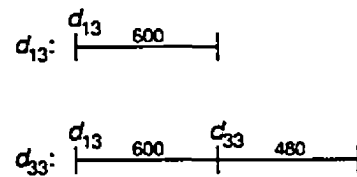
For d_{i1} ,



For d_{i2} ,

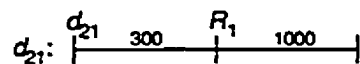


For d_{i3} ,



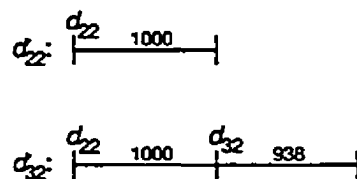
The construction of the schedule for R_1 will be discussed in detail. Each of the three attributes will be handled in turn.

Attribute d_{11} In step 1 of Procedure TOTAL, the simple serial schedules for d_{11} are examined to determine if any new schedules can be formed or if any schedules are not applicable to the current relation. For d_{11} the d_{11} schedule is not considered as it cannot be used to reduce itself. With only one candidate schedule to consider, schedule d_{12} is selected as the $BEST_{11}$ candidate schedule for d_{11} :



$$\begin{aligned} \text{Total time} &= C(300) + C(0.5 \times 2000) \\ &= 1300. \end{aligned}$$

Attribute d_{12} In step 2 of Procedure TOTAL, two schedules are added to the candidate schedules for attribute d_{i2} .



Each of the schedules for d_{12} are applied to R_1 . Obviously, the d_{12} schedule is not considered, leaving four schedules to be evaluated.

$$d_{22}: \begin{array}{c} d_{12} \quad 800 \quad d_{22} \quad 530 \quad R_1 \quad 1340 \\ |-----|-----|-----| \\ | \quad \quad \quad | \quad \quad \quad | \quad \quad \quad | \end{array}$$

$$\begin{aligned} \text{Total time} &= C(800) + C(0.53 \times 1000) + C(0.67 \times 2000) \\ &= 800 + 530 + 1340 \\ &= 2670. \end{aligned}$$

$$d_{32}: \begin{array}{c} d_{12} \quad 800 \quad d_{22} \quad 530 \quad d_{32} \quad 498 \quad R_1 \quad 1247 \\ |-----|-----|-----|-----| \\ | \quad \quad \quad | \quad \quad \quad | \quad \quad \quad | \quad \quad \quad | \end{array}$$

$$\begin{aligned} \text{Total time} &= C(800) + C(0.53 \times 1000) + C(0.36 \times 1400) \\ &\quad + C(0.62 \times 2000) \\ &= 800 + 530 + 498 + 1247 \\ &= 3075. \end{aligned}$$

$$d_{22}: \begin{array}{c} d_{22} \quad 1000 \quad R_1 \quad 1340 \\ |-----|-----| \\ | \quad \quad \quad | \quad \quad \quad | \end{array}$$

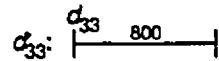
$$\begin{aligned} \text{Total time} &= C(1000) + C(0.67 \times 2000) \\ &= 1000 + 1340 \\ &= 2340. \end{aligned}$$

$$d_{32}: \begin{array}{c} d_{22} \quad 1000 \quad d_{32} \quad 938 \quad R_1 \quad 1247 \\ |-----|-----|-----| \\ | \quad \quad \quad | \quad \quad \quad | \quad \quad \quad | \end{array}$$

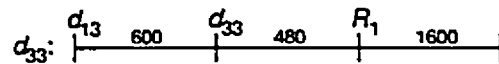
$$\begin{aligned} \text{Total time} &= C(1000) + C(0.67 \times 1400) + C(0.62 \times 2000) \\ &= 1000 + 938 + 1247 \\ &= 3185. \end{aligned}$$

Because the schedule d'_{22} has the smallest total time it is selected as the $BEST_{12}$ schedule.

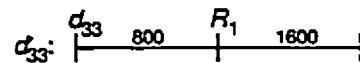
Attribute d_{13} In step 2 of Procedure TOTAL only one schedule is added to the above schedules for attribute d_{13} .



Each of the schedules with the exception of d_{13} are applied to R_1 .



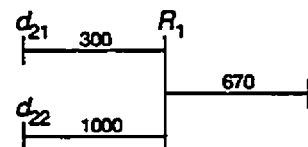
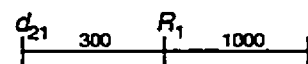
$$\begin{aligned} \text{Total time} &= C(600) + C(0.6 \times 800) + C(0.8 \times 2000) \\ &= 600 + 480 + 1600 \\ &= 2680. \end{aligned}$$

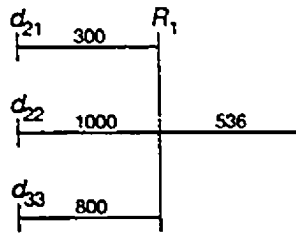


$$\begin{aligned} \text{Total time} &= C(800) + C(0.8 \times 2000) \\ &= 800 + 1600 \\ &= 2400. \end{aligned}$$

Because d_{33}^1 has the smallest total time, it is chosen as the $BEST_{13}$ schedule.

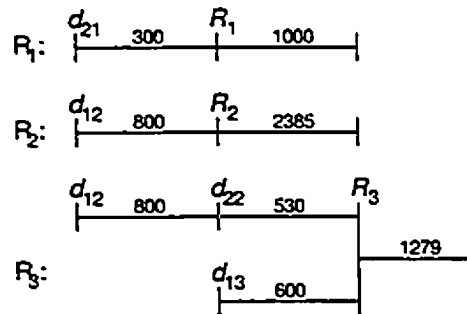
In Steps 3 and 4 of Procedure TOTAL, the $BEST_{1i}$ schedules are ordered by smallest total time and integrated to construct the following three schedules:





Since the first of these schedules has the smallest total time; it is chosen as the solution of Algorithm AHY General (total time) for R_1 .

The respective solutions for relations R_2 and R_3 are constructed in the same manner. The query processing strategy that is constructed by Algorithm AHY General (total time) for the example query is



The application of Algorithm W to the query is now examined. In step 1, the sequences for constructing reducers are determined. On inspection, it is clear that at most three reducer may be constructed: d_{i1} , d_{i2} and d_{i3} .

Reducer for d_{i1} The first semi-join considered is $d_{21} \times d_{11}$, where the cost is 300 units and the benefit is 700 units. The marginal profit of the semi-join

with respect to R_2 is computed as

$$\begin{aligned} MP_{R_3} &= (4500 \times 0.17) - 250 \\ &= 515. \end{aligned}$$

Since the profit and marginal profit are positive, the semi-join is added to the schedule for constructing the d_{i1} reducer. As there are no further semi-joins to consider, the reducer is therefore d_{11}^* . For this reducer the construction schedule is simply the semi-join $d_{21} \times d_{11}$.

Reducer for d_{i2} The first semi-join to consider is $d_{12} \times d_{22}$, where the cost is 800, the benefit is 2115 and the marginal profit with respect to R_3 is calculated as

$$\begin{aligned} MP_{R_3} &= (4500 \times 0.17) + 800 - 530 \\ &= 1035. \end{aligned}$$

This semi-join is added to the construction schedule for d_{i2} . Next we consider $d_{22}^* \times d_{32}$. The cost is 530, the benefit is 3869 and the marginal profit with respect to R_1 is computed as

$$\begin{aligned} MP_{R_1} &= (2000 \times 0.05) - 446 + 530 \\ &= 184. \end{aligned}$$

This semi-join is added so d_{32}^* is constructed by the semi-join sequence $d_{12} \times d_{22} \times d_{32}$.

Reducer for d_{i3} The only semi-join to be considered is $d_{13} \times d_{33}$. The cost is 600, the benefit 1800 and the marginal profit with respect to R_1 is calculated as

$$\begin{aligned} MP_{R_1} &= (2000 \times 0.2) - 480 \\ &= 400 - 480 \\ &= -80 \Rightarrow 0. \end{aligned}$$

As there is no marginal profit in using d_{33}^* , no reducer is constructed for d_{13} .

Therefore the construction sequences that will be considered consist of the following semi-joins:

$$d_{21} \xrightarrow{300} d_{11}$$

$$d_{12} \xrightarrow{800} d_{22} \xrightarrow{530} d_{32}$$

The reducers produced are d_{11}^* and d_{32}^* . The size of d_{11}^* is estimated as

$$500 \times 0.5 = 250$$

and the size of d_{32}^* is estimated as

$$1400 \times 0.53 \times 0.6 = 446.$$

In step 2 of Algorithm W, the use of each reducer is considered. Reducer d_{11}^* is considered first, since it is the smallest. The effects of constructing the reducer; the relation sizes and the selectivity of d_{11}^* with respect to each relation is given below. Next, the effects of using the reducer are considered. The cost

R_i	$S(R_i)$	$\rho(d_{11}^*)$
1	1000	1.0
2	4500	0.83
3	6000	0.0

of $d_{11}^* \times d_{21}$ is 250 the benefit is 765, therefore this semi-join is profitable and should be appended to the final schedule. Consequently, the use of d_{11}^* implies that the semi-join sequence for constructing d_{11}^* should be appended to the final schedule prior to its use. Clearly, no other reductions are possible with d_{11}^* as

it cannot be used to reduce itself and relation R_3 does not have a common-join attribute for d_{i1} . After the use of the reducer, the reduction effects are estimated to be such that $|R_2| = 4500 \times 0.85 = 3735$.

