The Pennsylvania State University
The Graduate School
College of Engineering

ALGORITHMS TO IDENTIFY PARETO POINTS IN MULTI-DIMENSIONAL DATA SETS

A Thesis in
Mechanical Engineering
by
Michael A. Yukish

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2004

The thesis of Michael A. Yukish was reviewed and approved* by the following:

Timothy W. Simpson
Associate Professor of Mechanical Engineering and Industrial Engineering
Thesis Advisor
Chair of Committee

Mary Frecker
Associate Professor of Mechanical Engineering

Mark Traband
Affiliate Professor of Industrial Engineering

Soundar R. T. Kumara
Distinguished Professor of Industrial and Manufacturing Engineering

Richard C. Benson
Professor and Head of Department of Mechanical Engineering

*Signatures are on file in the Graduate School.

Abstract

The focus in this research is on developing a fast, efficient hybrid algorithm to identify the Pareto frontier in multi-dimensional data sets. The hybrid algorithm is a blend of two different base algorithms, the Simple Cull (SC) algorithm that has a low overhead but is of overall high computational complexity, and the Divide & Conquer (DC) algorithm that has a lower computational complexity but has a high overhead. The hybrid algorithm employs aspects of each of the two base algorithms, adapting in response to the properties of the data.

Each of the two base algorithms perform better for different classes of data, with the SC algorithm performing best for data sets with few nondominated points, high dimensionality, or fewer total numbers of points, while the DC algorithm performs better otherwise. The general approach to the hybrid algorithm is to execute the following steps in order:

1. Execute one pass of the SC algorithm through the data if merited
2. Execute the DC algorithm, which recursively splits the data into smaller problem sizes
3. Switch to the SC algorithm for problem sizes below a certain limit

In order to determine whether Step 1 should be executed, and to determine at what problem size the switch in Step 3 should be made, estimates of both algorithms' run times as a function of the size of the data set, the dimension of the data set, and the expected number of nondominated points are needed. These are developed in the thesis.

To aid in increasing the speed and reducing the computational and storage complexity of the algorithm, and to enable the ability for the algorithm to adapt to the data, a canonical transformation of the data to a *Lattice Latin Hypercube* (LLH) form is developed. The transformation preserves the Pareto property of points but reduces storage space and algorithm run time.

In order to test the three algorithms, three different methods for creating randomized data sets with arbitrary dimensionality and numbers of nondominated points are developed. Each of the methods provides insight into the properties of nondominated sets, along with providing test cases for the algorithms. Additionally, a spacecraft design problem is developed to serve as a source of real world test data.

Table of Contents

List of Tables

List of Figures

Nomenclature

$\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \ldots$	A table of data, also can be thought of as a set of points in a multi-dimensional space
$\mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots$	A single row in a table, or a single point in a multi-dimensional space
$\mathbf{x}_i$	The ith row in table **X**
x	A single element from table **X**
x_i	The ith element in row **x**
x_{ij}	The jth element in the ith row of table **X**
$\ln n$	Natural logarithm of n
$\log n$	Logarithm Base 2 of n
$\log_\alpha n$	Logarithm Base α of n

All of the following functions are predicated on the data being all points Pareto

mbi[n]	Estimator for the the number of comparisons of the marry algorithm, infinite dimension, and balanced. Takes size of table as input
pbi[n]	Estimator for the the number of comparisons of the Pareto algorithm, infinite dimension, and balanced. Takes size of table as input.
pgui[n]	Estimator for number of comparisons, Pareto algorithm, generalized unbalanced case, infinite dimension.
mgui[n]	Estimator for number of comparisons, Marry algorithm, generalized unbalanced case, infinite dimension.
unbi[n, b]	Estimator for the maximum number of comparisons required to identify Pareto points in the left/right side of the marry algorithm, with n total points of which b are to be tested
pgui[n, α]	Estimator for number of comparisons, Pareto algorithm, generalized unbalanced case, infinite dimension. The parameter α indicated the level of unbalance.
mmui[n, α]	Estimator for number of comparisons, Marry algorithm, maximum unbalanced case, infinite dimension. The parameter α indicates the level of unbalance.
pmui[n]	Estimator for number of comparisons, Pareto algorithm, maximum unbalanced case, infinite dimension
mbf[n,d]	Estimator for number of comparisons, Marry algorithm, balanced case, with finite dimension d
pbf[n]	Estimator for number of comparisons, Pareto algorithm, balanced case, with finite dimension d.

pgui[*n, α*]	Estimator for number of comparisons, Pareto algorithm, generalized unbalanced case, finite dimension *d*. The parameter α indicates the level of unbalance.
mgui[*n, α*]	Estimator for number of comparisons, Marry algorithm, generalized unbalanced case, finite dimension *d*. The parameter α indicates the level of unbalance.
mmuf[*n,d*]	Estimator for number of comparisons, Marry algorithm, maximum unbalanced case, finite dimension *d*.
pmuf[*n,d*]	Estimator for number of comparisons, Pareto algorithm, maximum unbalanced case, finite dimension *d*.
The next set of estimators all assume balanced operation, all points Pareto	
mff[*n,d*]	Estimator for number of function calls, marry algorithm, finite dimension *d*.
pff[*n,d*]	Estimator for number of function calls, Pareto algorithm, finite dimension *d*.
mdf[*n,d*]	Estimator total amount of data passed, Marry algorithm, finite dimension *d*.
pdf[*n,d*]	Estimator for total amount of data passed, Pareto algorithm, finite dimension *d*.
Various Estimators	
spp[*n,d*]	Estimator for numbr of comparisons with a single point Pareto, balanced data, dimension *d*.
bfm[*n,d*]	Run time estimator for brute force Marry algorithm, dimension *d.* If *d* not given, then infinite dimension.
dcbfm[*n,d*]	Run time estimator for marry algorithm that does a divide of initial problem, then brute force on subproblems.
dcbfm*k*[*n,d*]	Run time estimator for Marry algorithm that does *k* divides of initial problem, then brute force on subproblems. If *d* not given, then inifinite dimension. . If *d* not given, then infinite dimension.
bfp[*n,d*]	Run time estimator for brute force Pareto algorithm. If *d* not given, then inifinite dimension.
dcbfp[*n,d*]	Run time estimator for Pareto algorithm that divides the problem first, then does brute force on subproblems. If *d* not given, then inifinite dimension.
dcbfp*k*[*n,d*]	Run time estimator for Pareto algorithm that does *k* divides on initial problem, then does brute force on subproblems. If *d* not given, then inifinite dimension.

Acknowledgements

There are of course many people to acknowledge as playing key roles in my successful competion of this degree. I offer many thanks to my committee for their time and their comments on my thesis. Special thanks are offered to my thesis advisor, Dr. Tim Simpson, for patience in what turned out to be a long, slow process as I shifted rudder on my thesis topic multiple times.

My supervisors at the Applied Research Lab provided continual support throughout, by both allowing time away from regular tasks and by direct financial support as I approached the final months. Dr. Mark Traband, in particular, served as a member of my committee and provided continual gentle nudges as required.

Dr. Asok Ray was my initial advisor in 1992 when I first started towards my degree. I must acknowledge his influence, as he instilled a deep respect for rigor and mathematics which I believe I carried through into this current work.

We live an age of unbelievable growth in computer power and how it allows us to access and manipulate information. In particular, the symbolic mathematics tool Mathematica and the access to the research of others through the internet were crucial to my completing this thesis. So I offer my thanks to Stephen Wolfram and Tim Berners Lee, the inventors of Mathematica and the World Wide Web respectively.

My long suffering family learned to stop asking when I'd be done a number of years ago. My children have never known their dad not to be a PhD student. I greatly appreciate your patience.

Finally, to my wife Elizabeth, who accepted living on a grad student's stipend when she deserved better, and watched me drive in late to the office to work on this degree on more nights than can be counted. I couldn't have done it without you.

"If one is the master of one thing and understands one thing well, one has at the same time insight into and understanding of many things."

- Vincent Van Gogh

CHAPTER 1

Introduction

The motivation for this work stems from ongoing efforts to improve the computational design optimization process for large, complex systems such as aircraft, land vehicles, or spacecraft. In particular, this work supports a shift from computational design *optimization* to design *exploration*. Whereas past focus has been on specifying constraints and objectives for a system *a priori* and generating a "best" design or a set of best designs in accordance with those objectives (Hazelrigg 1996; Scott and Antonsson 1999; Shah and Wright 2000; Tang and Krishnamurty 2000; Allen 2001; Thurston 2001), the new paradigm is to generate thousands or millions of candidate designs across the trade space, store them in a database, and then allow users to *a postieri* express their constraints and objectives and view the best designs from the set of candidate designs (Balling 1999; Stump et al. 2002; Yukish and Harris 2002).

The effort in design exploration is supported by ongoing research in the areas of design automation, metamodeling and response surfaces, and by research in methods to visually explore design trade spaces. Multidisciplinary design optimization (MDO), where models of systems and subsystems are integrated to form a set of interacting analyses that can generate feasible design concepts, has risen to fill the need for design automation. There has been a significant body of research on methods for structuring the interacting analyses and posing their solution, with tradeoffs between ease of assembling the codes, the ability to parallelize their execution, ease of solving the posed problem, and minimizing their run time (Cramer et al. 1994; Rogers and Bloebaum 1994; Sobieszczanski-Sobieski and Haftka 1997; Alexandrov and Kodiyalam 1998; Alexandrov and Lewis 1999; Rogers et al. 1999; Alexandrov and Lewis 2000). Along with structuring the problem to minimize run time, and using efficient optimization algorithms, work in metamodeling and response surfaces has allowed one to replace codes that have run times as long as hours or days with codes that run in the order of milliseconds. These accelerated codes can be used to reduce the run time of the MDO problems by orders of magnitude (Box and Draper 1959; Jones et al. 1998; Barton et al. 2000; Simpson et al. 2001; Simpson et al. 2003).

Moore's law has continued to hold true and computing power has continued to grow at an exponential rate, further reducing computation time. Work in parallel computing and concepts such as grid computing and computer workfarms allow one to spread processes across hundreds or potentially thousands of computers to accelerate the creation of design concepts. Taken together, it has become possible to generate hundreds of thousands of design concepts in a short period of time.

A database populated with N designs, where each design is described by a set of d parameters, can be considered to be a set of points occupying a trade space of dimension d. Taking this view, one can use tools developed for *multi-dimensional data visualization* and *visual data mining* to explore the trade space (Goebel and Gruenwald 1999; Nagel et al. 2001; Keim 2002). There are a number of freely available and commercial tools (e.g., Spotfire[1], VisDB[2], Cviz[3], HighTower[4], XmdvTool[5]) that are used in industries such as drug discovery and data mining. Within each tool, one can arbitrarily choose which dimensions of the data to visualize, and assign them to 3D coordinate axes or other methods of indicating a value to see relationships and correlation between variables. One can also dynamically constrain the data by removing points that do not satisfy some easily set criteria through a process called *brushing* (Becker and Cleveland 1987).

To better support the design process, the visual data mining tools can be augmented to allow the user to express and visualize an arbitrary preference structure over the trade space (Stump et al. 2002). With few exceptions, the preference criteria will be multiobjective and there are any number of different possible criteria from which to choose. Some examples are weighting and scoring, setting targets, prioritization methods such as lexicographic goal programming, min-max criteria, and efficiency (Rosenthal 1985).

The efficiency criterion is also known as *noninferiority*, *nondominated*, *Pareto-admissibility*, and *Pareto-optimality*. A point is a member of the nondominated set, also known as the Pareto frontier, if no other point is more preferred in all attributes. Conversely, a dominated point is one for which there exists another point that is equal or better in all attributes and strictly better in at least one, e.g., a car is faster, roomier, and less costly than another.

The adoption of the efficiency criteria requires the assumption of monotonicity in each attribute, whereby it is assumed that a user's preference would never decrease as one attribute is increased while holding the others constant. Given that this assumption holds, Rosenthal (1985) states that "…a rational person would never deliberately select a dominated point. This is probably the only important statement in multiobjective optimization that can be made without the possibility of generating some disagreement."

The Pareto frontier has been long recognized as critical to design decision making, and much research has occurred and is ongoing in developing algorithms to generate points that occupy the frontier (Cheng and Li

[1] http://www.spotfire.com

[2] http://www.dbs.informatik.uni-muenchen.de/dbs/projekt/visdb/visdb.html

[3] http://www.alphaworks.ibm.com/formula/CViz

[4] http://www.hightowersecurity.com/index.html

[5] http://davis.wpi.edu/~xmdv/

1997; Messac et al. 1999; Tappeta and Renaud 1999; Tappeta and Renaud 1999; Deb 2001; Deb et al. 2002). However this work again assumes an *a priori* expression of preference and constraints, as per the design *optimization* problem. Recall that the design *exploration* problem starts with existing points and lets users arbitrarily choose dimensions to place preferences on and apply constraints. After culling the data set with constraints, the visualization tool must then identify the points that are nondominated as per the preference criteria and display them to the user.

Figure 1 shows the visualization of the Pareto frontier in the Advanced Trade Space Visualization (ATSV) software developed by a team of researchers to include this author (Stump et al. 2004). In the figure, the nondominated points are indicated in black. As users move the slider bar at the bottom of each screen, they adjust the constraints, and a new set of points are feasible. The interface must recalculate and display the Pareto frontier in real time, with large numbers of points. This turns the problem of identifying the Pareto frontier from a fixed set of points into a computing problem analogous to the sorting problem for 1D data, or identifying the convex hull for higher dimensional data (O'Rourke 1993). The need to update in real time drives the goal of efficient computation.

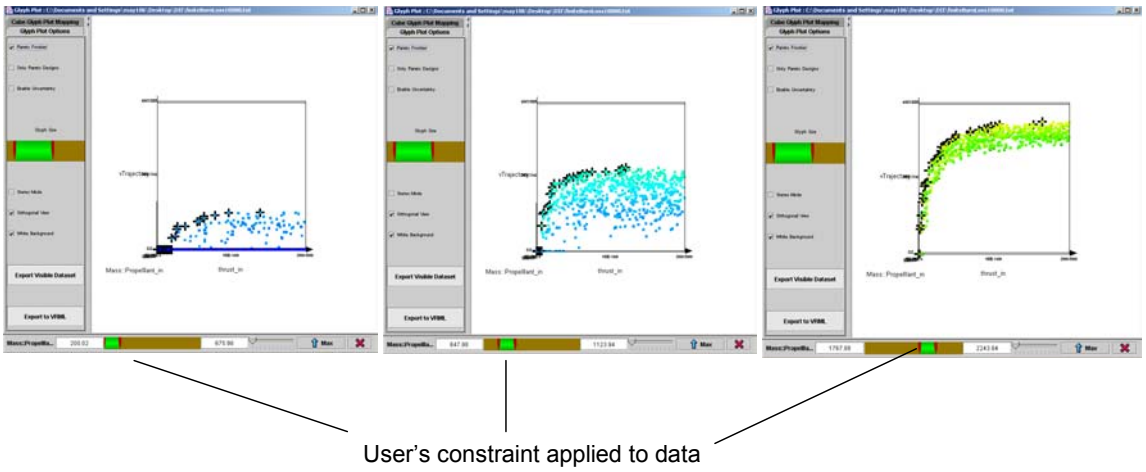

Figure 1: Visualizing the Pareto points (black crosses) as user dynamically varies the data set

The literature on algorithms for identifying the Pareto set is surprisingly sparse, consisting mainly of a three papers: (Kung et al. 1975; Bentley 1980; Bentley et al. 1990). The first two papers develop the divide and conquer algorithm that is used in this thesis. While the papers provide an order of complexity for the divide and conquer algorithm, they do not develop actual run time estimates for the algorithms needed to calculate the switch points in a hybrid version. The third paper develops an expected linear run time algorithm, but this algorithm is predicated on there being no correlation between each point's values in the different dimensions, i.e., the data is uncorrelated between dimensions. This assumption results in relatively few numbers of Pareto points as a percentage of the total data set. The assumption breaks down for data with

correlation between dimensions and also for uncorrelated data of high dimensionality. Experience with the design problems addressed in this thesis show that the uncorrelated assumption does not hold in general.

There has been recent work in this area in the constraint programming community (Gavanelli 2001; Gavanelli 2002), but the common view is that of (Ehrgott and Gandibleux 2000) who, in their annotated bibliography of multiobjective combinatorial optimization, consider the problem "efficiently solvable and not of mathematical interest." Whereas similar problems such as sorting continue to draw attention, algorithms to identity Pareto points from data sets have not. A probable reason for this view is that there has not been a requirement for rapid, continual updating of Pareto sets, as is needed in a visualization tool such as the ATSV.

Apart from visualization, as design *exploration* becomes more prevalent, large data sets will be used, and so algorithmic efficiency for computing the Pareto points will become increasingly important. There has been recent activity in the evolutionary computing literature as researchers have developed a need to efficiently identify nondominated points (Murata and Ishibuchi 1995; Zitzler and Thiele 1999; Knowles and Corne 2000; Zitzler et al. 2000; Deb 2001; Deb et al. 2002). In these works the authors have developed $O(N^2)$ algorithms to identify the nondominated points. A recent paper in the evolutionary computing literature (Jensen 2003) has identified the aforementiomed algorithms of Kung and Bentley as applicable to the evolutionary computing problem, and conducted empirical tests to demonstrate their efficacy.

Consequently, the goal in this research is to develop a fast, efficient hybrid algorithm to identify the Pareto frontier in multi-dimensional data sets. The hybrid algorithm is a blend of two different base algorithms, the Simple Cull (SC) algorithm that has a low overhead but is of overall high computational complexity, and the Divide & Conquer (DC) algorithm, as introduced by Kung(Kung et al. 1975) and developed by Bentley (Bentley 1980), that has a lower computational complexity but high overhead. The hybrid algorithm employs aspects of each of these two base algorithms, adapting in response to the properties of the data.

Each of the two base algorithms perform better for different classes of data, with the SC algorithm performing best for data sets with few nondominated points, high dimensionality, or few total numbers of points, while DC algorithm performs better otherwise. The general approach to the hybrid algorithm is to

1. First execute one pass of the SC algorithm through the data if merited
2. Execute the DC algorithm, which recursively splits the data into smaller problem sizes
3. Switch to the SC algorithm for problem sizes below a certain limit

In order to determine whether Step 1 should be executed, and to determine at what problem size the switch in Step 3 should be made, estimates of both algorithms' run times as a function of the size of the data set,

the dimension of the data set, and the expected number of nondominated points are needed. These are developed in the thesis.

To aid in increasing the speed and reducing the computational and storage complexity of the algorithm, and to enable the ability for the algorithm to adapt to the data, a canonical transformation of the data to a *Lattice Latin Hypercube* (LLH) form is developed. The transformation preserves the Pareto property of points, but reduces storage space and algorithm run time.

In order to test the algorithms, three different methods for creating randomized data sets with arbitrary dimensionality and numbers of nondominated points are developed. Each of the methods provides insight into the properties of nondominated sets, along with providing test cases for the algorithms. Additionally, a spacecraft design problem is developed to serve as a source of real world test data.

Contributions of the thesis

This thesis builds on the work of Kung and Bentley (Kung et al. 1975; Bentley 1980) and also uses many existing results from the field of number theory. The additional contributions to the body of knowledge are listed here.

1. The finite burn loss model
2. The concept of transforming the data to LLH form, and all analyses throughout the remainder of the thesis associated with the data in this form
3. The approach for proving the expected number of Pareto points in uncorrelated data of varying dimension
4. All methods for generating random data with a predetermined expected number of Pareto points
5. Development of the SC algorithm and analytical estimates of its run time for all cases
6. Analytical estimates of the run time of the DC algorithm for all cases
7. Development of the randomized DC algorithm
8. Development of the hybrid algorithm and analytical estimates of its run time for all cases
9. All experimental results

The remainder of the thesis is structured as follows. Chapter 2 develops a spacecraft design problem to further motivate the need for Pareto algorithms, and to provide test data for the algorithms. Chapter 3 develops the basic properties of data sets with respect to nondominated points. Chapter 4 develops the algorithms for creating test data. Chapter 5 introduces the basic computational framework for the Pareto algorithms, develops the SC algorithmsand presents the linear time algorithm for 2D data. Chapter 6 develops the DC algorithms. Chapter 7 develops the hybrid algorithms. Chapter 8 tests the algorithms against the spacecraft data. Chapter 9 provides conclusions and offers opportunities for future research.

CHAPTER 2

Satellite Design Problem

This chapter presents a realistic model of sufficient complexity to act as a source of data for validating the algorithms and to provide an example of the type of trade studies conducted in systems design. The model will be used in later chapters to generate data against which the Pareto algorithms will be tested.

The model is based on the literature in satellite design (Abraham 1965; Fortescue and Stark 1995; Brown 1996; Brown 1998; Wertz and Larsen 1999; Fleeter 2000). The satellite design problem allows one to choose the orbit radius, payload mass, propellant mass and main engine thrust and determine the satellite's total cost, total mass, and ΔV available over the life of the mission. In general, a design goal is to minimize satellite mass, as the launch cost is directly related to satellite mass. This would lead one to size the engine as small as possible, but an effect known as finite burn loss causes a loss in efficiency at low thrust levels, providing a counter-balancing effect. Key design variables of interest are as follows:

- Orbit radius at Mars
- Propellant mass
- Engine thrust
- Payload mass (includes everything but propulsion system mass)
- Total mass
- Total cost
- ΔV arrival acceptable for orbit insertion

The key calculations to solving the problem are the calculations for finite burn loss, tank design, and main engine design, as described in the following sections.

2.1 Finite Burn Loss

The Tsiolkowski equation (Brown 1996), is typically used for computing the propellant required to execute a maneuver that results in a change in velocity, and is

$$\Delta V = g_c I_{sp} \ln\left(\frac{M_i}{M_f}\right) \tag{1}$$

where ΔV is the difference in the arrival velocity and the orbit velocity, g_c is the gravitational constant for Earth, I_{sp} is the specific impulse of the engine and propellant combination, and M_i and M_f are the initial and final masses of the satellite. By manipulating Eq. (1), one can also solve for M_i or M_f. For example,

doing so with values of $M_i = 1140$ kg , $\Delta V = 2.8$ km/s , and $I_{sp} = 312$ seconds gives a propellant mass of 1430 kg.

Note that the mass of propellant only depends on the specific impulse, the change in velocity, and the final mass, with a lower final mass resulting in a lower propellant mass. This would compel one to choose the smallest engine possible in order to minimize the engine's contribution to satellite mass, as specific impulse is typically independent of engine size.

The Tsiolkowski equation does not, however, account for *finite burn loss*. Finite burn loss occurs when the satellite's attitude is fixed in an inertial frame over the course of a continued engine burn as the satellite enters orbit. The ideal direction for a satellite's engine to point, in order to minimize the amount of propellant needed to enter into an orbit, is tangential to the flight path to maximize the vector product of thrust with the flight path, but if the satellite's attitude is fixed in space and is not adjusted to maintain the tangential relation to the flight path, then a loss in efficiency occurs, see Figure 2. Calculating finite burn loss requires numerically solving the differential equations of motion, as the orbital dynamics are changing continually as the satellite's weight decreases and the angle with respect to the flight path varies, and orbital parameters vary.

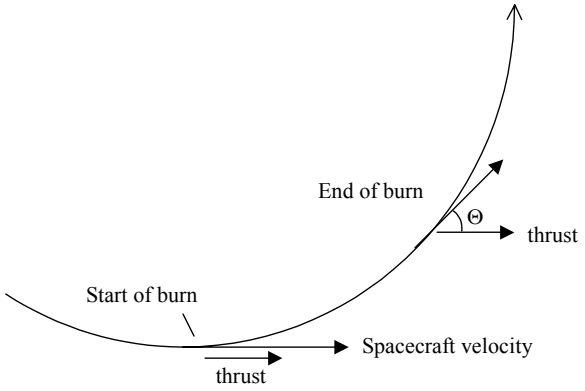

Figure 2: Finite burn loss

To solve for propellant mass required to arrive in orbit, the problem is solved backward by modeling a satellite that is *departing* orbit and is *gaining* mass as it departs, with the starting mass equal to the dry weight of the satellite. The goal is to find what the maximum arrivalΔV for a given propellant mass, dry mass, engine specific impulse and propellant mass flow rate, and orbit radius. The geometry of the problem is shown in Figure 3, with the satellite's attitude indicated by the triangle.

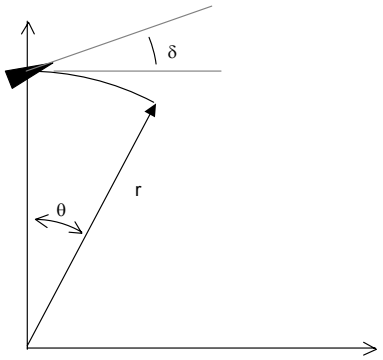

Figure 3: Problem geometry

The satellite has a fixed attitude during the departure, δ . The differential equations modeling the motion in polar coordinates are

$$\vec{F} = m\vec{a}$$
$$\vec{a} = (\ddot{r} - r\dot{\theta}^2)\hat{r} + (r\ddot{\theta} + 2\dot{r}\dot{\theta})\hat{\theta}$$
$$\vec{F} = \left(\frac{-\mu m}{r^2} + I_{sp}g_c \sin(\theta + \delta)\right)\hat{r} + \left(I_{sp}g_c \cos(\theta + \delta)\right)\hat{\theta}$$
$$m = m_{\text{final}} + \dot{m}_{\text{prop}}t$$

The set of coupled equations are nonlinear and cannot be solved for in closed-form; however, they can be solved numerically, thereby determining relationships for $r(t)$, $\dot{r}(t)$, $\theta(t)$, and $\dot{\theta}(t)$. The relationship between time and energy can be derived, $E(t) = m\left(\frac{1}{2}\left(\dot{r}^2 + r^2\dot{\theta}^2\right) - \mu / r\right)$, and the energy achieved at time t when all propellant is added obtained. From this energy, the velocity can be calculated.

Solving numerically and examining the propellant needed to achieve the desired energy increase for varying satellite masses and propellant flows shows the effect of finite burn loss for satellites of varying empty mass and a fixed radius of orbit (see Figure 4).

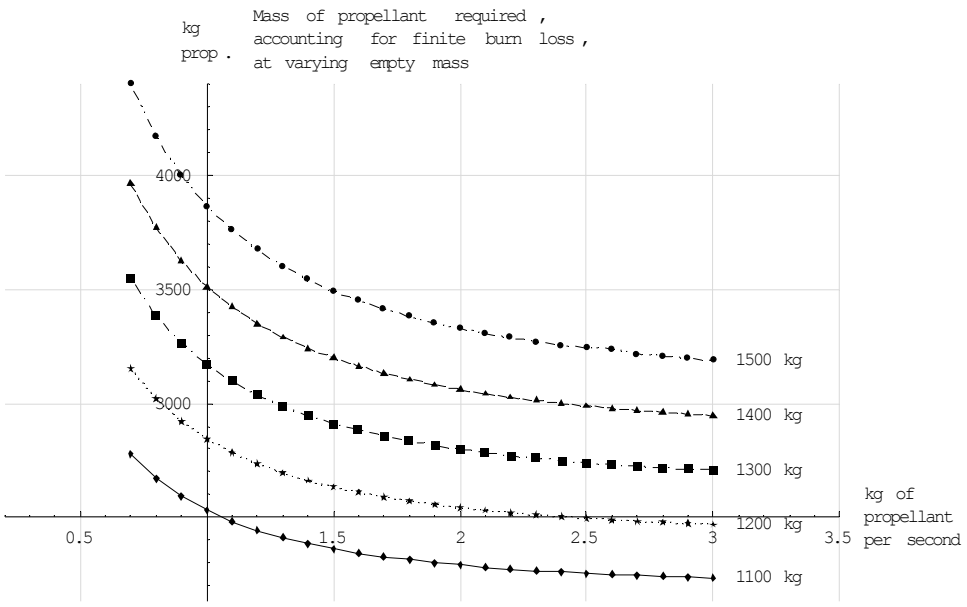

Figure 4: Finite burn loss comparison

One can see that as the propellant flow rate and therefore the thrust decreases, the propellant required for the maneuver rapidly increases. At a certain point, the maneuver cannot be accomplished at the thrust available. In the other direction, as the propellant flow rate and thrust increase, the mass of propellant required to complete the maneuver asymptotically approaches from above the idealized value determined by the Tsiolkowski equation.

Figure 5 shows how the propellant required varies with respect to propellant flow rate and specific impulse.

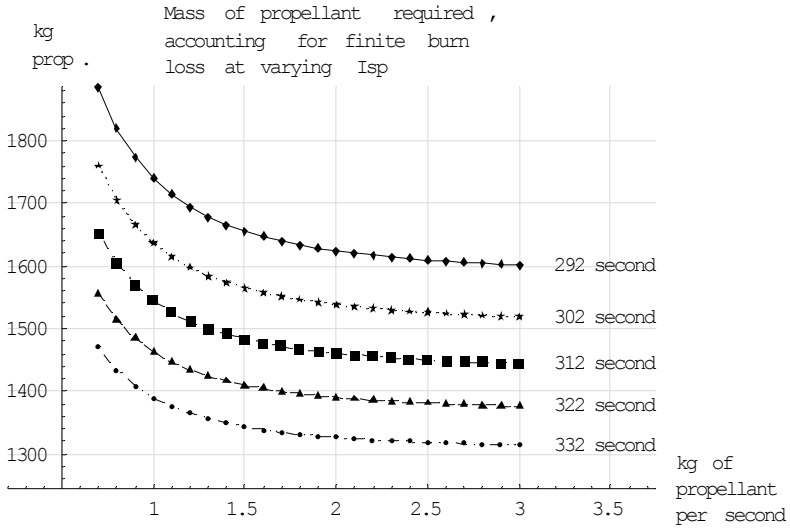

Figure 5: Finite burn loss at varying Isp

An example of an arrival trajectory computed for the case where the engine is of a very low thrust design is shown in Figure 6. The arrival trajectory approaches Mars, which is modeled by the circle, from the lower right. The satellite's thrust vector is fixed pointing left to right as indicated by the arrows. It is clear that for large portions of the trajectory the thrust vector is not tangential to the flight path, and so the approach is inefficient.

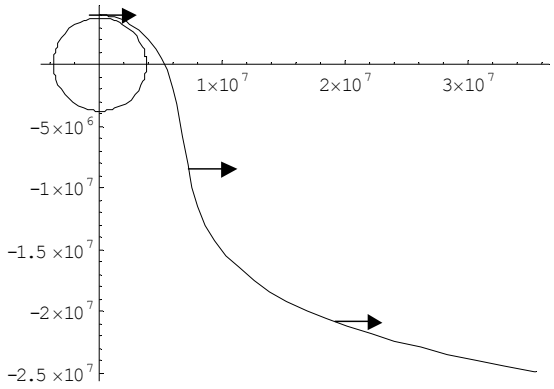

Figure 6: Low thrust trajectory

Figure 6 is to illustrate a point only. In reality, for an orbit insertion with this low of thrust, specialized guidance algorithms would be used. Table 1 shows the pertinent variables for the orbital calculation. The non-shaded variables are input, and the shaded variables are calculated. The independent variables in bold are changed for each design iteration, while the others are fixed. The bold output variables are used in other calculations or as a final value.

Table 1: Variables for the orbital calculation

Time and Propellant		
isp	312.5688139	seconds
m	4.28E+04	km^3/s^2
Dry mass	**1.08E+03**	**kg**
Propellant mass flow rate	**0.261211472**	**kg/s**
orbit radius	**4.00E+06**	**meters**
Propellant Mass	**8.00E+02**	**kg**
g	9.8	m/s^2
Time	3062.652623	seconds
vTrajectory	**0**	**km/s**
vOrbit	3272.155253	km/s
best Del	0	Radians

2.2 Tanks

The tank design rules are derived from (Brown 1996). The design rules are for tanks machined from titanium. Each tank is designed to minimize its weight, given a fixed volume of propellant and an internal pressure. The tank walls are designed to withstand the required internal pressure, but are constrained to a minimum thickness for machining considerations. The natural shape for minimum weight is a sphere, but the maximum diameter is constrained due to packaging considerations. Above a certain volume the tank will assume a barrel shape with spherical end caps, as shown in Figure 7.