Next, the use of reducer d_{32}^* is considered. The effects of its construction on the relation sizes and its selectivity with respect to each relation is shown below. The cost of $d_{32}^* \times R_1$ is 498, the benefit is 380; the cost of $d_{32}^* \times R_3$

R_i	$S(R_i)$	$\rho(d_{32}^*)$
1	1000	0.62
2	1980	0.93
3	2131	1.0

is 498, the benefit is only 138. Clearly, since neither semi-join is profitable, the reducer will not be constructed, leaving the following as the final schedule for the construction and application of reducers:

$$d_{21} \xrightarrow{300} d_{11}^* \quad d_{11}^* \xrightarrow{250} R_2$$

In step 3 of Algorithm W those reducers that were not used in steps 1 and 2 are reexamined, appending any profitable semi-joins to the final schedule. Taking into consideration the construction and use of the reducer d_{11}^* the current relevant statistics are:

R_i	$S(R_i)$	$S(d_{i2})$	$\rho(d_{i2})$	$S(d_{i3})$	$\rho(d_{i3})$
1	1000	800	0.53	600	0.60
2	3735	1000	0.67	—	—
3	6000	1400	0.93	800	0.80

The first semi-join considered is $d_{13} \bowtie d_{33}$. The cost is 600 and the benefit is 2400 so this semi-join is appended to the final schedule. The current relevant statistics¹⁰ subsequently become

R_i	$S(R_i)$	$S(d_{i2})$	$\rho(d_{i2})$
1	1000	800	0.53
2	3735	1000	0.67
3	3600	1400	0.76

Reconsidering the semi-join $d_{12} \bowtie d_{22}$ shows that it has a cost of 800 and a benefit of 1755. As a profitable semi-join it is appended to the final schedule. The final semi-join to consider is $d_{22} \bowtie d_{32}$ which has as cost of 530 and a benefit of 2321, therefore it is also appended to the final schedule. As there are no more semi-joins to be considered the final schedule produced by Algorithm W is:

$$\begin{array}{cccc}
 d_{21} \xrightarrow{300} d_{11} & d_{11}^* \xrightarrow{250} R_1 & d_{13} \xrightarrow{600} d_{33} & R_1 \xrightarrow{1000} QS \\
 & & d_{12} \xrightarrow{800} d_{22} & R_2 \xrightarrow{1980} QS \\
 & & d_{22} \xrightarrow{530} d_{32} & R_1 \xrightarrow{1279} QS
 \end{array}$$

Based on the sample query Algorithm AHY General (total time) produces a schedule with total cost of 7694 units. For the same data Algorithm W produces a schedule with a total cost of only 6739 units.

¹⁰ Note that it is not possible to correctly estimate the size for d_{32} however, the maximum it can be is 1145. The selectivity is changed to reflect this modification.

This example illustrates two problems associated with the AHY General (total time) algorithm.

1. Because the algorithm constructs reduction schedules for each relation independently of each other it does not take advantage of the possible use of highly effective reducers on other relations.
2. While the reduction schedules are executed in parallel, there is no synchronization mechanism in place to avoid redundant data transmissions¹¹. For example, consider final schedules that were produced by the AHY algorithm for relations R_2 and R_3 . Clearly, if these schedules were synchronized only one transmission of d_{12} would be required, reducing the total cost by 800 units.

4.4 ALGORITHM DW

Static strategies, such as Algorithm W rely on the accurate size estimation of intermediate results in order to produce good semi-join schedules. In static heuristics, small errors are propagated and typically compounded as the heuristic progresses, resulting in sub-optimal schedules [ES80]. In this section we propose a purely dynamic version of Algorithm W which we refer to as Algorithm DW. In this algorithm, only one semi-join is examined at a time and executed immediately if it is gainful. This heuristic requires minimal monitoring, and does not call for any modification of the execution schedule as the dynamic information eliminates

¹¹ Some of these issues are addressed in the COLLECTIVE version of the algorithm however, insufficient details were available to permit a comparative performance evaluation.

any estimation errors. Being a dynamic version of Algorithm W, the complexity and cost-effectiveness of Algorithm DW are identical to those of Algorithm W.

In the dynamic heuristic the use of centralized control (the query site) is assumed. Information on the sizes of intermediate results are relayed to the control site which decides on the next operation to be performed. Clearly, we incur additional overhead (as discussed in [BRJ89]) but our primary concerns are:

1. The number of messages that are sent from the relation sites to the query site, reporting on the size of partial results. In particular, the number of messages required is the same as the number of partial results produced, and we can assume that these messages are relatively small since only information on the cardinality of the reduced relation (the number of tuples) and the cardinality of the attributes of that relation need to be sent to the central site (query site).
2. The possibility of increased response time, since all of the reducers are constructed in serial rather than in parallel as in Algorithm W.

4.4.1 Description of Algorithm DW

In the case of the dynamic algorithm, all assumptions and definitions remain the same as in the static cases except for a slight variation in the definition of the selectivity of one attribute with respect to another. Consider the dynamic execution of the following semi-join sequence: $d_{aj} \bowtie d_{bj} \bowtie d_{cj}$. The selectivity of d_{aj} w.r.t. d_{bj} is estimated as

$$\frac{|d_{aj}|}{|D(d_{ij})|}$$

When the semi-join $d_{aj} \bowtie d_{bj}$ is executed, the cardinality of both the reduced R_b and d_{bj} is known exactly. The selectivity of d_{bj}^* with respect to d_{cj} is subsequently estimated as

$$\frac{|d_{bj}^*|}{|D(d_{ij})|}$$

4.4.2 Outline of Algorithm DW

Under the assumption that a query consists of m relations and n join-attributes and Let J be the set of all unused join-attributes and N the set of join-attributes that have been found non-profitable. Initially J contains all join-attributes and N is empty.

Algorithm DW is executed as follows:

Step 1: From the join-attribute set J , select attribute d_{ij} such that $\forall x, y S(d_{ij}) \leq S(d_{xy})$, $x = 1, \dots, m$; $y = 1, \dots, n$.

Step 2: Order the attributes d_{xj} by size such that $S(d_{aj}) \leq S(d_{bj}) \leq \dots \leq S(d_{mj})$.

Step 3: Consider the semi-join $d_{aj} \bowtie d_{bj}$. The semi-join is executed if and $MP(d_{aj} \bowtie d_{bj}) > 0$ or $P(d_{aj} \bowtie d_{bj}) < 0$ but $G(d_{aj} \bowtie d_{bj}) > 0$. If the semi-join is executed the next semi-join considered is $d_{bj}^* \bowtie d_{cj}$, otherwise $d_{aj} \bowtie d_{cj}$ is considered. This step is repeated until all applicable semi-joins are considered. Remove the common join-attribute j from J . If no semi-joins were performed add j to N and return to Step 1, otherwise continue.

Step 4: Perform all profitable semi-joins $d_{x_j}^* \bowtie R_i$, where $d_{x_j}^*$ is the reducer constructed in Step 3.

Step 5: Remove all common-join attributes from N and add them to J . Repeat steps 1 to 5 while J is not empty.

When the set J becomes empty Algorithm DW will have applied all of the profitable reductions that it was capable of identifying. Finally the reduced relations are shipped to the query site.

As noted in [Sel89], little work has been carried out in the validation of optimization algorithms. In most cases only analytical comparisons are made between algorithms. The use of empirical studies not only provides comparisons between algorithms but also a means of evaluating the validity of assumptions made and the techniques used.

5.1 METHODOLOGY

The framework for evaluating the algorithms is based on the following objectives:

- To Test Algorithm W with a wide variety of select-project-join (SPJ) type queries.
- To compare Algorithm W with the Apers-Hevner-Yao (AHY) algorithm.
- To compare the performance of Algorithm W with its dynamic version Algorithm DW.

Some would argue that using only SPJ queries is too restrictive with respect to the type of query tested. However, we do not consider this to be a limitation since it is possible to translate any query into SPJ form.

5.1.1 The Test Queries

For the evaluations a query is considered to be the statistical information on the relations and attributes that are participating in the query after all local site processing. While it is unrealistic to construct explicit queries as in [Bod85], this statistical representation facilitates the construction of a wide variety of test queries.