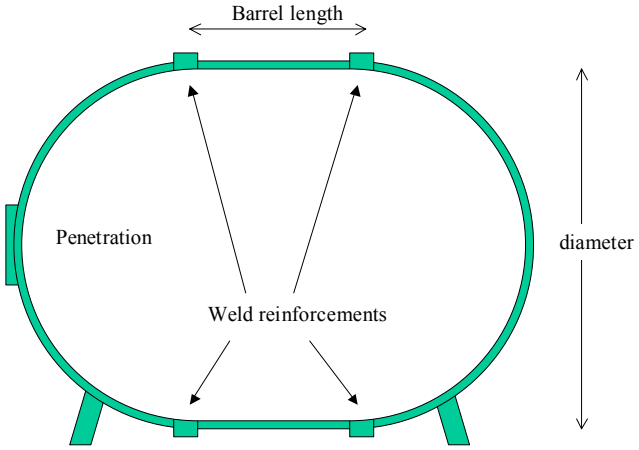

Figure 7: Tank design

The design variables for a tank are shown in Table 2. The shaded variables in the table are calculated, while the non-shaded variables are input. Most of the variables are fixed in value. Variables shown in bold are of interest, as they couple with other subsystems or are important to the system properties. The independent bold variables are varied, while the dependent bold variables are used in other calculations.

Table 2: Design variables for tank

Tank		
max pressure	700	psi
allowable stress	100000	psi
Wall density	4428	kg/m3
machine tolerance	0.0001	m
num penetrations	1	#
tank volume	**0.696**	**m^3**
max int radius	0.55	m
min Thickness	0.000254	m
weld width	0.1016	m
penetration radius	0.0254	m
internal radius	0.549760513	m
sphere thickness	0.001924162	m
barrel thickness	0.003848324	m
tank length	1.103369351	m
tank mass	**32.74763302**	**kg**
external radius	0.565153808	m
tank inertia	6.543063313	kg-m^2
cost	**1293.167642**	

The change in topology from sphere to barrel induces discontinuities in the derivative of the relation between tank volume and tank weight (see Figure 8). The first discontinuity marks the beginning of the transition from sphere to barrel shape. The second slope discontinuity marks the point at which the two weld reinforcement sections no longer overlap, as the barrel section grows in length. The tank is manufactured first as two halves of a sphere and (possibly) a barrel section, and then the sections are welded together to form the tank. A spherical tank requires one weld, while a barrel shape requires two. The welds require reinforcement, doubling the shell thickness for two inches each side of the weld.

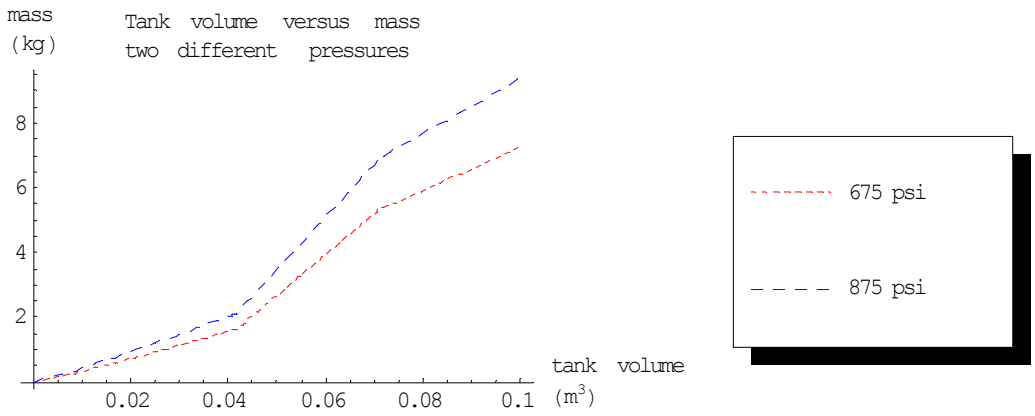

Figure 8: Tank volume versus mass

The relationship of cost for the tank also reflects the discontinuity in the first derivative of tank mass with respect to volume (Figure 9).

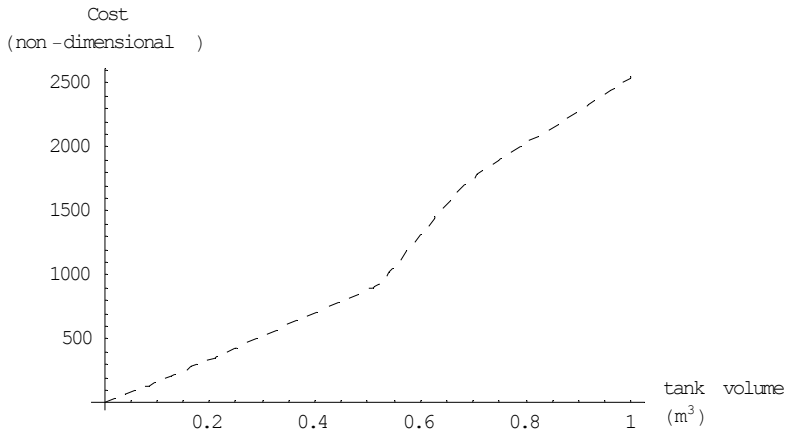

Figure 9: Tank cost-volume relationship

2.3 Engine

The main engine is considered to be a standard bi-propellant design with combustion chamber and bell-shaped nozzle (see Figure 10).

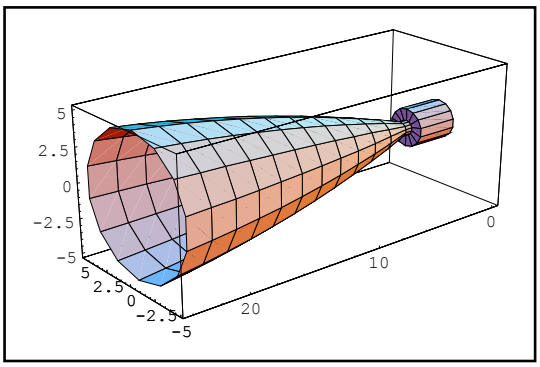

Figure 10: Main engine

The main engine design rules are derived from (Brown 1996; Sutton and Biblarz 2001). In the design rules, the weight of the engine is a direct function of only the thrust of the engine, and is slightly less than linear as shown in Figure 11.

.

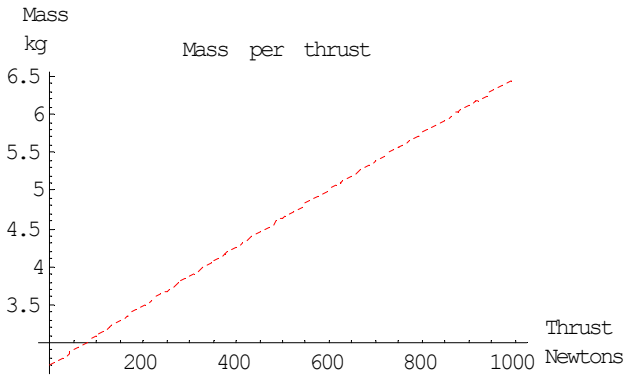

Figure 11: Engine mass per thrust

The relationship between thrust and cost is approximately quadratic as shown in Figure 12.

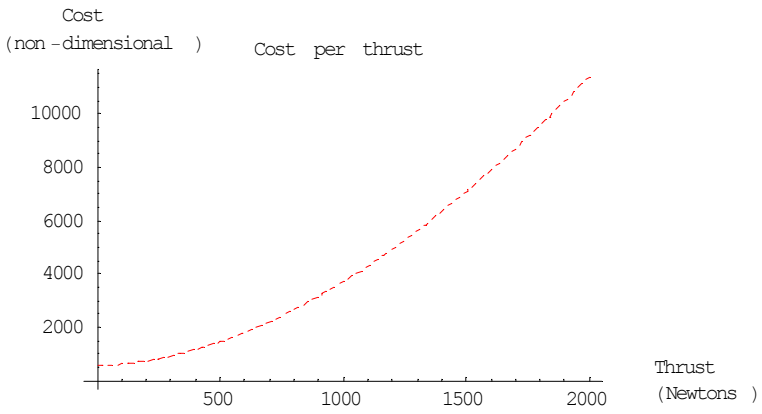

Figure 12: Engine cost-thrust relationship

The design variables for the engine calculation are shown below in Table 3. The non-shaded bold variables vary between design cases, while the other non-shaded ones are fixed. The shaded design variables are calculated, and the ones in bold are used in other calculations.

Table 3: Engine design variables

Main Engine		
thrust	**2.00E+02**	**Newton**
chamber pressure	300	psi
area ratio	90	exit/throat
cone angle	15	degrees
percent bell	0.9	fraction
chamber throat ratio	2.6	
fuel choice	N_2O_4/MMH	
ratio of specific heats	1.25	
specific gas constant	386.3522305	J/kg-K
combustion temp	3413.888889	Kelvin
mixture ratio	1.5	
engine mass	**3.484140777**	**kg**
throat diameter	0.008163282	meter
throat area	5.23383E-05	sq. meter
exit area	0.004710446	sq meter
chamber diameter	0.021224534	meter
chamber length	0.112721893	meter
nozzle length	0.116351111	meter
exit diameter	0.077443697	meter
engine length	0.233154646	meter
theoretical impulse	335.2199269	seconds
real impulse	312.5688139	seconds
exit pressure	0.178584432	psi
press ratio	1679.877673	
exhaust velocity	3193.888979	m/s
lambda	0.982962913	
propellant flow rate	**0.065302868**	**kg/s**
cost	**743.7066136**	

2.4 Problem Setup

Graphically plotting the dependencies between calculations results in Figure 13. The mass budget and cost analysis blocks merely sum their inputs to give mass and cost totals. The ΔV analysis solves the nonlinear differential equations numerically in order to determine the ΔV available to the mission.

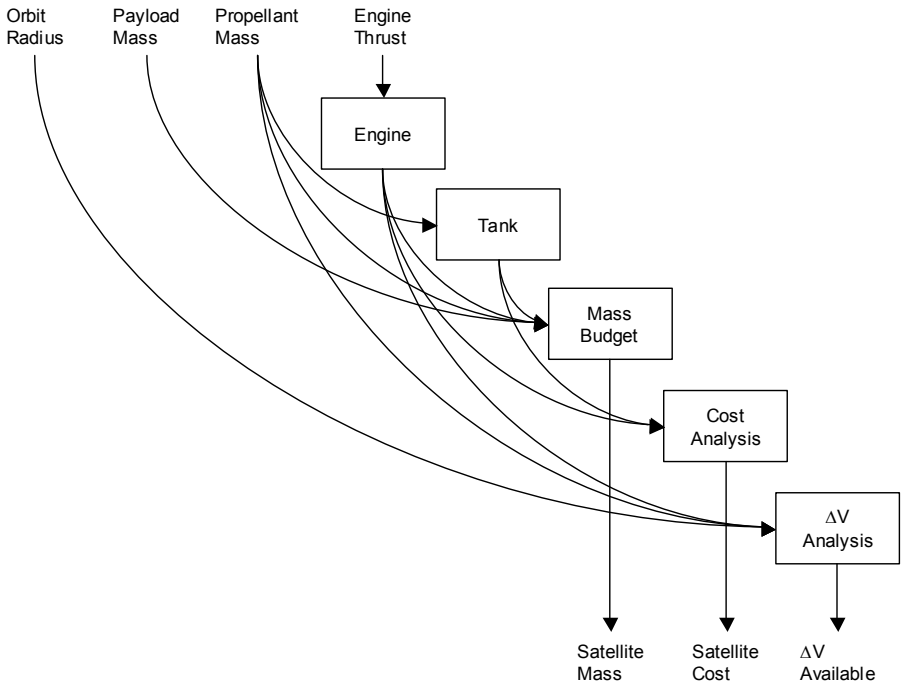

Figure 13: Calculation dependencies

Of the input and output parameters, payload mass, orbit radius, cost, total mass, and ΔV can all be considered performance parameters that a user would express a preference against. Assuming that the probe is used for remote sensing, a larger payload mass implies more sensors of greater accuracy, while a lower orbit implies better resolution. Cost and total mass are to be minimized, while ΔV available would be maximized. Propellant mass and thrust are physical design variables, which only affect the preferences through the calculations of mass, cost, and ΔV. So the design space consists of seven dimensions, with five of the seven dimensions candidates for selection in determining the Pareto frontier. As will be shown in Chapter 8, the number of Pareto points for any particular choice of dimensions and application of constraints will far exceed that for a random set of data.

2.5 Summary

The satellite model presents a typical trade space of interest to a system designer, with multiple competing objectives. The model can be exercised using Monte Carlo simulation to generate tens of thousands of points over the trade space so that a decision maker can *a postieri* set constraints, requirements, and preferences, and gain understanding of the tradeoffs between them.

CHAPTER 3

Mathematical Properties of Data Sets

This chapter introduces the basic terminology to be used in the remainder of the thesis, develops the Lattice Latin Hypercube transformation, explores some of the implications of having the data in LLH form, and develops an analytical estimate of the number of Pareto points in uncorrelated data. The development of the LLH form and its properties and the method for analytically estimating the expected number of Pareto points are new contributions to the literature.

3.1 Definition of Terms

The primary data structure used in this work is a two-dimensional table consisting of rows and columns. A table is indicated in capital bold, $\mathbf{V}$. Each row in the table equates to a single *record*, indicated in lower case bold, $\mathbf{v} \in \mathbf{V}$. A record consists of a number of *elements* corresponding to the columns of the table, and indicated in lower case $v \in \mathbf{v}$. The expression v_i indicates the *i*th element of $\mathbf{v}$, and v_{ij} indicates the *j*th element in the *i*th record in $\mathbf{V}$.

Each column in the table constitutes a totally ordered set, with the ordering indicated by the operator $\leq$. This implies the following four conditions:

1. Reflexivity: $v \leq v$ for all v in a column of $\mathbf{V}$
2. Weak antisymmetry: $v \leq w$ and $w \leq v$ implies $v = w$
3. Transitivity: $v \leq w$ and $w \leq x$ implies $v \leq x$
4. Comparability: for any v, w in a column of $\mathbf{V}$, either $v \leq w$ or $w \leq v$

A more restrictive condition adopted for this work is that within a column, each element has a unique value, i.e., there are no duplicate values in columns. So the fourth condition is changed to indicate either the condition $v \leq w$ or $w \leq v$ holds, but not both. The rationale for this restriction is given next.

Consider the design instances for a spacecraft listed in Table 4. Note that the values in each column are unique.

Table 4: Design instances

Spacecraft					
ΔV	Duration	Mass	Diameter	Length	Cost
25,000	13000	2000	13.2	230	1,000,000
22,400	12700	2543	13.1	229	1,200,000
28,250	14075	2100	14.7	215	950,000

The table can also be considered to represent a collection of *points* in a multi-dimensional space. For example, if the table has N rows and d columns, then it can be considered to represent N points that are located in a d-dimensional space (e.g., Table 4 shows three designs in a six dimensional space). In this view, a column may also be referred to as a *coordinate* or a *coordinate axis*. Either view, as a table or as a multi-dimensional space, has its unique advantages in terms of understanding the concepts of nondominance and the analysis of the Pareto algorithms, and both views are used at different times where advantageous.

3.2 Transformation to Lattice Latin Hypercube Form

Given a table constituted of N records $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_N\}$ where each $\mathbf{v} = \{v_1, \cdots v_d\}$ has d elements, one can define a transformation T that operates on the set $\mathbf{V}$ such that it generates a new table $\mathbf{Z} = T(\mathbf{V})$ which preserves the total ordering among the elements of each column as follows. For each element v_i in $\mathbf{v}$, set the value for the corresponding element z_i in $\mathbf{z}$ to the position of v_i in the order statistic for the ith column where the most preferred values come first in the new columns of $\mathbf{Z}$. In short, replace each element with where its position would be if the column were to be transformed into a sorted list. For Table 4, if transformed by T it would appear as shown in Table 5.

Table 5: Transformed design instances

Spacecraft					
ΔV	Duration	Mass	Diameter	Length	Cost
2	2	1	2	3	2
1	1	3	1	2	3
3	3	2	3	1	1

Since the values in each column of Table 4 are by definition unique, $\mathbf{V}$ can be unambiguously transformed into $\mathbf{Z}$. Each column in $\mathbf{Z}$ consists of a permutation of the list of numbers $\{1, 2, \cdots, N\}$, and so the collection of points in $\mathbf{Z}$ can be considered to be N points in a d-dimensional grid, where each possible

hyperplane in the grid has exactly one point located in it. It is this property that gives rise to the term *lattice latin hypercube*. At times, the nomenclature L_N is used to indicate a lattice with N points.

3.3 Definition of Pareto Ordering and Nondominance

Given two records $\mathbf{v}, \mathbf{w} \in \mathbf{V}$ where $\mathbf{V}$ has N records each with d elements, the record $\mathbf{v}$ is said to *dominate* $\mathbf{w}$ if for every $i \in [1, d]$, $v_i \leq w_i$, with a strict inequality for at least one attribute. If no other record in $\mathbf{V}$ dominates $\mathbf{w}$ it is considered to be *nondominated*. Another method of identifying $\mathbf{w}$ as nondominated is to refer to it as a *Pareto* point.

The transformation T preserves the Pareto ordering by definition. Figure 14 shows an example of the conversion from a continuously sampled space on the left to the lattice on the right. The assumption is that lower values of each coordinate axis are preferred.

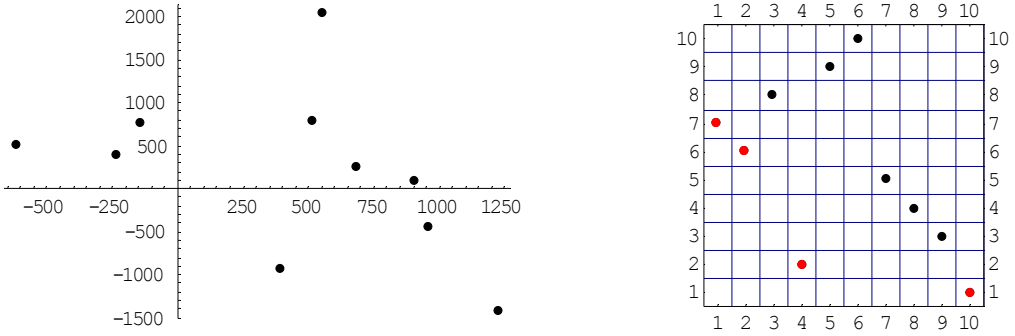

Figure 14: Conversion to lattice

Through the remainder of this thesis, an assumption is that the data has been transformed into this LLH form. The advantages for doing so are as follows:

1. All further manipulations are with integer variables vice floating point numbers, speeding up algorithms and minimizing memory use
2. Sorting algorithms against the data or subsets of the data can be implemented in linear time (Cormen et al. 1994)
3. The LLH form allows one to analyze the data prior to determining the Pareto points, to best choose an algorithm to use

The key disadvantage for manipulating the data is that, in the event that two points share the same value in one of their dimensions, the transformation will arbitrarily place an order on them. This can be handled by proper bookkeeping in the algorithms to identify the Pareto frontier, via methods sketched out in Section 9.3.2.

3.4 Properties of LLH Form for 2D Case When All Points are Pareto

For the 2D case with data converted to LLH form, an interesting property arises that if all points in the data set are Pareto, then they are all located on the diagonal only. To prove this, assume there are N points in the lattice, with each row and each column containing only one point. Define an operator S such that for a record $\mathbf{z}$, the operation $S(\mathbf{z}) = z_1 + z_2 + \cdots +, z_d$, i.e., the operator sums the elements of $\mathbf{z}$. The set of points that satisfy $S(\mathbf{z}) = N + 1$ defines the points on the diagonal. Figure 15 shows a lattice $\mathbf{Z}$ with $N = 8$, and it shows three sets, A, B and C. Let $|A|$ be the number of points in the set A, and similar for $|B|$ and $|C|$. In Figure 15, the preferred points are in the lower left corner. Therefore, to prove that all points Pareto implies all points on the diagonal, we need to show that if a point $\mathbf{z}$ is below the diagonal, then there must be at least one point in the set C, i.e., $\mathbf{z}$ dominates at least one other point.

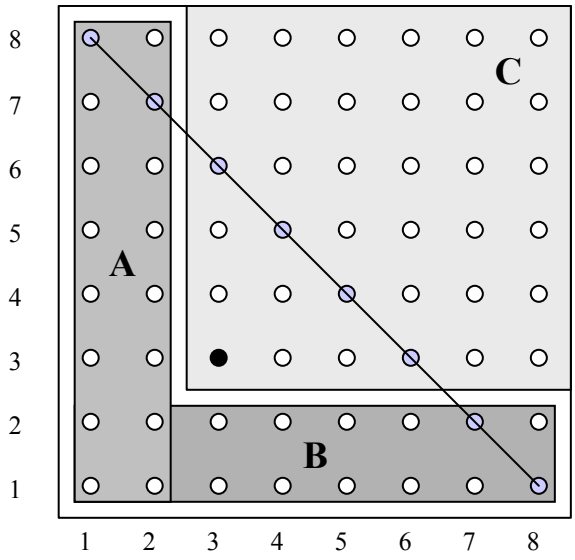

Figure 15: Lattice segmented into three disjoint sets A, B, and C

The point $\mathbf{z}$ satisfies the relation $S(\mathbf{z}) < N + 1$ since $\mathbf{z}$ is below the diagonal. Not including $\mathbf{z}$ leaves $N - 1$ points remaining in $\mathbf{Z}$. In order for $\mathbf{z}$ to not dominate another point, the remaining points must reside in the sets A and B. The number of points in the set $A \cup B = |A \cup B|$ has the relation

$$\begin{aligned} |A \cup B| &= |A| + |B| - |AB| \\ &\le |A| + |B| \end{aligned} .$$

Since every column has one and only one point, similar for every row, then from Figure 15 it can be seen that $|A| = z_1 - 1$ and $|B| = z_2 - 1$, and

$$\begin{aligned} |A \cup B| &= (z_1 - 1) + (z_2 - 1) \\ &= (z_1 + z_2) - 2 \\ &< N - 1 \end{aligned} .$$

Therefore the set C must have at least one point, and $\mathbf{z}$ dominates at least one point. A similar argument shows that for any point above the diagonal, the point must be dominated.

The implication of this result is that a potential algorithm for identifying Pareto points is to first convert the data to LLH form and then drop all points above the diagonal, since they are dominated by definition, and repeat until no further points are below the diagonal.

3.5 Diagonal Property at Higher Dimensions

Unfortunately, the 2D diagonalizing algorithm does not work for higher dimensions, since it is possible to have dominated points above the diagonal plane defined by (for the 3D case) $S(\mathbf{z}) = N + 2$. For a *d*-dimensional space the diagonal for which all points below it dominate at least one other point is defined by the set of points $\mathbf{z}$ where $S(\mathbf{z}) = N + (d-1)$. The expression $d-1$ in the equation comes from the fact that each axis is numbered 1 to N. That this relation is true can be shown as follows, where A_i is a set of points such that their *i*th coordinate is less than z_i :

$$\begin{aligned}\left|\bigcup_{i=1}^{d} A_i\right| &\le \sum_{i=1}^{d} |A_i| \\ &= \sum_{i=1}^{d} z_i - 1 \\ &= S(\mathbf{z}) - d \\ &< N + (d-1) - d \\ &= N - 1.\end{aligned}$$

The upper bounding diagonal plane to ensure a point above it is dominated is, however, not the same as the lower bounding diagonal plane. The upper bounding diagonal is determined by the relation

$$\begin{aligned}\bigcup_{i=1}^{d} A_i &\le \sum_{i=1}^{d} N - z_i \\ &= dN - S(\mathbf{z})\end{aligned} \quad .$$

Setting the right side equal to $N + (d-1)$ and solving gives

$$S(\mathbf{z}) = (d-1)N + (d-1) .$$

So only in the case of $d = 2$ are the lower diagonal and upper diagonal hyperplanes coincident. For higher dimensions, there is a gap between them, as shown in the 3D case in Figure 16.

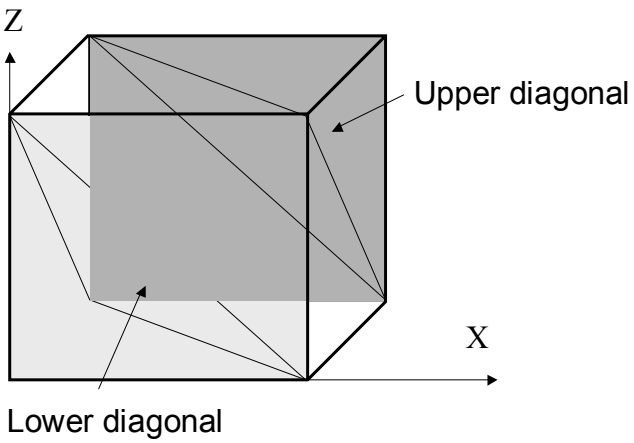

Figure 16: Bounding hyperplanes

This gap can contain points that are dominant and points that are nondominant. It is easy to show that the probability that points are below the hyperplane is approximately proportional to the content contained in the simplex determined by the hyperplane (equal to the hypercube's content multiplied by $1/d!$). So for a 3D case the volume under the plane is approximately $1/6$ of the total volume of the cube. An algorithm that repeatedly removes points below the diagonal will only work for the 2D case.

3.6 Expected Number of Pareto Points in Random Lattices

This section explores the properties of tables that have the values of the columns distributed randomly, with no correlation between the columns. Expressions for the expected number of Pareto points are calculated for 2D first. The 2D result is then used as the basis for calculating the expected number of Pareto points in higher dimension data sets.

3.6.1 Random 2D Lattice

The method for determining the expected number of Pareto points for a 2D data set is based upon a certain method of construction for creating an N point lattice. Assume that an N point lattice has been created and is augmented to create an $N+1$ point lattice as in Figure 17. For this section, assume that the most preferred point is in the top right corner. This new point is added from the bottom, with a column inserted in one of $N+1$ possible locations, as in Figure 17. The figure shows on the left a 2D lattice with $N=5$, the occupied positions as gray circles and the Pareto points circled in bold. The 5 point lattice is extended to a 6 point lattice by adding a point to the bottom, inserting its column in one of 6 possible locations, with equal probability given to the 6 possible locations. On the right is the 6 point lattice created by inserting the new point, shown in green. In this case, the new point is dominated.

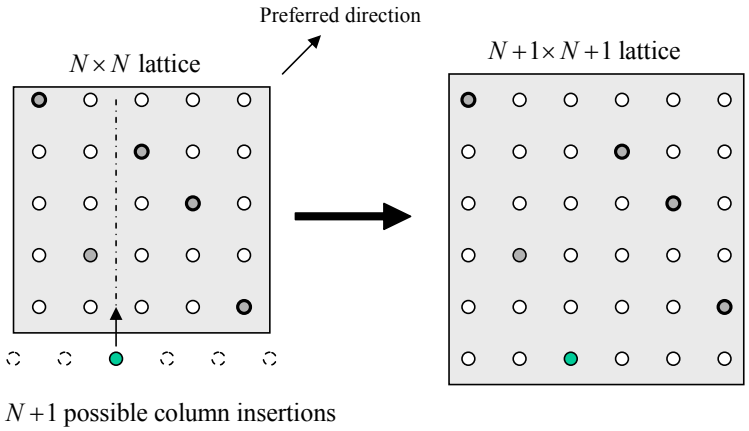

Figure 17: Construction of lattice

Since every column in the N lattice has an element in it, the only position for the new point such that it will be a Pareto point is the rightmost position, and therefore the probability that it is nondominated is $1/(N+1)$, and the expected number of nondominated points in the $(N+1)th$ row is $1/(N+1)$. Summing up over all of the rows gives the expression for expected number of Pareto points as

$$e(n) = \sum_{k=1}^{n} \frac{1}{n} \tag{2}$$

which is the definition of the harmonic number H_N (Conway and Guy 1996). Recall that the harmonic number asymptotically approaches for large N the relation

$$H_n \sim \ln n + \gamma + \frac{1}{2n}$$

where $\gamma = 0.57721...$ is the Euler-Mascheroni constant. So in general, the expected number of Pareto points in a randomly selected N lattice is proportional to $\ln N$.

Figure 18 shows the result of experimental determination of the expected number of Pareto points versus the analytical calculation via Eq. (2). The experiment considered data sets with size ranging from 1 to 100. At each size, 60 experiments were conducted. Each experiment consisted of first generating a random data set of the appropriate size, then calculating the number of Pareto points in the data set. The mean number of Pareto points per size is plotted. The figure shows agreement between the analytical and experimental.

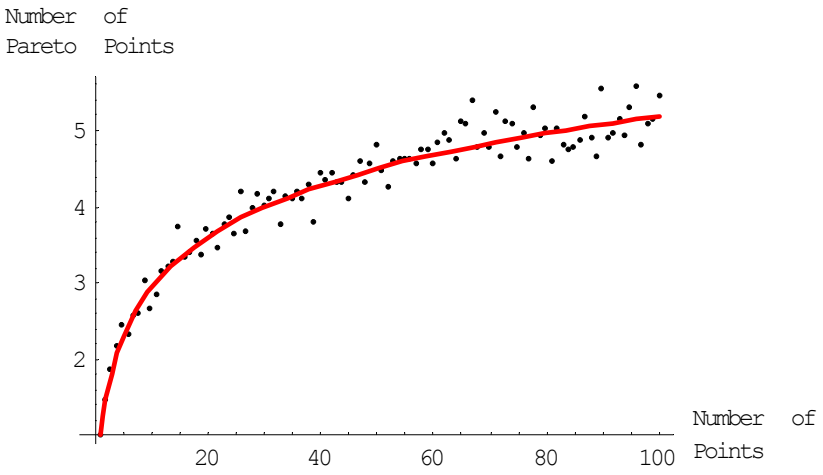

Figure 18: Experimental and Analytical results for expected number of Pareto points in 2D problem

The lattice in Figure 19 can be constructed where rows are added from below, but any alternate method that can successfully generate all of the possible permutations of lattices with equal probability to any possible permutation must also yield the same result. Consider a method where points are added at random positions in both row and column. This method must result in the same expected number of Pareto points, which is H_N. A question to ask is what is the probability, when inserting a new point with randomly chosen row and column into a random lattice, that it is a Pareto point? The answer to this question is a key to calculating the expected number of Pareto points in 3D and higher dimension lattices.

The key point to note in answering the question is that while it is possible to have two different N-1 point lattices, and insert a point into the same row and column of both, and end up with two new lattices that are identical, the position of the most recently inserted point will be different in each case. Because of this property, the probability that any one Pareto point in an N point lattice was the most recently inserted point is the same for all points, and so the probability that the most recently inserted point is Pareto is H_N / N.

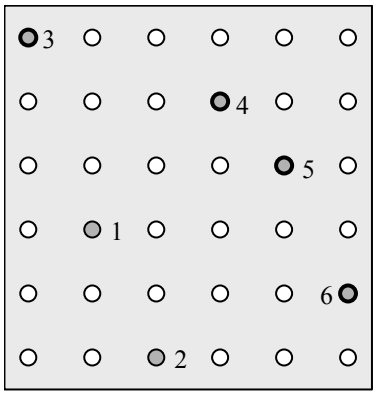

Figure 19: Pareto set

3.6.2 Expected Number of Pareto Points at Higher Dimensions

Computing the expected number of Pareto points at higher dimensions builds on the previous section's results. In the 2D case, a bottom-up construction for building up L_N was used that ensured all possible L_N are equally likely, and that was easy to use in computing the expected number of Pareto points. This section shows how to compute the analytical estimate of the number of Pareto points for the 3D case, and how to extrapolate to higher dimensions.