By varying a number of parameters it was possible to construct queries with the following characteristics:

- Each query consisted of between 1 and 6 relations and the number of join-attributes varied between 2 and 4. Overall, this gave us 12 different types of test queries (e.g. 3 relations – 2 attributes, 3 relations – 3 attributes, etc.)
- The cardinality of each join-attribute domain varied between 500 – 1500.
- Each relation had between 800 and 6000 tuples.
- To provide realistic queries, the number of join-attributes in each relation were varied between 1 and the maximum number of join-attributes with the restriction that the query remain “connected” in the sense that all relations must be joined to answer the query. For our evaluations we considered 3 levels of connectivity namely, 50%, 75% and 100%. A detailed outline of connectivity can be found in Appendix A.

- Each relation has one other (non-joining) attribute which is required at the query site.

Generating the query (statistics table) is accomplished in a 4 step process:

1. Given the number of relations and the maximum number of join-attributes, the cardinality of the domain for each join-attribute is randomly chosen.
2. Next, the occurrence of join attributes within each relation are randomly determined such that the desired connectivity is satisfied.
3. The cardinality for each join-attribute of each relation is randomly chosen such that it does not exceed the cardinality for its associated domain. In addition, the cardinality is restricted to guarantee that the selectivity will be in the range $0.5 \leq \rho(d_{ij}) \leq 1.0$.
4. Lastly, the cardinality for each relation is randomly chosen such that the cardinality of the relation exceeds the cardinality of any of its join-attributes.

The actual query construction is handled by the C program `create_query.c` (see Appendix B). Given the desired number of relations and the maximum number of join-attributes, the program will produce a query statistics table as well as the input parameters that are required for constructing the actual relations.

5.1.2 The Test Database

A major advantage in adopting the statistical representation of queries is that in order to execute a query we are only required to construct the relations that are

participating in the query (as opposed to having to construct an entire database).

The Wisconsin benchmark database proposed by Bitten, Dewitt and Turbyfill [BDT83] provided a good basis for developing the benchmark distributed database required for evaluating the queries. For simplicity, as in [BDT83], only integer values are considered. The primary difficulty with using the Wisconsin database as defined is that relations are populated with attribute values in an identical fashion. For example, suppose it is decided that the domain for some attribute say A is 1000, hence the possible values will be in the integer range 0 – 999. If for relation R_1 attribute A is to have a selectivity value of 0.5, attribute A will be populated with the integers 0 – 499. Similarly, if for relation R_2 , attribute A is to have a selectivity of 0.8, attribute A will be populated with values in the range 0 – 799. Clearly, using this method of populating the attribute values results in the creation of key attributes only. As the use of non-key attributes is of concern this particular approach to populating attribute values is not appropriate.

The benchmark database employed for evaluation purposes is based essentially on the Wisconsin benchmark database [BDT83] with modifications to the attribute domain value selection and population methods. The modifications are outlined as follows:

1. To overcome the problem of key attributes, the values for a particular attribute are randomly selected from the domain pool of values for that attribute. For example, suppose the cardinality of the domain for attribute A is 1000, which implies that the possible values are 0 – 999. If attribute A of relation R_1

is required to have a cardinality of 500 (or equivalently having a selectivity of 0.5), 500 different values will be randomly selected from domain, thus constituting the actual values for attribute A , $D(d_{1a})$.

2. When uniformly populating the relation with values, the values for attribute A are simply selected in a uniform manner from $D(d_{1a})$. Because the attribute domain is based on a random selection of values from the overall domain of values, the problem of attribute dependence is not encountered.
3. For non-uniform populations, each value in $D(d_{1a})$ is used once after which a beta function is used to select each subsequent value. Using each value at least once guarantees the correct initial selectivity for the attribute, after which the beta function provides a skewed distribution in the number of occurrences of each value.

In addition to the statistical information, `create_query.c` also produces input files corresponding to each relation in the query. These input files are subsequently used by the `relbuilder.c` program to construct the relations that are described in the statistical table. Details regarding `relbuilder.c` can be found in Appendix B.

5.2 EXPERIMENTAL RESULTS

The performance the heuristics was evaluated with 6 test runs, corresponding to each connectivity – distribution pair (e.g. 50% connectivity – uniform distribution, 50% connectivity – random distribution, etc.) Each run consisted of 1,200

queries. For each query, 100 semi-join schedules were constructed and executed using each of the heuristics, recording the costs incurred. Overall, a total of 7,200 queries were used to evaluate the performance of the algorithms.

Descriptions of each run can be found in Table 5.1. Summaries of the data collected for each run can be found in Appendix C.

<u>Run #</u>	<u>Distribution</u>	<u>Connectivity</u>
1	uniform	100%
2	random	100%
3	uniform	75%
4	random	75%
5	uniform	50%
6	random	50%

Table 5.1 Descriptions of individual runs.

5.2.1 Relevance of Results

The application of t tests (see Appendix B) to the experimental results clearly indicate that for all six experimental runs, the differences observed in the percent reduction of Algorithm W to that of the AHY Algorithm are statistically significant. In particular, the probability that the results are due to chance under 198 degrees of freedom is approximately 1:10,000. This is not the case for the comparison of Algorithm DW to Algorithm W.

Given the relative similarity in results for Algorithms DW and W it is not surprising to find the difference in percent reduction was not found to be significant in all runs. In Table 5.2, the runs in which it was not possible to disprove the

Query	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6
3-2	-	-	-	-	-	-
3-3	-	-	-	-	-	-
3-4	-	-	-	-	-	-
4-2	-	-	-	-	-	-
4-3	-	■	-	■	-	-
4-4	-	■	-	■	-	■
5-2	-	■	-	-	-	-
5-3	■	■	■	■	-	-
5-4	-	■	■	■	■	-
6-2	-	-	-	■	■	-
6-3	■	■	■	■	■	-
6-4	■	■	■	■	■	■

Table 5.2 Statistical relevance of differences between Algorithm DW and Algorithm W.

null hypothesis (that the differences in the means arose by chance) are indicated by a square. Clearly for uniform distributions (runs 1, 3 and 5), the results are significant for queries 3-2 to 5-2. For random distributions (runs 2, 4 and 6), the results are only significant for queries 3-2 to 4-2. Additional experimentation with larger run sizes should be conducted to determine whether the questionable runs are in fact significant.

5.3 CONCLUSIONS

5.3.1 Algorithm W versus AHY (total cost)

The following conclusions are made based on the results of the test runs:

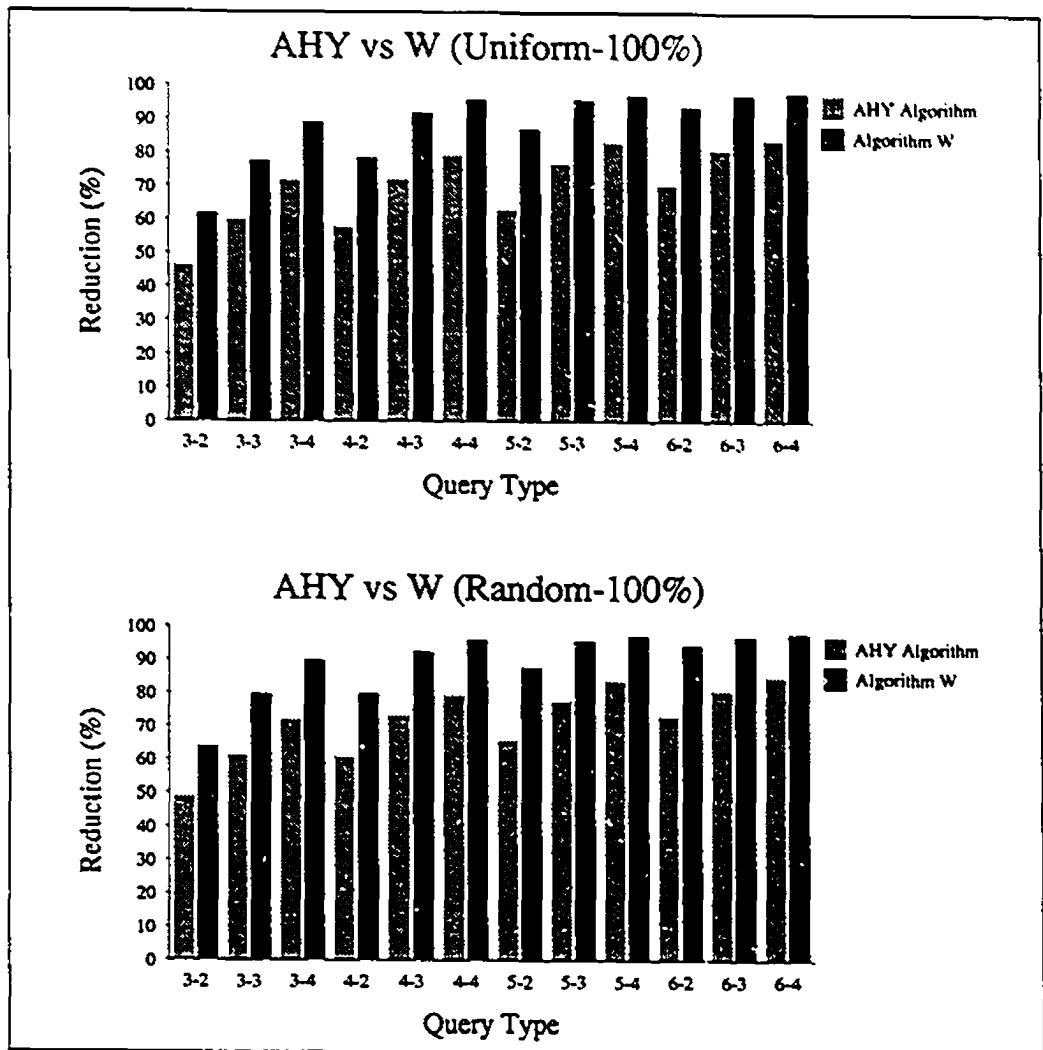


Figure 5.1 W – AHY cost comparisons for 100% connectivity.

- Overall Algorithm W performs satisfactory in reducing the volume of network data transfers during the query processing. On average it provides a reduction of between 32% and 97% (approximately) over the unoptimized total cost¹².

¹² Note, the degree of reduction is dependent upon the number of join-attributes as well as the overall connectivity of the query.

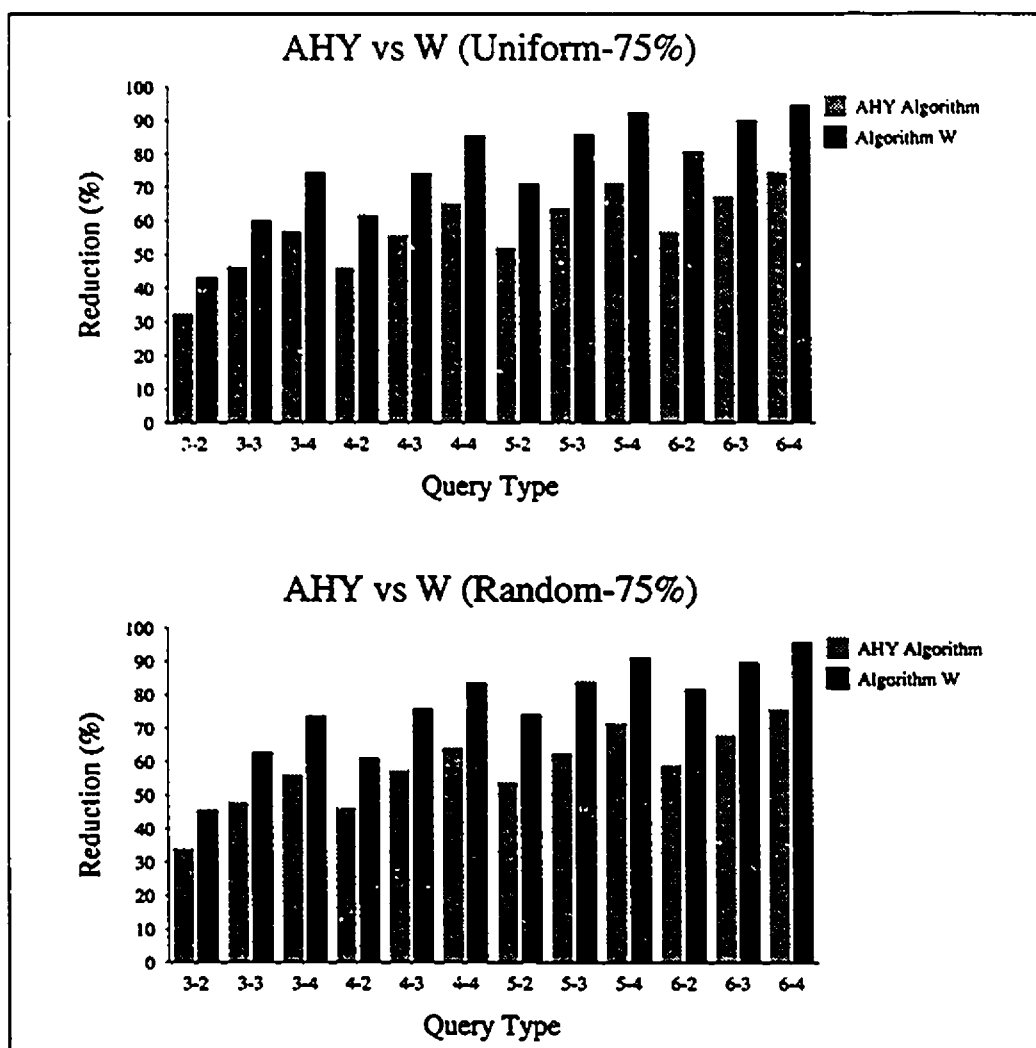


Figure 5.2 W – AHY cost comparisons for 75% connectivity.

- Algorithm W clearly outperforms AHY (as illustrated in Figures 5.1–5.3). On average Algorithm W outperforms AHY by approximately 18%. The greatest difference in performance is found in those queries involving only two join-attributes, with queries involving three relations being an exception.
- Results indicate that Algorithm W performs well under both uniform and

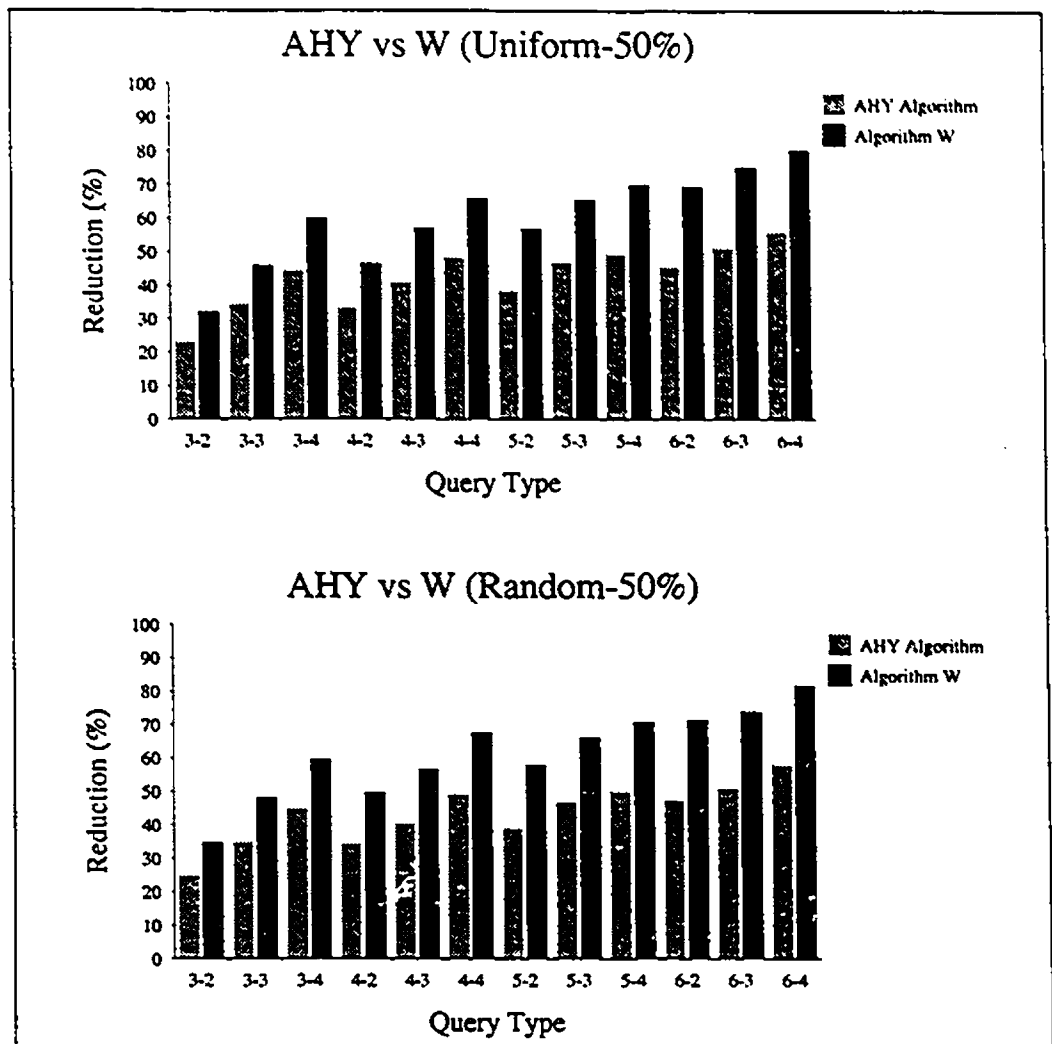


Figure 5.3 W – AHY cost comparisons for 50% connectivity.

random data distributions.

- Null queries¹³ were a frequent occurrence in the 100% connectivity queries. In particular, the null queries occurred most frequently above the 4–2 query type. From Figure 5.3 and Tables C.1 and C.2 it is evident that the execution schedules produced by Algorithm W produce a null query very quickly due

¹³ A null query simply refers to an empty solution.

to the manner in which the schedules are constructed. AHY by contrast, produces a null query late in its execution or not until all of the all of the final relations are joined. Clearly, it is advantageous to identify null queries as early as possible, thus minimizing the amount of unnecessary network data transfers.

- Algorithm W does not suffer from the synchronization and redundant transmission problems that arise from attempting to optimize the schedules produced by AHY General (total time).

5.3.2 Algorithm W versus Algorithm DW

A comparison of Algorithm W to Algorithm DW was undertaken to answer the following questions:

Will a purely dynamic heuristic outperform its static counterpart ?

For the most part the answer is unequivocally “no” (see Figures 5.4–5.6) however, the relevance tests indicate that further experimentation is required for query types greater than 5–2 under uniform distributions (and for query types greater than 4–2 for random distributions). Overall, Algorithm W outperforms DW by 2% to 6% on average (approximately). In the case of the three relation, two attribute query type, the difference is quite significant. This is a result of the naive strategy adopted by Algorithm DW. Clearly, the availability of up to date information is not sufficient for the heuristic, hence some method must be provided to take into account any global conditions. Currently, the greedy approach used by DW considers the execution of a semi-join with no form of “look ahead” except

for the calculation of marginal profit. In addition, the decision to begin with the cheapest semi-join may not lead to the construction of the best reducer. If for a particular semi-join the anticipated reduction does not occur, the execution of that semi-join has become an unneeded cost expenditure. In large queries these wasted transmissions are generally countered by improvements gained over the course of the execution of the query. However, in small queries the limited number of semi-joins does not allow DW to recover from "poor" semi-joins.

The problem is to design an improved algorithm which will construct the "best" reducer first while still attempting to minimize the overheads. The results clearly indicate that further research is required in this area.

Will the response time of a dynamic heuristic (DW) increase due to the increased number of serially executed semi-joins ?

In most cases a significant increase in response time is experienced however, in some cases with large queries under 50% connectivity the response time is actually improved with respect to Algorithm W. (See Table C.7). However, the question of relevance along with the proposal of an improved heuristic requires that further testing be performed in order to fully answer this question.

Does the transmission of the statistics regarding intermediate results back to the query site constitute a significant cost ?

Based on the results, the transmission of the statistics of intermediate results does not constitute a significant cost. In the significant cases, this overhead does not exceed 3% of the overall total volume of data transferred to process the

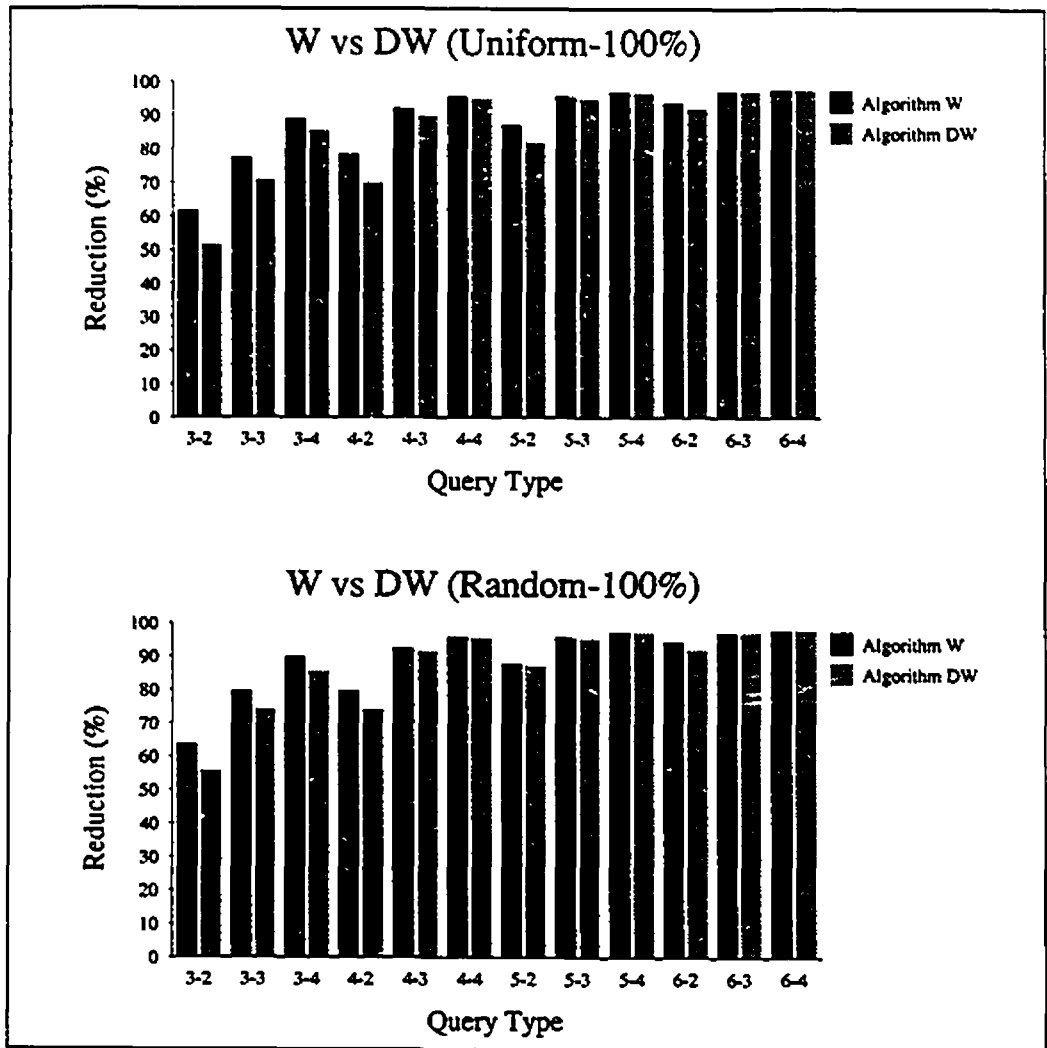


Figure 5.4 W - DW cost comparisons for 100% connectivity.

query. (See Table C.8)