In the 3D case, again new points are inserted from the bottom. But whereas in the 2D case insertion from the bottom meant picking one point from a 1D line of N possible points, in the 3D case it means picking one point from a 2D *plane* of N^2 possible insertion points, as shown in Figure 20.

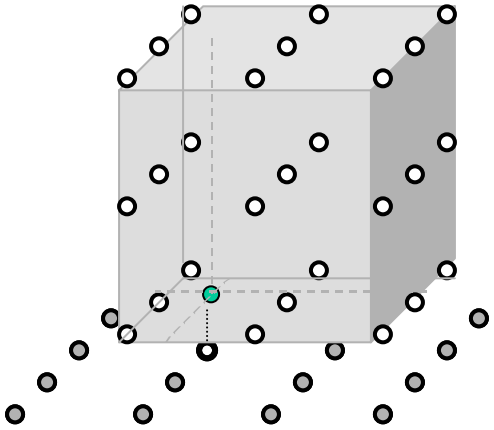

Figure 20: Points inserted from below in 3D lattice

This new point, placed at the bottom plane of the current cube of points, is a Pareto point in the cube if and only if it is a Pareto point in the 2D problem formed by projecting all points onto a plane parallel to the bottom plane. If it is Pareto in the N lattice, it remains Pareto in $N+1$ and higher lattices, as points that are inserted below it can not dominate it. As this problem is identical to the one in the previous section, the probability that the added point is Pareto is H_N / N. Building up to a 3D L_N, the expected number of Pareto points is then

$$E[\text{pareto points in 3D latttice}] = \sum_{i=1}^{N} \frac{H_i}{i} = \sum_{i=1}^{N} \frac{1}{i} \sum_{j=1}^{i} \frac{1}{j}.$$

This can be solved in closed-form (Knuth 1997) to be equal to

$$\sum_{i=1}^{N} \frac{1}{i} \sum_{j=1}^{i} \frac{1}{j} = \frac{1}{2}\left({H_N}^2 + {H_N}^{(2)} \right)$$

where the generalized term ${H_N}^{(k)}$ is defined as

$$H_N^{(k)} = \sum_{i=1}^{N} \frac{1}{i^k} .$$

While the sequence H_N grows without bound for larger N all of the sequences $H_N^{(k)}$ for $k > 1$ do converge. Note that $H_N = H_N^{(1)}$. For example,

$$\lim_{N\to\infty} H_N^{(2)} = \frac{\pi^2}{6} .$$

Again, experiment corroborates the analytical approach as shown in Figure 21. For the 3D experiment, size was varied from 1 to 100, with 60 random lattices per size, and the mean value of the number of Pareto points per size is shown. The solid line is the analytical estimate.

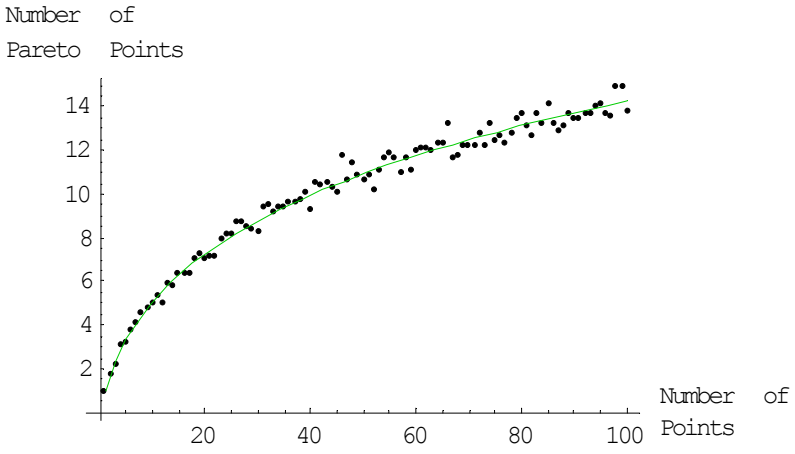

Figure 21: Expected number of Pareto points for 3D problem, with analytical estimate as the solid curve, and experimental values as the points

The expected number of Pareto points for a 4D problem is, by generalizing the 3D case, equal to

$$\begin{aligned} E[\text{Pareto points in 4D lattice}] &= \sum_{i=1}^{N} \sum_{j=1}^{i} \frac{H_j}{j} \\ &= \sum_{i=1}^{N} \sum_{j=1}^{i} \sum_{k=1}^{j} \frac{1}{ijk} \end{aligned}$$

and the expected number of Pareto points in d dimensions is

$$E[\text{Pareto points in lattice of dimension } d] = \sum_{i_1=1}^{N} \sum_{i_2=1}^{i_1} \cdots \sum_{i_{d-1}=1}^{i_{d-2}} \left(\frac{1}{i_1 i_2 \cdots i_{d-1}} \right).$$

All of these expressions can be converted into combinations of generalized harmonic numbers (Knuth 1992). Some of the simplifications are listed in Table 6.

Table 6: Relationship of dimension of random lattice to expected number of Pareto points in terms of generalized harmonic numbers

Dimensions	Expression for the expected number of Pareto points
2	H_N
3	$\frac{1}{2!}\left(H_N{}^2 + H_N{}^{(2)}\right)$
4	$\frac{1}{3!}\left(H_N{}^3 + 3H_N + 3H_N{}^{(2)}\right)$
5	$\frac{1}{4!}\left(H_N^4 + 6H_N^2 H_N^{(2)} + 3H_N^{(2)2} + 8H_N^2 H_N^{(3)} + 6H_N^{(4)}\right)$
6	$\frac{1}{5!}\left(H_N^5 + 10H_N^3 H_N^{(2)} + 15H_N H_N^{(2)2} + 20H_N^{(2)} H_N^{(3)} + 20H_N^2 H_N^{(3)} + 30H_N H_N^{(4)} + 24H_N^{(5)}\right)$
7	$\frac{1}{6!}\begin{pmatrix} H_N^6 + 15H_N^4 H_N^{(2)} + 40H_N^3 H_N^{(3)} + 90H_N^2 H_N^{(4)} + 144H_N H_N^{(5)} + \ldots \\ +120H_N H_N^{(2)} H_N^{(3)} + 40H_N^{(3)} H_N^{(3)} + 15H_N^{(2)} H_N^{(2)} H_N^{(2)} + 90H_N^{(2)} H_N^{(4)} + \ldots \\ +120H_N^{(6)} \end{pmatrix}$

While the expressions in Table 6 can be written in terms of combinations of various $H_N^{(k)}$, there is no simple closed-form solution for the expression that directly emerges. Examining the 5D case and generalizing shows why this is true. First, rewrite the terms $H_N{}^k = \left(H_N\right)^k$ as $H_N^{(1)} H_N^{(1)} H_N^{(1)} \cdots H_N^{(1)}$ where $H_N^{(1)}$ is multiplied together k times. Now rewrite terms such as $H_N^{(1)} H_N^{(1)} H_N^{(2)} H_N^{(3)}$ as $H_N^{(1,1,2,3)}$ for simplification. Using the modified notation, the 5D case can be written as

$$\frac{1}{120}\left(H_N^{(1,1,1,1,1)} + 10H_N^{(1,1,1,2)} + 15H_N^{(1,2,2)} + 20H_N^{(2,3)} + 20H_N^{(1,1,3)} + 30H_N^{(1,4)} + 24H_N^{(5)}\right) \quad (3)$$

and similar for the other dimensions. Observation reveals that the terms in the superscript are all of the possible summations of integers less than or equal to 5 such that their sum is 5. This applies for higher dimensions too. The number of possible summations such that they equal a number N is the well-known partition function, $P(N)$, and this function itself does not have a closed-form solution even in terms of the number of combinations N, much less the particulars of the combinations (Conway and Guy 1996). However, the number of Pareto points does scale in proportion to $\ln^{d-1} N$ in the limit for random data of dimension d, as can be seen by replacing the H_N terms with their logarithm approximation.

Figure 22 shows results of experimental and analytical calculations for the expected number of Pareto points for lattices of size ranging from 1 to 100, and dimension ranging from 2 to 9. The solid lines are the

analytical calculations, while the points represent experimental results. The experiments were conducted as per the 2D and 3D case described previously.

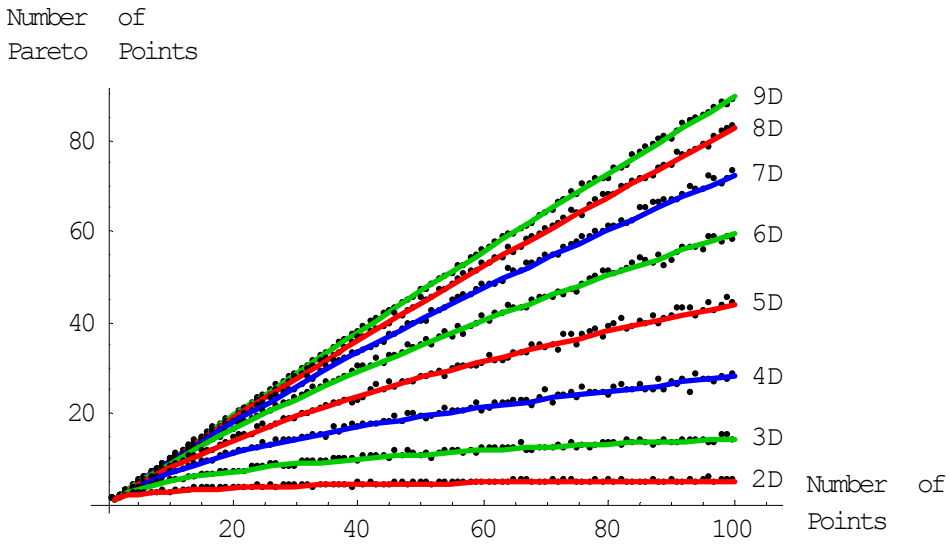

Figure 22: Analytical and experimental results for expected number of Pareto points for lattices ranging in size from 1 to 100, and dimension from 2 to 9.

3.7 Expected Number of Points Dominated by "Best" Pareto Point in the 2D Problem

Given a random data set in 2D, each point will dominate zero or more other points. Label the point in the data set that dominates the most other points as p', and the number of points it dominates as D^*. Then a useful result for computing expected run times of the SC algorithm, and in setting the break point in the hybrid algorithm, is to determine the expected value of D^*.

To compute this, first note the following form of 2D correlated lattice, where the available cells in the lattice are truncated in the preferred direction (up and right in Figure 23). The critical dimensions of the lattice are the size of the lattice N and the length of the top row M. The positions available to be occupied are shown in black, while the positions that will remain empty are shown grayed out for reference. The positions along the diagonal are shown in bold.

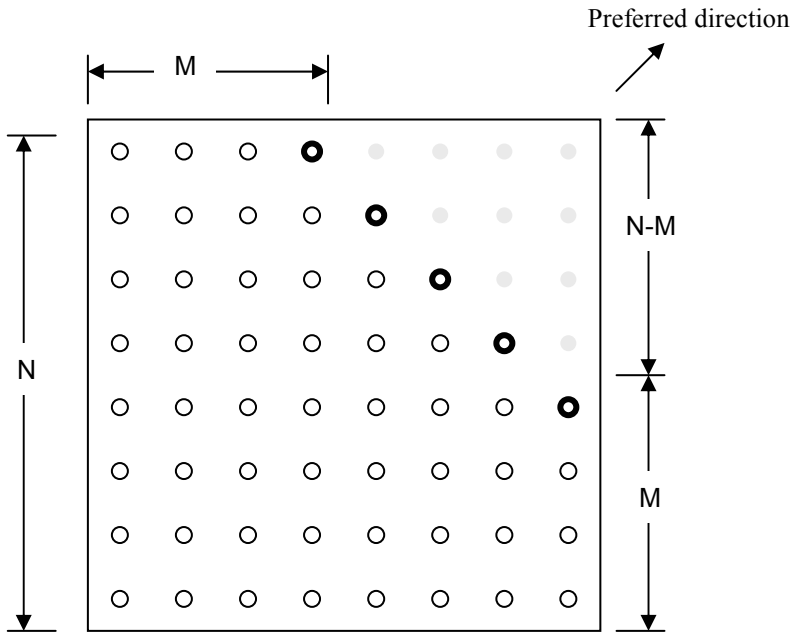

Figure 23: Correlated lattice

The number of permutations possible in the lattice is

$$M^{N-M}M! \tag{4}$$

That this is true can be seen by noting that options for placement of a point in the top row are M, and options for the next row down are also M, and so on down through the first $N-M$ rows. The final M rows have $M!$ placements, resulting in Eq. (4). The probability that a lattice is randomly generated that meets this restricted form is, assuming all permutations are equally likely

$$\frac{M^{N-M}M!}{N!}. \tag{5}$$

This probability includes cases where a lattice has no points occupying the bold positions along the diagonal in Figure 23. Narrowing the probability to cases where the lattice meets the constraints of Figure 23 and has at least one bold position occupied results in

$$\frac{M^{N-M}M!-(M-1)^{N-M+1}(M-1)!}{N!}. \tag{6}$$

As shown in Sections 3.4 and 3.5, for a 2D problem of size N, any point with coordinates X and Y will leave a maximum of $2N-(X+Y)$ points nondominated, and so dominate a minimum of $N-(2N-(X+Y))-1$ points, which simplifies to $(X+Y)-N-1$. The number D^* is then expressed as

$$D^* = \max_i \left(X_i + Y_i - N - 1\right) \tag{7}$$

To say that a particular lattice has a value of D^* is the same as saying the lattice can be cast in the form of Figure 23, i.e., a truncated lattice with value $M = D^* + 1$ and that at least one position along the upper cut diagonal (bold points) is occupied. The probability that this occurs, for a random lattice, is

$$P[D^* = M - 1] = \frac{M^{N-M} M! - (M-1)^{N-M+1} (M-1)!}{N!} \tag{8}$$

The expected value for the D^* is the expression

$$E[D^*] = \sum_{M=1}^{N} (M-1) \frac{M^{N-M} M! - (M-1)^{N-M+1} (M-1)!}{N!} \tag{9}$$

.

This can be manipulated algebraically as follows. Define Eq. (5) as $f(M)$ so that

$$E[D^*] = \frac{1}{N!} \sum_{M=1}^{N} (M-1) \left[f(M) - f(M-1) \right]. \tag{10}$$

This can be manipulated and simplified using the relations $f(N) = 1$ and $f(0) = 0$ to give

$$E[D^*] = N - \sum_{M=1}^{N} \frac{M^{N-M} M!}{N!}. \tag{11}$$

A closed-form approximation to the sum can be developed using the following relation (Knuth 1997):

$$\sum_{k=0}^{n} \frac{(n-k)^k (n-k)!}{n!} = \sqrt{\frac{\pi n}{2}} - \frac{2}{3} + \frac{11}{24} \sqrt{\frac{\pi}{2n}} + \frac{4}{135n} - \frac{71}{1152} \sqrt{\frac{\pi}{2n^3}} + O\left(n^{-2}\right),$$

which leads to the expression

$$E[D^*] \simeq N - \sqrt{\frac{\pi N}{2}} - \frac{2}{3} + \frac{11}{24} \sqrt{\frac{\pi}{2N}} + \frac{4}{135N} - \frac{71}{1152} \sqrt{\frac{\pi}{2N^3}} \tag{12}$$

.

Plotting the ratio of $E[D^*]$ to N in Figure 24 shows that, for randomly distributed 2D data, the best point will dominate a proportion of the other points, with that proportion rapidly approaching 1. The solid line is the approximation, and the points are exact solutions to the summation, to show their correlation.

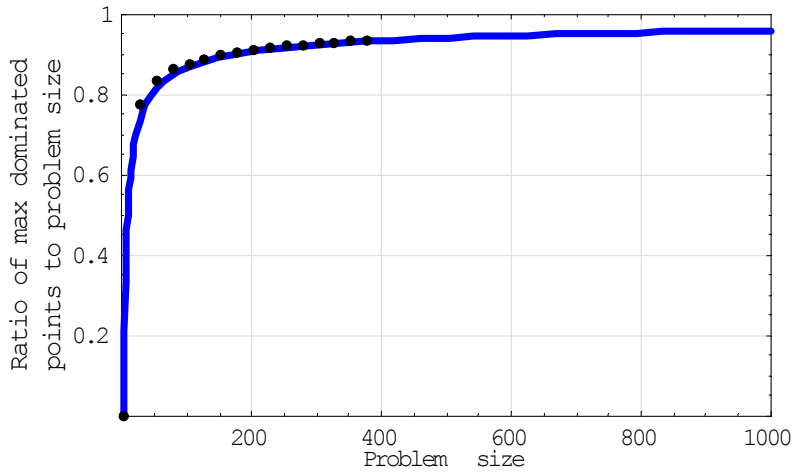

Figure 24: Ratio of problem size to number of points dominated by the "best" Pareto point for a 2D lattice

What this implies is that, for a 2D table randomized with no correlation between columns, if one first identifies the point p' and uses it to cull the remaining points to remove those that are dominated by p', one can expect to remove most of the points on the first pass. As this is what the SC algorithm does, the SC algorithm proves to be very efficient for random 2D lattices, as shown later.

This general relation diminishes for higher dimensions, although for a fixed dimension the ratio does continue to increase with problem size. Figure 25 shows the results of empirical computer experiments used to determine the relationship between lattice size, dimension, and expected minimum number of points dominated for lattices of dimension three through six. The 2D curve from Figure 24 is also shown for reference.

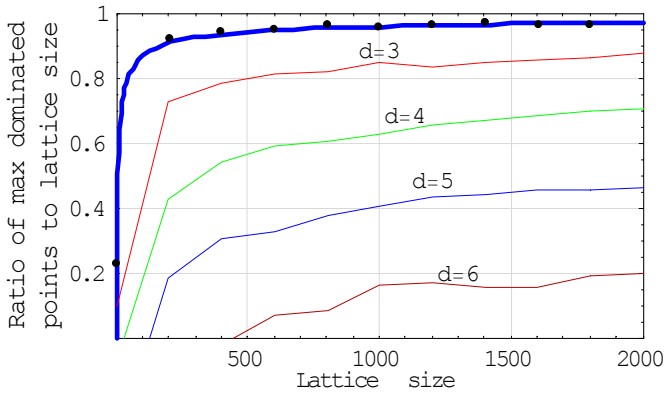

Figure 25: Varying ratios of points dominated

Figure 26 is a plot showing how, for 100,000 points, the ratio of minimum expected dominated points to total table size decreases with increasing dimension. So at high dimension with random data sets, the SC algorithm's performance will suffer.

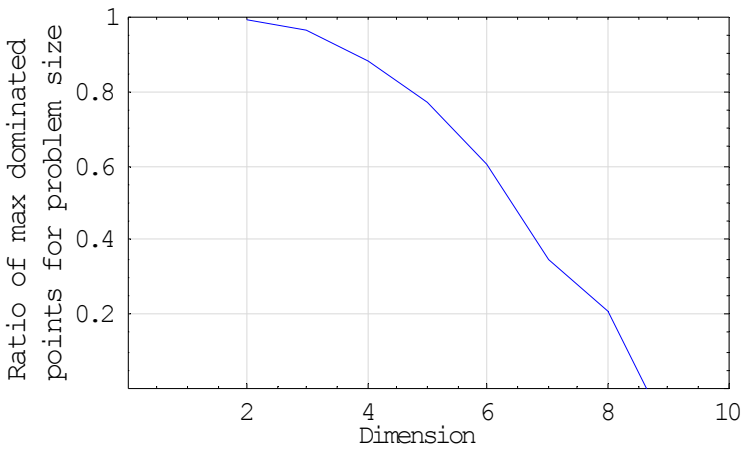

Figure 26: Varying ratio of points dominated for a problem size of 100,000 and varying dimension

3.8 Summary

This chapter introduced the basic mathematical nomenclature that is used throughout the remainder of the thesis and also introduced the transformation of the data to LLH form. The transformation preserves the Pareto partial ordering between points, while reducing required storage space and sort time. In the following chapters the LLH form is used in optimizing the design of the algorithms and in developing methods to analyze the data in real time to improve algorithm performance.

The chapter also developed a new method for developing an analytical estimate of the number of Pareto points for random lattices of arbitrary dimension. The analytical estimate agrees with experimental results, showing that the expected number of Pareto points scales by $\ln^{d-1} N$.

For random lattices, one can ask what the expected value for the maximum number of points dominated by any one point in the lattice is. Analytical results for the 2D case and empirical results for the higher dimensional cases show that, as lattice size increases, the ratio of total size to maximum number of points dominated by any one other point is monotonically increasing. However, the growth in this ratio becomes very slow as the dimension increases.

CHAPTER 4

Generating Test Data

This chapter introduces methods for creating test data. The literature on algorithms for identifying the Pareto frontier has to date focused only on random data where the distribution of data in each dimension is independent (Bentley et al. 1990). This independence between dimensions creates data sets that are not realistic for actual design problem sets. Using such random data can lead one to choose an algorithm that performs well against the random independent data but poorly otherwise.

There are in general four types of test data that are used in the testing algorithms for determining the Pareto points: best case, worst case, random uncorrelated case, and random correlated case. The first two, best case and worst case, are dependent on the algorithm to be tested and are covered in the chapters related to that particular algorithm. The random uncorrelated case is straight forward to generate, therefore, this chapter focuses on the random correlated case. All three methods for generating data are new to the literature.

4.1 Partitioned Lattice Method

The uncorrelated random lattice has a fixed relationship between the number of points N in the lattice and the expected number of Pareto points. One would like to generate lattices that are random, yet have a selectable number of expected Pareto points. This section develops one approach for generating lattices with a predetermined expected number of Pareto points.

The proposed approach uses an analogy to the use of partitioned matrices in linear algebra. The procedure is to first create a set of separate random uncorrelated sublattices such that the sum of their sizes adds up to N, then to form a partitioned lattice of size N from the sublattices, such that no point in one sublattice dominates a point in another. A 2D example is shown in Figure 27. For these lattices, the preferred point is in the bottom left, with Pareto points marked in red.

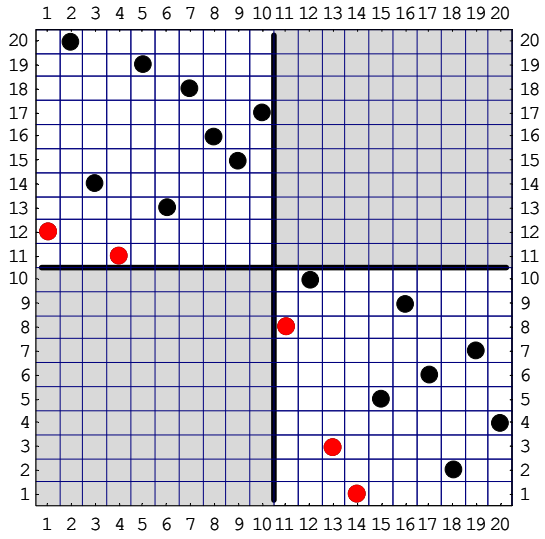

Figure 27: A 2D lattice of size 20 constructed from two lattices of size 10

Note that no point in the lower right block dominates a point in the upper left and vice versa. If we define the notation $E_d(N)$ to indicate the expected number of Pareto points in a random lattice of dimension d and size N, then the expected number of Pareto points for the lattice in Figure 27 with $N = 20$ is $2E_2(N/2)$. Since in general, for arbitrary dimension d and arbitrary nonzero lattice sizes n and m the relationship

$$E_d(n) + E_d(m) > E_d(n+m) \tag{13}$$

strictly holds, this is greater than the value one would get without partitioning, i.e., $2E_2(N/2) > E_2(N)$. Equation (13) suggests that, for a desired value of E_{Target}, one can arbitrarily choose a set of sublattice sizes $n_1, n_2, ..., n_k$ such that they sum to N and $E_{\text{Target}} \simeq \sum E_d(n_i)$.

The number of possible candidate choices for the set of sublattices $\{L_{n_1}, L_{n_2}, ..., L_{n_k}\}$ is the partition function $P(N)$. An approach to determining the set of lattices would be to start at one end of the set of possible partitions and search through the set until the best choice is found. However, searching through the set of all possible partitions would be difficult as the function $P(N)$ grows exponential with N; also, the function E_d is not monotonic in the lexicographic ordering of the set of partitions, which is how algorithms to generate the set typically present them.

Figure 28 shows, for a lattice with $N = 20$, how the list of possible partitions maps to the expected number of Pareto points. The black points correspond to the list being sorted in lexicographic order, which is clearly not monotonic. The red points, forming a smooth line, are for the set of partitions sorted by the expected number of Pareto points they generate. The other point to take from the figure is that there are over 600 candidate partitions for a lattice of size 20.

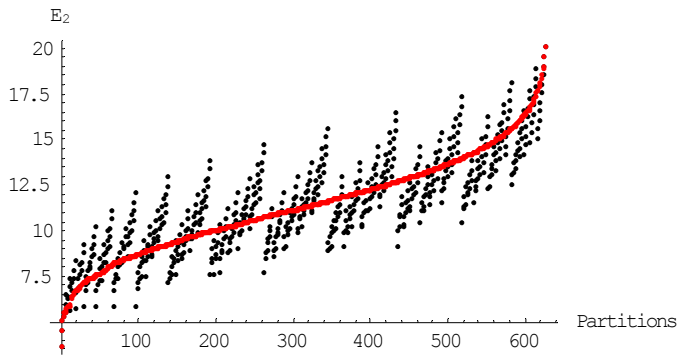

Figure 28: Expected number of Pareto points for a partitioning of a lattice, where the candidate partitions are sorted in lexicographic order (black) and sorted by $E_2(\bullet)$ (red)

In order to deal with the issues of lexicographic ordering and the exponential growth of the number of possible partitions with problem size, consider as candidate partitions only sets of the form $\{m, m, \ldots m, m-1, \ldots m-1, r\}$, where m is some integer less than or equal to N, $m-1$ is the integer one less than m and for which there may be zero or more occurrences, and r is a remainder such that $r < m-1$ and so that the elements of the set sum to N. A couple examples of such sets follow for $N = 15$.

$$\{6,6,3\}$$
$$\{4,4,3,3,1\}$$
$$\{4,3,3,3,2\}$$
$$\{2,2,2,2,2,1,1,1,1,1\}$$

The total number of candidate sets meeting this criteria is

$$\begin{aligned} \sum_{i=1}^{N} \lfloor N/i \rfloor &< N \sum_{i=1}^{N} \frac{1}{i} \\ &= N H_N \\ &\sim N \ln N \end{aligned} \quad . \tag{14}$$

So the number of candidates increases by $N \log N$ as opposed to exponential. Figure 29 plots the list of candidate sets as sorted in lexicographic order versus their expected number of Pareto points, and shows a monotonic relation that smoothly covers the possible values of E_d. For the figure, $N = 30$ and $d = 2$.

Some additional processing can be done to choose the starting point for the algorithm. Numerically solving the relationship

$$E_{\text{Target}} = n \times E_d(N/n) \tag{15}$$

for n, and then using the value of $\lfloor N/n \rfloor$ to create the starting partition set ensures that the starting point will be close to the final partition set chosen.

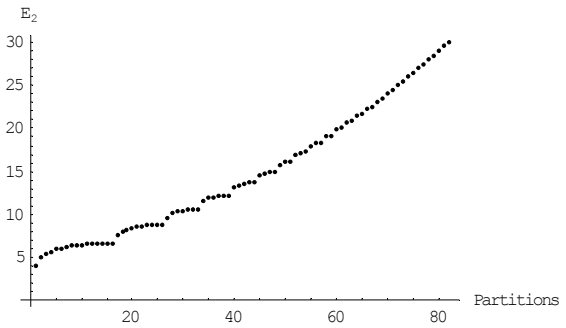

Figure 29: Relation of the restricted subset of candidate partitions to E_d

The algorithm to generate a random correlated lattice (GRCL) for *N* points in *d* dimensions developed for this research is presented next.

Algorithm GRCL Given a size N, a dimension d and an expected number of Pareto points $E \in [E_d(N), N]$, generate a random lattice that has an expected number of Pareto points close to but not exceeding E.

GRCL-1	[Form starting candidate partition] Form a partition set numerically solving Equation (15) for n, determining $q = \lfloor N/n \rfloor$, and forming $I_0 = \{q, q, ..., q, r\}$ where r is a remainder less than q.
GRCL-2	[Generate and test candidate partition sets] Generate and test, in reverse lexicographic order, the candidate partitions I_i until the relation satisfies $E_d(I_i) \le E$. The monotonic relation of the partitions with respect to $E_d(\cdot)$ guarantees that, upon stopping, the set I_i will be the one that most closely meets E without exceeding it. The restriction of E to the range $E \in [E_d(N), N]$ guarantees that the iteration will stop.
GRCL-3	[Determine the size of the chosen set I_i] Count the number of elements in the set I_i in the variable k.
GRCL-4	[Create base lattice] Create a base lattice of size k and dimension d such that all points in the lattice are nondominated. This can be done by (using the 3D case for an example) generating a set of points such that each lies on the plane X+Y+Z=0. The values for X and Y are randomly chosen, and the relation Z=-(X+Y) is used to set Z. Finally, the points are normalized to make a lattice.
GRCL-5	[Create sublattices] For $i \in I_i$, create a lattice of size i and dimension d. The sublattice will be random and uncorrelated, and there will be k of the sublattices.
GRCL-6	[Create final lattice] Replace each of the points in the base lattice with a sublattice chosen without replacement at random from the set of sublattices.

Figure 30 shows, for $N = 20$ and $d = 2$, a series of random correlated lattices with increasing expected number of Pareto points.

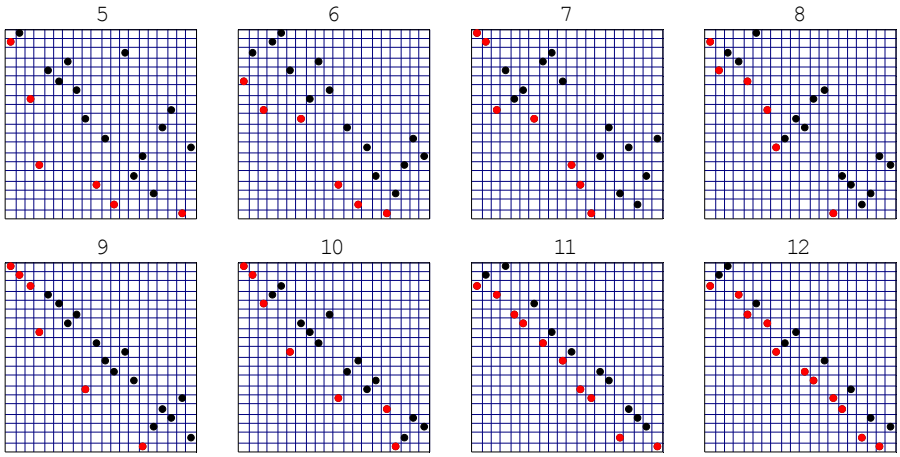

Figure 30: Lattices with increasing number of expected Pareto points (listed above each plot)

4.2 Lattice Hyperplanes Method

The GRCL algorithm has the potential to generate data that can favor a divide and conquer approach due to decoupling between portions of the point space. To counter this property, a different method based on populating points on separated hyperplanes is also developed. While in practice this method is more cumbersome, the development of the method illuminates a number of important properties that define Pareto points. For clarity, this approach is developed for a 3D space and then extended to problems of arbitrary dimension.

Start with two planes, parallel to each other, with their normals equal to $\begin{bmatrix}1 & 0 & 0\end{bmatrix}$, and separated by a distance $q = .5$, as shown in Figure 31.