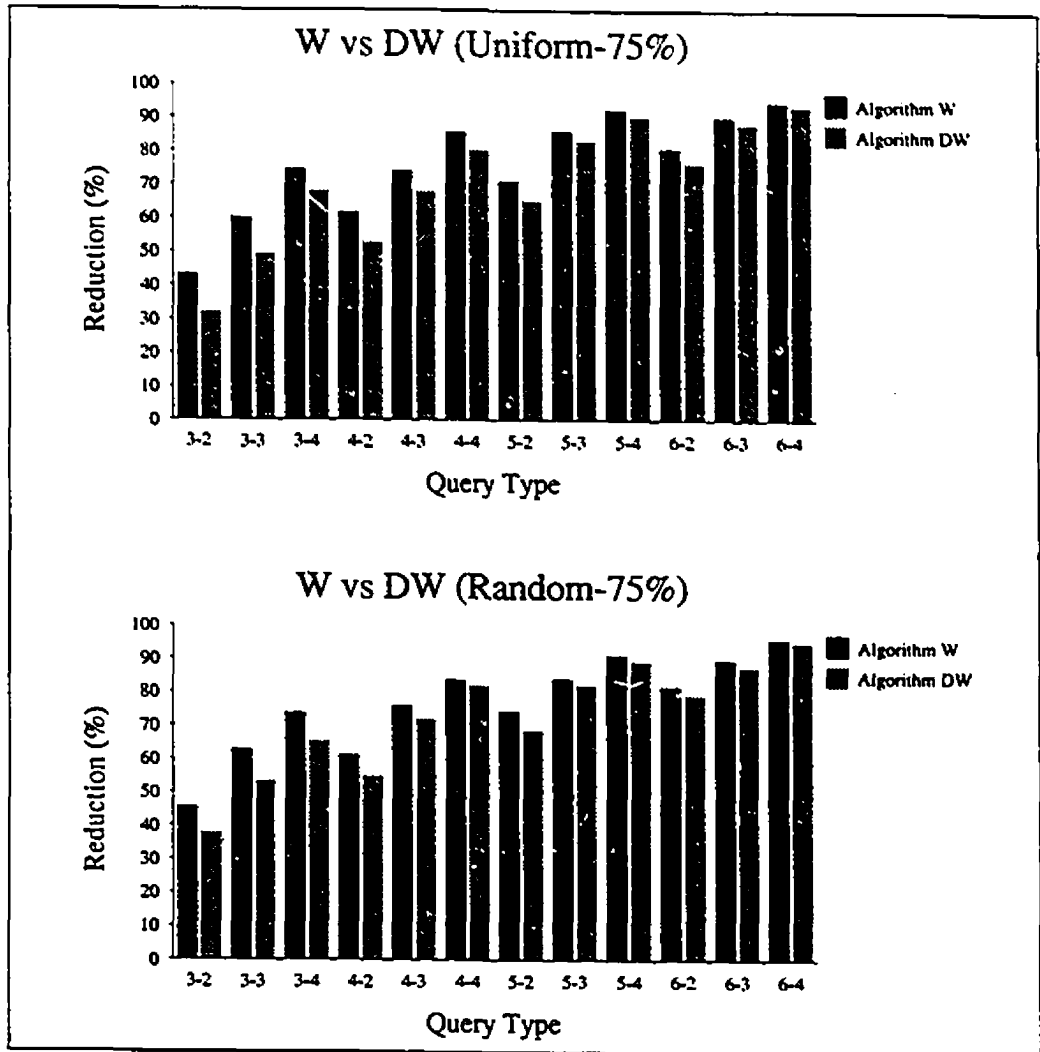


Figure 5.5 W – DW cost comparisons for 75% connectivity.

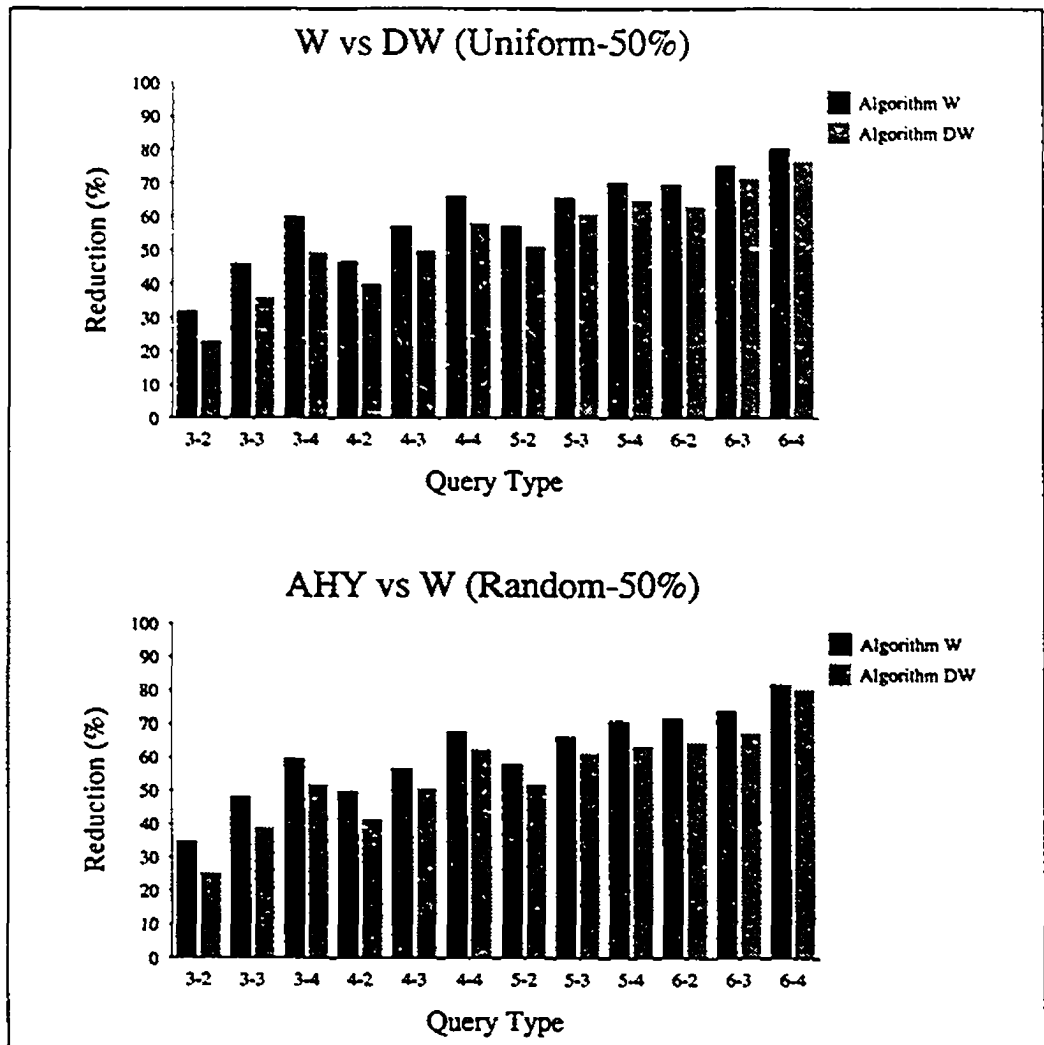


Figure 5.6 W – DW cost comparisons for 50% connectivity.

Chapter 6

Conclusions and Future Work

Two new semi-join based heuristics for minimizing the total volume of network data transfers in distributed query processing have been presented. Algorithm W is a static heuristic which uses the concepts of profit, marginal profit and gain to construct inexpensive and highly selective reducers. Algorithm W has been shown to be both efficient and cost effective with a worst case complexity of $O(nm^2)$ and a best case complexity of $O(nm)$. Algorithm DW is a purely dynamic version of Algorithm W which executes semi-joins in a greedy one at a time manner. Unlike other dynamic heuristics proposed, DW does not require schedule monitoring or reformulation during execution.

The experiments using random data distributions indicate that heuristics using a uniform data distribution assumption do not experience any noticeable drop in performance. Additional experimentation should be carried out on very large relations to see if this is still the case.

Extensive testing of the algorithms indicate that Algorithm W consistently outperforms the Apers-Hevner-Yao (AHY) General (total time) algorithm. Re-

sults seem to indicate that the Algorithm W outperforms Algorithm DW, even though the differences were not significant. It should be noted that the lack of relevant statistical results suggest that additional experimentation is required to form a definitive answer. It was also shown that the overheads associated with the dynamic heuristic were minimal with respect to the overall cost, thus illustrating that the use of up to date information does not require extensive overhead. Acknowledging that Algorithm DW is essentially a very naive and simple heuristic, it is clear that further research is required to determine whether it is possible to develop a dynamic heuristic which will provide any significant improvement over Algorithm W.

6.1 FUTURE WORK

Continued development of both static and dynamic heuristics. Some specific examples include:

- Modify algorithm W to use marginal profit as the selection criteria, as opposed to the current method based on minimum cost.
- Using the concept of marginal profit, develop a new dynamic heuristic which constructs reducers using more than one common-join attribute.
- Investigate the use of bloomfilters [Mul90, Mul93] in a dynamic heuristic. The characteristics of bloomfilters suggests that they can provide insight into the relative reduction capabilities of each of the join-attributes.

Another key area of continued research is in the development of more sophisticated and flexible benchmark database. The desired database should allow for the use of primary keys, foreign keys, and composite keys. In addition, controls would be added to provide some degree attribute distribution and attribute dependence in order to more closely model real world data. Lastly, the software for constructing the database should be flexible enough to allow for random database generation, based on predefined parameters or manual construction via a user interface.