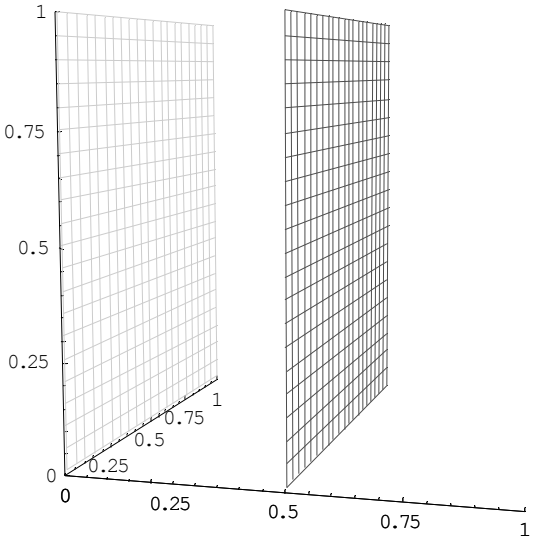

Figure 31: Parallel planes

Randomly place points on both planes, using a uniform distribution in the Y and Z directions as shown in Figure 32.

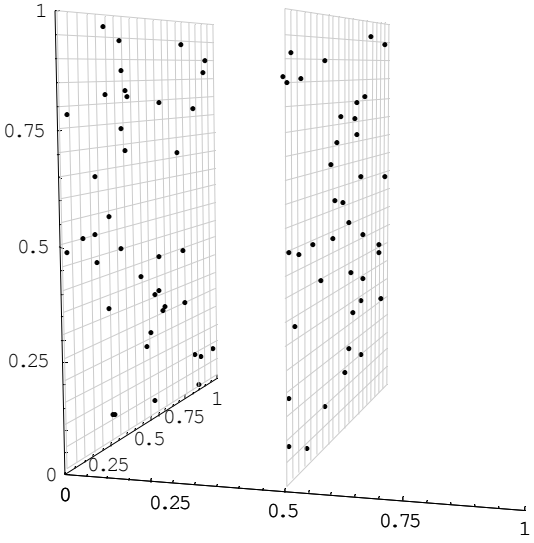

Figure 32: Planes with random points

Rotate both planes about the origin, so that their normal vectors are now $[1 \quad 1 \quad 1]$, as per Figure 33.

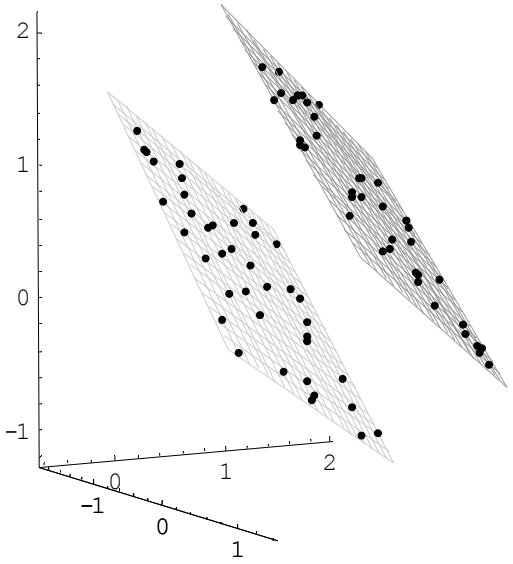

Figure 33: Rotated 2D point clouds

Letting dominance be defined by higher values of X, Y, and Z, then each point on the plane closer to the origin defines a space on the outer plane such that if that space is occupied, then the point is dominated. Figure 34 shows the space, in red, that must be unoccupied for the point on the inner plane to be nondominated.

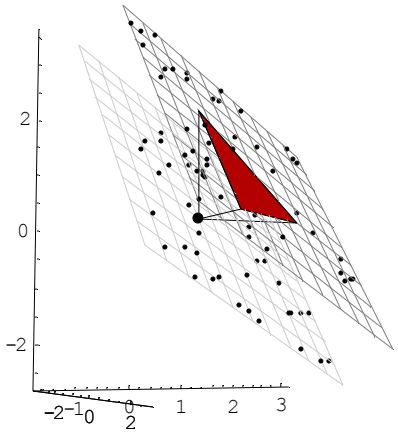

Figure 34: Dominance between planes

If the area marked in red is a and the area of the outer plane is A, and N points are randomly distributed on the outer plane, then the probability that any one point in the outer plane does not lie in the region colored red is $(A-a)/A$. Therefore, the probability that all points do not lie in the red region, and the point on the inner plane is nondominated, is

$$P[\text{point is nondominated}] = \left(\frac{A-a}{A}\right)^N . \tag{16}$$

The area a can be determined by the separation q between the planes through basic geometry. Therefore, by controlling the number of planes, the separation between planes, and the number of points distributed randomly on the planes, one can generated random data with a predetermined expected number of Pareto points.

In general, the approach is, for a d-dimensional problem, to first create a series of (d-1) dimensional hyperplanes perpendicular to the first axis with a fixed separation between them that are randomly populated with points, then rotate all of them about the origin so that they are now perpendicular to the $\begin{bmatrix}1 & \cdots & 1\end{bmatrix}^T$ vector, and then normalize the data into LLH form. The formal description for problems of arbitrary dimension follows.

The geometric figure shown in Figure 34, and reproduced in Figure 35, is a *trirectangular* tetrahedron. The red face is referred to as the *base facet*, and the vertex opposite the base facet is the *peak vertex*. It is trirectangular because the three edges touching the base facet intersect at right angles to each other. For this case, the lengths of the edges that meet at the peak vertex are all the same length.

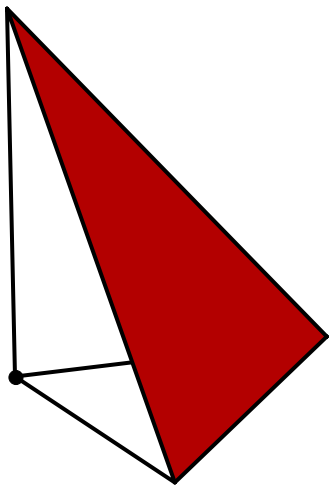

Figure 35:Trirectangular tetrahedron

The d-dimensional generalization of a tetrahedron is a *simplex*, which is for dimension d the simplest geometric form possible. Referring to a simplex in d dimensions as a d-simplex, it will have $d+1$ facets, with each facet being a (d-1)-simplex itself. For example, the tetrahedron above is a simplex in 3D space, it has 3+1=4 facets, and each facet is a triangle, which is just a simplex in 2D space.

A *d*-dimensional simplex will be referred to as a d-rectangular simplex if all edges emanating from the peak vertex are orthogonal to each other, which is the case here. For this work, the lengths of the edges extending from the peak vertex are equal to each other, resulting in a higher dimensional analog to a right triangle referred to as a *d-right simplex*.

The generalized term for the volume (3D) or area (2D) is the *content*. For a *d*-right simplex with orthogonal edges of length l, the content c is

$$\begin{aligned} c &= \int_0^l \int_0^{x_1} \cdots \int_0^{x_{d-1}} dx_d \cdots dx_2 dx_1 \\ &= \frac{l^d}{d!} \end{aligned} \tag{17}$$

The distance q to the base facet from the peak vertex is related to the length of the edges, l, and the dimension d by the derived relation

$$q = \frac{l}{\sqrt{d}}. \tag{18}$$

The generalized Pythagorean theorem says that for a d-rectangular simplex, the square of the content of the base facet c_b is equal to the sum of the squares of the contents of the other facets. Since the other facets all have the same content c_s for this work the following relation holds:

$$\begin{aligned} c_b{}^2 &= \sum_d c_s{}^2 \\ &= d c_S{}^2 \end{aligned}.$$

Since the content of the side facets is, using Eq. (17), equal to

$$\frac{l^{d-1}}{d-1!}.$$

The content of the base facet is

$$\sqrt{d}\,\frac{l^{d-1}}{d-1!}.$$

Since we will be controlling the separation of the hyperplanes rather than edge lengths, Eqn. (18) can be rewritten using Eqn. to give

$$\begin{aligned} c_b &= \left(\frac{d^{d/2}}{d-1!} \right) q^{d-1} \\ &= \alpha_d q^{d-1} \end{aligned},$$

where α_d replaces the expression in the parentheses to simplify the notation. Assuming N points in the outer hyperplane, a total content of the outer hyperplane of 1 (hyperplane bounded by edges of length 1),

and that the base facet of the simplex generated by the candidate point lies wholly within the outer hyperplane, then the probability of a point on the inner hyperplane being dominated is given by

$$\left(1-\alpha_d q^{d-1}\right)^N,$$

and if there are M points on the inner hyperplane, then the total expected number of nondominated points in the data is

$$E[\text{nondominated points}] = M(1-\alpha_d q^{d-1})^N + N\ .$$

The calculation can be readily extended to multiple hyperplanes. Assuming that the separation between hyperplanes is a constant q, then the probability of a point on the *r*th hyperplane is nondominated is the product of the probabilities that it is not dominated by any of the points on the hyperplanes before it. Assuming there are M_i points on the *i*th hyperplane, then the expected number of nondominated points on the *r*th hyperplane is

$$M_r\left(\prod_{i=1}^{r-1}(1-\beta_d(r-i)^{d-1})^{M_i}\right).$$

The rotation from a plane perpendicular to the $\begin{bmatrix}1 & 0 & \ldots & 0\end{bmatrix}$ vector to a plane perpendicular to the $\begin{bmatrix}1 & 1 & \ldots & 1\end{bmatrix}$ vector is accomplished by first creating the matrix

$$R' = \left[\begin{array}{c|ccc} 1 & 0 & \ldots & 0 \\ \hline 1 & 1 & 0 & 0 \\ \vdots & 0 & \ddots & \vdots \\ 1 & 0 & \cdots & 1 \end{array}\right]$$

then applying the Gram-Schmidt orthogonalization (Golub and Van Loan 1996) to R' to get rotation matrix R, and then multiplying each point vector by R to rotate it to the desired plane.

As noted in the introduction to this section, this method proves to be cumbersome, and it is difficult to tune the various parameters such as points on each plane, and separation of the planes, to get a desired expected number of Pareto points. However, the method does suggest a simpler approach whereby the total number of points and the exact number of Pareto points can be specified, and a randomized data set generated to meet the requirements. Such a method is described in the next section.

4.3 Random Dominant Points Method

The approach to this method is to determine a number of random points that are guaranteed to be nondominated, and then create the remainder of the points randomly so that they are guaranteed to be dominated. This method has the advantage of generating a lattice with exactly the number of desired Pareto points, rather than a data set that has the expected number of Pareto points.

Assume the problem is to generate a set of data with N points, of which Q are nondominated, and is of dimension d. Again, start with a $d-1$ dimensional plane perpendicular to the $[1 \quad 0 \quad ... \quad 0]$ vector, and randomly place Q points on the hyperplane. Rotate these points so that the hyperplane is now perpendicular to the $[1 \quad 1 \quad ... \quad 1]$ vector using the method described in the previous section. To generate each of the remaining $N-Q$ points that are dominated, in turn randomly select a nondominated point $\mathbf{z}$, generate a random point $\mathbf{w}'$ such that each of its coordinates is greater than zero, than create the dominated point $\mathbf{w}$ by letting $\mathbf{w} = \mathbf{z} + \mathbf{w}'$.

Since the only constraint on the coordinates of $\mathbf{w}'$ is that they be greater than zero, any number of methods of generating the random coordinates is available. The one used here is to let each of the coordinates be distributed i.i.d. with a half normal distribution (same as the normal distribution except restricted to positive values). Varying the parameter of the distribution, σ, it is possible to change the probability that $\mathbf{w}$ will be dominated by more than point.

Algorithm RDP Given a size N, a dimension d and an expected number of Pareto points $Q \in [1, N]$, generate a random lattice that has Q an expected number of Pareto points.

RDP-1	[Form points on hyperplane] Randomly assign Q points to the hyperplane perpendicular to the $[1 \quad 0 \quad ... \quad 0]$ vector.
RDP-2	[Rotate the hyperplane] Multiply each of the Q points by the rotation matrix R (generated as described above), so the hyperplane has been rotated to be perpendicular to the $[1 \quad 1 \quad ... \quad 1]$ vector.
RDP-3	[Randomly pick one of the Q nondominated points] Pick one of the nondominated points, with equal probability between them. Name the point $\mathbf{z}$.
RDP-4	[Create point $\mathbf{w}'$] Create a point $\mathbf{w}'$ such that each of its coordinates is randomly set i.i.d. via a half-normal distribution with chosen parameter σ. Varying σ varies the "thickness" of the resulting data set.
RDP-5	[Create dominated point $\mathbf{w}$] Let $\mathbf{w} = \mathbf{z} + \mathbf{w}'$.
RDP-6	[Repeat until done] Repeat steps RDP-3 through RDP-5 until a total of N points have been created.

Figure 36 shows how, for a fixed number of Pareto points and total points, the parameter σ affects the distribution of the data. The parameter adjusts the "thickness" of the data, which in turn affects for each

dominated point the number of Pareto points that dominate it. The top row has the raw data, and the bottom row the same data converted to LLH form.

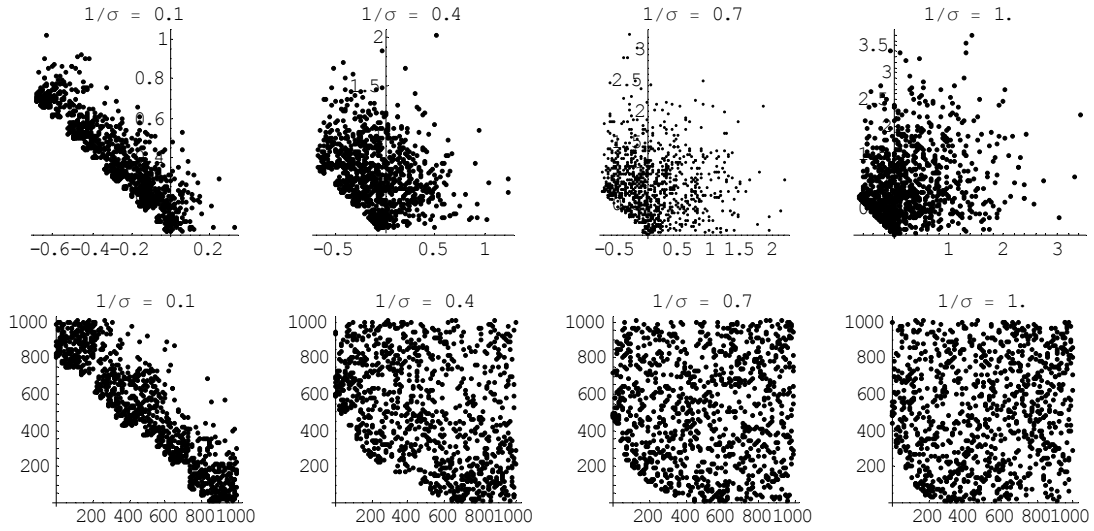

Figure 36: 1000 points total, 25 Pareto points, varying sigma for the half normal distribution

Figure 37 shows 1000 points, 40 Pareto points, and a large range of values for σ.

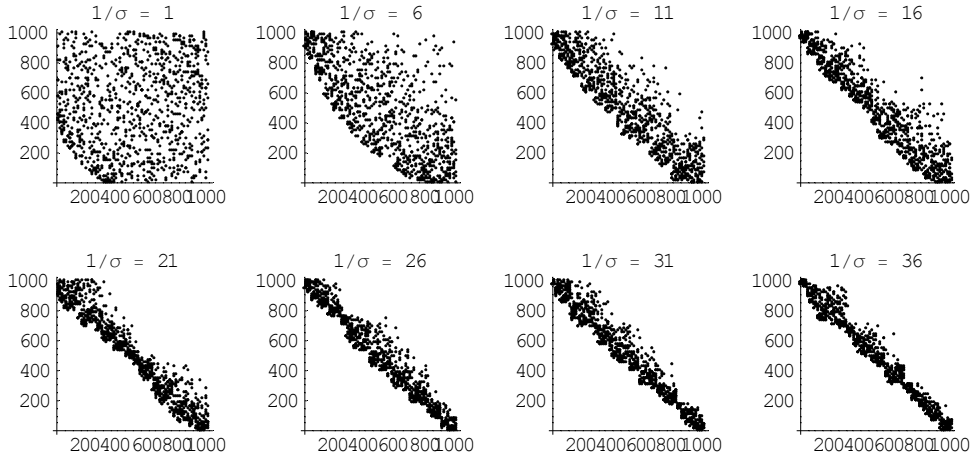

Figure 37: 1000 points, 40 Pareto points, 2D, varying σ

Figure 38 shows a set of data with varying numbers of Pareto points. One feature to notice is the appearance of the line in the data at the Pareto frontier. This artifact does not appear to impact the performance of the algorithms, as later chapters show. The top row is the raw data, while the bottom row is the same data in LLH form. The appearance of the line at the frontier is heavily dependent on the choice of σ.