Selected Bibliography

- [AHY83] P.M.G. Apers, A.R. Hevner, and S.B. Yao. Optimization algorithms for distributed queries. *IEEE Trans. Software Engineering*, SE-9(1):57–68, 1983.
- [AM91] J.K. Ahn and S.C. Moon. Optimizing joins between two fragmented relations on a broadcast local network. *Information Systems*, 16(2):185–98, 1991.
- [BDT83] D. Bitton, D.J. DeWitt, and C. Turbyfill. Benchmarking database systems a systematic approach. In *Proceedings of the 9th VLDB Conf.*, pages 8–19, 1983.
- [BGW⁺81] P.A. Bernstien, N. Goodman, E. Wong, C.L. Reeve, and J. Rothnie. Query processing in a system for distributed databases (SDD-1). *ACM Trans. Database Systems*, 6(4):105–128, 1981.
- [Bod85] P. Bodorik. *Query processing strategies in a distributed database*. PhD thesis, Carleton University, Ottawa, Canada, 1985.
- [BR88a] P. Bodorik and J.S. Riordon. Distributed query processing optimization objectives. In *Proc. of the 4th Int. Conf. on Data Engineering*, pages 320–329, 1988.
- [BR88b] P. Bodorik and J.S. Riordon. A threshold mechanism for distributed query processing. In *Proc. of the ACM CSC '88 Conf., Atlanta, GA*, pages 616–625, 1988.
- [BRJ89] P. Bodorik, J.S. Riordon, and C. Jacob. Dynamic distributed query processing techniques. In *Proc. of the 17th ACM Compt. Sci. Conf.*, pages 348–357, 1989.
- [CL84] L.P. Chen and V.O.K. Li. Improvement algorithms for semijoin query processing programs in distributed database systems. *IEEE Trans. Computers*, C-33(11):959–967, 1984.
- [CL87] J.S.J. Chen and V.O.K. Li. Optimizing joins in fragmented database systems on a broadcast network. In *Proceedings of the 7th Intern'l Conf. on Distributed Computing Systems*, pages 338–345, 1987.
- [CL90] J.S.J. Chen and V.O.K. Li. Domain-specific semijoin: A new operation for distributed query processing. *Information Sciences*, 52:165–183, 1990.
- [CP84] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems*. McGraw-Hill, 1984.

- [CY90] M. Chen and P.S. Yu. Using join operations as reducers in distributed query processing. In *Proc. of the 2nd Int. Conf. on Databases in Parallel and Distributed Systems*, pages 116–123, 1990.
- [ES80] R. Epstein and M. Stonebraker. Analysis of distributed data base processing strategies. In *Proceedings of the 6th VLDB Conf.*, pages 310–319, 1980.
- [ESW78] R. Epstein, M. Stonebraker, and E. Wong. Distributed query processing in a relational data base system. In *ACM SIGMOD Conf.*, pages 169–180, 1978.
- [Gra89] G. Graefe. Research problems in database query optimization. In G. Graefe, editor, *Proceedings of the ODBF Workshop*, pages 1–12, 1989.
- [HWY85] A.R. Hevner, O.Q. Wu, and S.B. Yao. Query optimization on local area networks. *ACM Trans. Office Information*, 3(1):35–62, 1985.
- [KR87] H. Kang and N. Roussopoulos. Using 2–way semijoins in distributed query processing. In *Proceedings of the 3rd Intern'l Conf. on Data Engineering*, pages 644–650, 1987.
- [Loh89] G.M. Lohman. Is query optimization a “solved” problem? In *Proceedings of the ODBF Workshop*, number Tech Report CS/E 89–005, pages 13–18, May 1989.
- [Mul90] J.K. Mullin. Optimal semijoins for distributed database systems. *IEEE Trans. Software Engineering*, 16(5):558–560, 1990.
- [Mul93] J.K. Mullin. Estimating the size of a relational join. *Information Systems*, 18(3):189–196, 1993.
- [OS88] B.C. Oui and B. Srinivasan. On the identification of semijoin sequences for distributed query processing. In *Distributed Processing*, pages 314–325. North-Holland, 1988.
- [OV91] M.T. Ozsu and P. Valduriez. *Principles of distributed database systems*. Prentice Hall International, 1991.
- [PC90] W. Perrizo and C. Chen. Composite semijoins in distributed query processing. *Information Sciences*, 50:197–218, 1990.
- [PLH89] W. Perrizo, J.Y.Y. Lin, and W. Hoffman. Algorithms for distributed query processing in broadcast local area networks. *IEEE Trans. Knowledge and Data Engineering*, 1(2):215–225, 1989.
- [RK91] N. Roussopoulos and H. Kang. A pipeline N-way join algorithm based on the 2–way semijoin program. *IEEE Trans. Knowledge and Data Engineering*, 3(4):486–495, 1991.

- [Sel89] P.G. Selinger. Five hard problems in query optimization. In G. Graefe, editor, *Proceedings of the ODBF Workshop*, number Tech Report CS/E 89-005, pages 19-22, 1989.
- [Teo92] T.J. Teorey. On dependability in distributed databases. Technical report, Center for Information Technology Integration, University of Michigan, 1992.
- [TI90] F.W. Tompa and J.I. Icaza. Adaptive selection of query execution strategies by learning automata. *Information Sciences*, 50:219-240, 1990.
- [VV84] Y.L. Varol and S.V. Vrbsky. Distributed query processing allowing for redundant data. In *Proceedings of the 4th Intern'l. Conf. on Distributed Computing Systems*, pages 389-396, 1984.
- [WC93] C. Wang and M. Chen. On the complexity of distributed query optimization. Technical Report RC 18671 (81715), IBM Research Division (USA), 1993.
- [WLC91] C. Wang, V.O.K. Li, and A.L.P. Chen. Distributed query optimization by one-shot fixed precision semi-join execution. In *Proceedings of the 7th Int. Conf. on Data Engineering*, pages 756-763, 1991.
- [YGC88] C.T. Yu, K.C. Guh, and A.L.P. Chen. An integrated algorithm for distributed query processing. In *Distributed Processing*, pages 507-521. North-Holland, 1988.
- [Yu85] C.T. Yu. Distributed query processing. In *Query Processing in Database Systems*, pages 48-61. Springer-Verlag, 1985.

The term “connectivity” is used to describe the general underlying presence of join-attributes within a query. For any query it is somewhat unrealistic to assume that every relation will have an occurrence of each join-attribute. To allow for varied occurrences of join-attributes within relations, the probability that a relation will have a specific join-attribute is based upon some probability. For the experiments conducted in this thesis, these probabilities were chosen to be 50%, 75% and 100%. It is important to note that the use of probabilistic selection alone will may not result in a valid query. For the heuristics presented in this thesis, the following conditions must be satisfied:

1. At least two relations must have an occurrence of the same join-attribute.
2. It must be possible to join every relation to form a single conjunctive normal form query.

To determine whether condition 2 holds, a graph is constructed with the relations as the nodes and the join-attributes as the edges. Condition 2 holds if it can be shown that the graph is fully connected, hence the term “connectivity”.

Therefore, a query with connectivity of 50% refers to one in which approximately half of the join-attributes do not occur within the relations. It is important to note that these percentages are approximations with respect to the queries of 5 and 6 relations. The exact minimum coverage (%) is given by the formula

$$\frac{2m - (n - 2)}{nm} \times 100, \quad n \geq 2; m \geq 1$$

where m represents the number of join-attributes and n the number of relations.

Proof. Let $n = 2$ and m be some arbitrary positive integer. With only two relations, condition 1 requires that each relation must have an occurrence of each of the m join-attributes. This also guarantees that condition 2 holds as well. Therefore, for two relations the minimum coverage is 100%. Hence the formula holds for $n = 2$.

Clearly, for each additional relation that is added to the query, it only requires the presence of one join-attribute (i.e. $n - 2$) in order to satisfy condition 2. Condition 1 will always be satisfied as the first two relations must have an occurrence of every join-attribute.

Hence, the formula computes the exact minimum coverage (%) of join-attributes for a query involving n relations and m join-attributes. \square

Appendix B

WWW Availability

Copies of this thesis, the programs described within, and the raw statistical results are available via the World Wide Web at the following URL:

<http://www.cs.uwindsor.ca/meta-index/research/dbrg/>

The programs used in this thesis are given in the following table.

<u>File Name</u>	<u>Description</u>
create_query.h	The header file for the create_query.c program.
create_query.c	Program for creating the query statistics.
relbuilder.h	The header file for the relbuilder.c program.
relbuilder.c	This program uses the query statistics to construct actual relations that match the statistical characteristics. Note, this program allows for either uniform or random data distributions when generating the relations.
betaf.c	The function used to generate the random distributions.
ahy.h	The header file for the ahy.c program.
ahy.c	The main logic for the Apers-Hevner-Yao Algorithm General (total time).
w.h	The header file for the w.c program.
w.c	The main logic for Algorithm W heuristic.

DW.h	The header file for the DW.c program
DW.c	The main logic for the Dynamic W heuristic.
dyn_sjoin.c	Program for executing semi-joins in the Dynamic W heuristic.
runAHY.c	Program for executing the schedules produced by the AHY algorithm on the physical database.
runW.c	Program for executing the schedules produced by the Algorithm W on the physical database.
sjoin.c	The function for executing semi-joins between the physical relations.

C.1 TOTAL COST

The experimental results for total cost analysis have been summarized into the following tables. Query types are given in column 1. The entries in each row (query type) represent the average over 100 runs. Column 2 gives the percentage by which the AHY algorithm reduces the unoptimized total cost; similarly columns 3 and 4 represent the respective percent reductions obtained by Algorithm W and DW. Column 5 shows the percentage improvement of Algorithm W over AHY and column 6 gives the percentage improvement of Algorithm DW over W. The averages over all of the query types are given at the bottom of the table.

<u>Type</u>	<u>AHY</u>	<u>W</u>	<u>DW</u>	<u>W-AHY</u>	<u>DW-W</u>
3-2	46.29	61.70	51.58	15.41	-10.12
3-3	59.93	77.48	70.83	17.54	-6.64
3-4	71.66	89.10	85.58	17.44	-3.53
4-2	57.65	78.43	69.96	20.78	-8.47
4-3	72.07	91.97	89.87	19.90	-2.11
4-4	79.14	95.50	94.96	16.36	-0.53
5-2	62.98	87.12	82.09	24.14	-5.03
5-3	76.83	95.67	94.90	18.84	-0.77
5-4	83.13	97.08	96.76	13.95	-0.32
6-2	70.51	93.64	92.00	23.13	-1.63
6-3	80.98	97.12	97.12	16.14	0.01
6-4	83.97	97.71	97.47	13.74	-0.24
Averages:	70.43	88.54	85.26	18.12	-3.28

Table C.1 Uniform distribution with approx. 100% connectivity.

<u>Type</u>	<u>AHY</u>	<u>W</u>	<u>DW</u>	<u>W-AHY</u>	<u>DW-W</u>
3-2	48.66	63.70	55.79	15.04	-7.91
3-3	61.15	79.56	74.02	18.41	-5.54
3-4	71.70	89.78	85.53	18.08	-4.25
4-2	60.62	79.72	74.02	19.10	-5.69
4-3	72.93	92.33	91.64	19.41	-0.70
4-4	79.32	95.84	95.39	16.52	-0.45
5-2	65.53	87.72	87.07	22.19	-0.65
5-3	77.62	95.77	95.21	18.15	-0.56
5-4	83.76	97.30	97.09	13.54	-0.21
6-2	73.16	94.35	91.88	21.19	-2.47
6-3	80.75	97.04	97.07	16.28	0.03
6-4	84.82	97.81	97.71	12.99	-0.10
Averages:	71.67	89.24	86.87	17.57	-2.37

Table C.2 Random distribution with approx. 100% connectivity.

<u>Type</u>	<u>AHY</u>	<u>W</u>	<u>DW</u>	<u>W-AHY</u>	<u>DW-W</u>
3-2	32.56	43.14	32.16	10.59	-10.99
3-3	46.37	60.02	49.23	13.65	-10.79
3-4	56.88	74.36	68.08	17.48	-6.28
4-2	46.02	61.55	52.81	15.53	-8.74
4-3	55.72	74.04	67.84	18.32	-6.21
4-4	65.25	85.47	80.27	20.21	-5.19
5-2	52.08	70.94	65.07	18.85	-5.86
5-3	63.72	85.84	83.16	22.12	-1.97
5-4	71.34	92.35	90.38	21.00	-1.97
6-2	56.72	80.82	76.47	24.09	-4.34
6-3	67.32	90.07	88.15	22.75	-1.92
6-4	74.45	94.56	93.54	20.11	-1.01
Averages:	57.37	76.10	70.60	18.73	-5.50

Table C.3 Uniform distribution with approx. 75% connectivity.

<u>Type</u>	<u>AHY</u>	<u>W</u>	<u>DW</u>	<u>W-AHY</u>	<u>DW-W</u>
3-2	33.93	45.44	37.78	11.51	-7.66
3-3	47.80	62.73	53.39	14.93	-9.34
3-4	56.08	73.58	65.37	17.50	-8.21
4-2	46.22	61.03	54.86	14.81	-6.16
4-3	57.31	75.84	71.75	18.53	-4.09
4-4	64.22	83.59	81.93	19.37	-1.66
5-2	53.83	74.21	68.72	20.38	-5.50
5-3	62.65	83.92	82.08	21.27	-1.84
5-4	71.52	91.16	89.31	19.63	-1.85
6-2	-1.85	81.64	79.07	22.61	-2.57
6-3	67.91	89.74	87.78	21.84	-1.96
6-4	75.64	95.80	95.11	20.16	-0.69
Averages:	58.01	76.56	72.26	18.54	-4.29

Table C.4 Random distribution with approx. 75% connectivity.

<u>Type</u>	<u>AHY</u>	<u>W</u>	<u>DW</u>	<u>W-AHY</u>	<u>DW-W</u>
3-2	22.96	31.99	23.28	9.03	-8.71
3-3	34.20	45.95	36.14	11.76	-9.82
3-4	44.45	60.10	49.37	15.65	-10.73
4-2	33.31	46.47	40.05	13.16	-6.42
4-3	41.02	57.06	49.78	16.04	-7.27
4-4	43.99	66.02	58.01	17.63	-8.02
5-2	38.44	57.01	51.23	18.57	-5.78
5-3	46.86	65.59	60.73	18.73	-4.86
5-4	49.25	70.05	64.88	20.80	-5.17
6-2	45.36	69.50	63.01	24.14	-6.49
6-3	51.15	75.23	71.40	24.08	-3.83
6-4	55.85	80.22	76.54	24.36	-3.67
Averages:	42.60	60.43	53.70	17.83	-6.73

Table C.5 Uniform distribution with approx. 50% connectivity.

<u>Type</u>	<u>AHY</u>	<u>W</u>	<u>DW</u>	<u>W-AHY</u>	<u>DW-W</u>
3-2	24.82	34.63	25.32	9.82	-9.32
3-3	34.82	48.08	38.96	13.27	-9.12
3-4	44.74	59.67	51.96	14.92	-7.71
4-2	34.21	49.66	41.56	15.45	-8.10
4-3	40.29	56.68	50.85	16.39	-5.83
4-4	49.15	67.66	62.48	18.52	-5.18
5-2	39.00	57.98	51.99	18.98	-5.99
5-3	46.93	66.28	61.32	19.35	-4.96
5-4	49.94	70.87	63.68	20.94	-7.19
6-2	47.46	71.55	64.44	24.10	-7.11
6-3	50.95	73.86	67.53	22.91	-6.33
6-4	58.04	81.80	80.13	23.75	-1.66
Averages:	43.36	61.56	55.02	18.20	-6.54

Table C.6 Random distribution with approx. 50% connectivity.

C.2 RESPONSE TIME

The following Table summarizes the percent increase in response time that Algorithm DW incurs over Algorithm W. The columns correspond to the individual test runs which are described in chapter 5.

Type	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6
3-2	72.12	72.38	76.71	67.70	70.06	83.78
3-3	65.16	66.71	80.69	81.94	60.20	54.65
3-4	67.62	63.74	59.69	59.69	45.80	46.32
4-2	78.12	86.36	96.46	73.76	87.80	85.12
4-3	62.63	71.03	71.05	71.12	29.83	21.98
4-4	71.36	52.90	65.17	61.96	10.49	9.12
5-2	77.22	69.73	78.72	88.35	62.29	32.38
5-3	66.84	83.32	57.35	50.00	17.25	11.00
5-4	64.87	88.63	46.63	49.47	5.10	3.22
6-2	70.88	86.11	76.38	67.95	26.23	40.07
6-3	72.07	75.20	48.59	72.68	-4.77	-3.54
6-4	79.21	55.27	36.30	24.75	2.31	-2.53
Averages:	70.67	72.62	66.14	65.21	34.38	31.80

Table C.7 Percentage increase in the response time for Algorithm DW over W.

C.3 OVERHEAD COSTS

The following Table summarizes the costs incurred by overhead in Algorithm DW as a percentage of the overall total cost.

Type	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6
3-2	0.05	0.05	0.06	0.08	0.12	0.14
3-3	0.10	0.11	0.13	0.15	0.30	0.35
3-4	0.17	0.19	0.32	0.27	0.82	0.83
4-2	0.08	0.09	0.12	0.12	0.22	0.26
4-3	0.13	0.13	0.25	0.28	1.02	1.30
4-4	0.20	0.24	0.50	0.60	2.45	2.79
5-2	0.11	0.12	0.17	0.21	0.41	0.62
5-3	0.18	0.19	0.52	0.49	2.00	2.21
5-4	0.24	0.23	1.08	0.99	3.52	3.95
6-2	0.17	0.18	0.29	0.35	0.99	0.95
6-3	0.28	0.24	0.76	0.73	3.48	3.57
6-4	0.39	0.47	1.55	2.18	4.06	4.62
Averages:	0.18	0.19	0.48	0.54	1.62	1.80

Table C.8 Overhead as a percentage of the total cost in Algorithm DW.

Vita Auctoris

NAME	William T. Bealor
PLACE OF BIRTH	Morris, Illinois, U.S.A.
YEAR OF BIRTH	1966
EDUCATION	W. F. Herman Secondary School, Windsor 1980 — 1985 University of Windsor, Windsor Ontario 1988 — 1993 BCS (Honours) University of Windsor, Windsor Ontario 1993 — 1996 M.Sc.