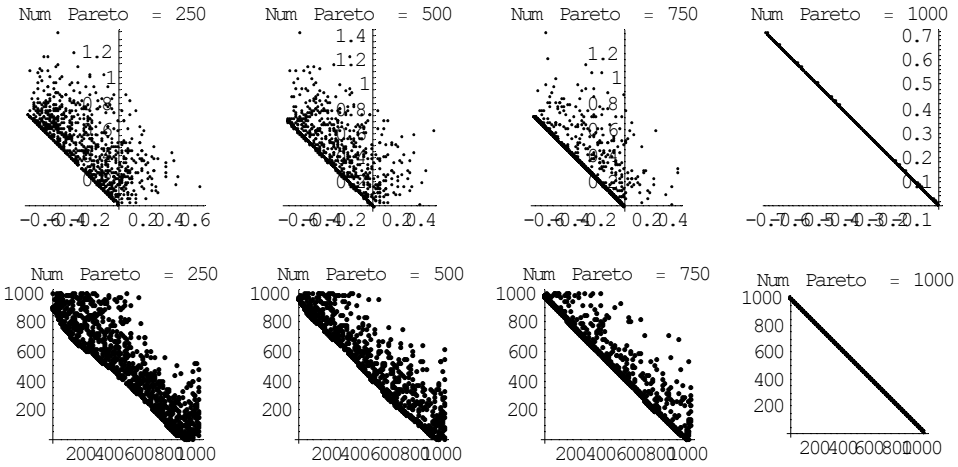

Figure 38: Data with 1000 points, s = 1/10. varying number of Pareto points

The rationale for using the half-normal distribution is that, with respect to direction, it has the property that letting each dimension be distributed half-normal uniformly distributes the points over the positive quadrant of the hypersphere. Other distributions do not have this property, instead they cluster the points near to the normal vector of the plane. An option in distributing the points is to let the distance from the origin for the dominated points be a random variable such as the exponential or any other distribution that generates positive deviates, and the direction of the dominated point from its parent point be randomly chosen to as to be uniformly distributed on the positive quadrant of the unit hypersphere. To generate the points for dimension *d*, let each of x_1 through x_d be i.i.d. half-normal with fixed variance. Then let the direction cosines be defined as

$$\xi_i = \frac{x_i}{\sqrt{\sum_{j=1}^{d} x_j^{\,2}}}.$$

The direction cosines will be uniformly distributed on the positive quadrant of the unit hypersphere.

4.4 Summary

Three different methods to generate random lattices with predetermined expected numbers of Pareto points are presented. Of the three, the RDP algorithm is the fastest in run time, allows the user to exactly specify the number of Pareto points, and provides a parameter to let the user adjust the "thickness" of the data. A drawback to the method is the appearance of a boundary artifact in the data, which does not appear in the GRCL algorithm. The GRCL algorithm avoids this artifact, but is slower to execute and does not guarantee an exact number of Pareto points in the data that it generates. It would be possible to mix and combine the two methods, but this is not explored here.

CHAPTER 5

Introduction to Algorithms to Determine Pareto Set

This chapter serves as an introduction to the basic issues governing a choice of algorithm to determine the Pareto frontier from a multi-dimensional data set. It also introduces the Simple Cull (SC) algorithm, a brute force algorithm that operates in $O(n^2)$ time, and the L2D algorithm, a linear time algorithm that only operates on 2D data. The chapter also introduces the computational framework used for implementing the algorithms, and executing the experiments.

The premise of the development of Pareto algorithms in the next three chapters of the thesis is that no single algorithm can perform best against all data sets, so a hybrid adaptive algorithm is needed to give the best performance. The work of Bentley, in which he develops a recursive divide and conquer algorithm for computing the Pareto frontier (Bentley 1980) and also identifies the property that certain data sets may be more amenable to treatment by asymptotically less computationally efficient algorithms (Bentley et al. 1990) will be the starting point for this effort.

A number of algorithms are presented and explored, with the final algorithm, Hybrid Divide & Conquer (HDC), being a hybrid of a lightweight but inefficient algorithm SC that is introduced in Section 5.2 , a lightweight and efficient algorithm L2D that only operates on 2D data (Section 5.4), and a heavyweight but efficient algorithm Divide & Conquer (DC) that operates recursively on higher dimensional data (Chapter 6).

SC is an algorithm that computes in $O(n^2)$ time, but is not recursive, leading to a low constant multiplier for computing the actual run time $K_{SC1}n^2$, and that has a run time independent of the dimension of the data. L2D is a very fast algorithm that computes on $O(n \log n)$ on 2D data, or if the data is already in LLH form the algorithm is $O(n)$. Also, the algorithm is not recursive, and runs in the same time against all data sets.

DC is a 'Divide & Conquer' algorithm that computes in $O(n \log^{d-1} n)$ if the data requires a comparison sort, or $O(n \log^{d-2} n)$ if the data is already in LLH form. The DC algorithm is recursive leading to a large constant multiplier for computing its run time $\kappa_{DC1} \times n \log^{d-2} n$, that includes dimension as a factor in computing its run time. This implies that, for small problem sizes of high dimension, the SC algorithm will run faster than DC even though it is asymptotically less efficient. A hybrid algorithm that applies a DC

strategy of recursively dividing the problem until the problem reaches some critical size or a dimension of 2, then switches to either the L2D or the SC algorithm, combines the best of both.

Also, one can best attack certain data sets by using one or more passes of a brute force algorithm such as the SC algorithm first, followed by running the remainder of the hybrid algorithm. This is true for data sets with relatively small numbers of Pareto points. As it is possible to compute bounds on the number of Pareto points up front, one can choose to adapt the algorithm to complement the data.

5.1 Computational Framework

The algorithms and experiments developed in this thesis are all implemented in Mathematica, chosen for its ease of code development, ease in instrumenting code, and ease in developing graphics to display the results. However, the use of Mathematica does distort the comparison of the different algorithms in a number of ways as compared to what results would be with a compiled language such a C or FORTRAN.

Mathematica is a mixture of interpreted code and code that is resident to its kernel, with the code in the kernel running orders of magnitude faster. Therefore any algorithm that can take advantage of kernel-resident code has an advantage. One example of kernel-resident code is the algorithm for sorting.

Mathematica also has a heavy overhead for structuring recursive function calls as compared to compiled code, thereby penalizing an algorithm that uses recursion. The DC algorithm developed in Chapter 6 depends on the recursion.

Mathematica does not support ‘pass by reference’ in its function calls, instead only supporting ‘pass by value’. This also penalizes recursive algorithms in both execution time and in memory required. Furthermore, how the DC algorithm is implemented in Mathematica is strongly affected by the need to pass variables by value as opposed to by reference. As the algorithms are compared, it is necessary to keep these points in mind, and understand that comparing the algorithms if implemented in a different language would mean different relative performance between them.

5.2 Simple Cull Algorithm

This section presents the Simple Cull (SC) algorithm, explores issues of estimated run time related to its operating on LLH data, and presents experimental and analytical estimates of best case, worst case, and generalized case run times. While the algorithm is fairly obvious to define and implement, the implications of operating on the LLH form and the analytical and experimental model comparisons are unique to this thesis. The SC algorithm is as follows.

Algorithm SC Given a table **Z** find the nondominated points

SC-1	[Sort **Z** in descending order using a linear weighting scheme] Form a new table **U** where **U** is a permutation of **Z** such that for $i < j, S(\mathbf{u}_i) < S(\mathbf{u}_j)$.
SC-2	[Add top point to collection of Pareto points] Remove the top element of **U**, (refer to it as $\mathbf{u}'$) and add it to the collection of Pareto points **P**.
SC-3	[Cull the remaining points] Compare $\mathbf{u}'$ to each of the remaining elements in **U**, deleting those which are dominated.
SC-4	[Any points left?] If **U** is empty, algorithm is complete and **P** contains Pareto points from **Z**.
SC-5	[Repeat as necessary] Go back to SC-2.

That the algorithm is correct can be seen by the fact that at each execution of Step SC-2, the top element is non-dominated. No other point above it in the list dominates it or else it would have been culled from the list already. It is not dominated by points below it in the list either, since for a top element $\mathbf{u}'$ the relation $S(\mathbf{u}') < S(\mathbf{u})$ implies that if $\mathbf{u}$ is less than $\mathbf{u}'$ in one coordinate, then it must exceed it in at least one other.

SC is analogous to a bubble sort routine (Knuth 1997), and it has a worst case complexity of $\mathrm{O}\left(n^2\right)$ since if each element of **Z** is Pareto, each element must be compared with every other element, for a total number of comparisons of $\left(N^2 - N\right)/2$. The initial sort in step SC-1 requires $\mathrm{O}\left(n \log n\right)$ complexity. If **Z** has only a few Pareto points, then the algorithm runs very efficiently as the first few passes through the data will eliminate most of the points. If only one point is Pareto, the algorithm will complete with one pass. If all points are nondominated, the comparisons will occur in the order of $(n-1)+(n-2)+\cdots+3+2+1$, i.e., the most comparisons occur early in the algorithm.

The choice of a linear weighting ranking function to use in sorting the list may seen arbitrary, as in fact there are many candidate ranking functions from which to choose. In general, any weighting function of the form $\sum w_i u_i^{\,p}$ would work, as would functions such as $\max_{i \in I}\left[\max\left[X_{1i}, X_{2i}, ..., X_{di}\right]\right]$ and $\max_{i \in I}\left[\min\left[X_{1i}, X_{2i}, ..., X_{di}\right]\right]$. Some of these are plotted in Figure 39 for the 2D case.

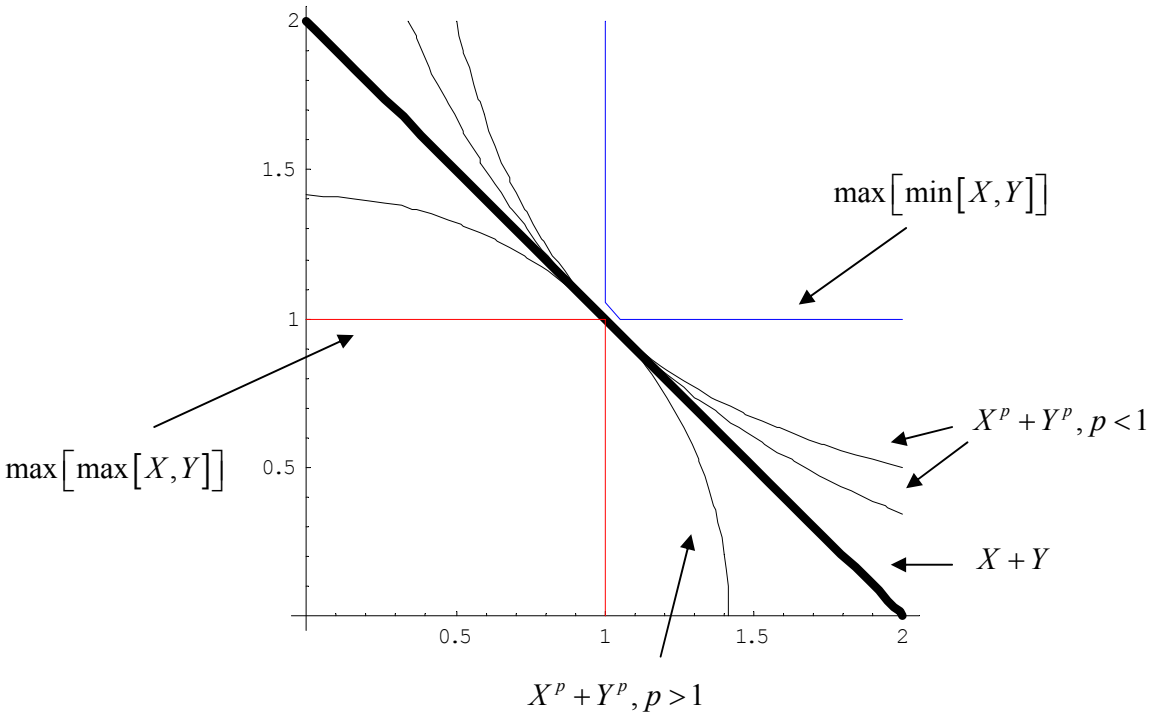

Figure 39: Different ranking functions

However, using the ranking function that is just the linear sum of the dimensions $S(\mathbf{u})$ guarantees that the most possible dominated points will be removed on the first pass through the algorithm. This can be seen by recalling the results of Section 3.5 and noting that the minimum number of points dominated by any point $\mathbf{z}$ is $S(\mathbf{z}) - (d-1)N - 1$. So maximizing the minimum number of points removed on the first pass implies using a linear weighting scheme with all of the weights set to 1.

5.2.1 Analytical Estimate of the Expected Run Time for SC for Arbitrary Dimension

This section computes the expected run time for the SC algorithm given a problem of size N with k Pareto points. With a predetermined number of Pareto points, it is possible to directly compute upper and lower bounds on the number of comparisons required. The lower bound is reached when all dominated points are removed during the first pass, in which case the run time is

$$\frac{k^2 - k}{2} + N - k = \frac{k^2 - 3k + 2N}{2}. \tag{19}$$

This result is derived by first noting that there will always be $(k^2 - k)/2$ comparisons between the k nondominated points. Since the remaining $N - k$ dominated points are removed on the first pass, using $N - k$ comparisons to do so, the total number of comparisons comes to the result in Eq. (19).

The upper bound on run time occurs when the dominated points are not culled until the last pass of the algorithm, in which case the number of comparisons is

$$\frac{k^2-k}{2}+k(N-k)=\frac{2Nk-k^2-k}{2}. \tag{20}$$

This result is derived by noting that all k nondominated points must be compared with the $N-k$ dominated points, with the dominated points not being removed until the last set of comparisons. Plotting the two curves in Figure 40 shows that the difference between them is an order of magnitude over much of the possible values of k.

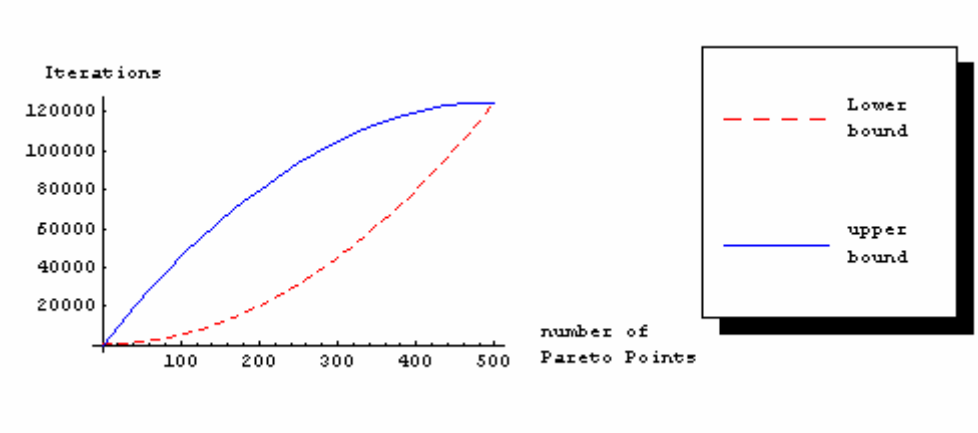

Figure 40: Upper and lower bounds on comparisons, N=500

The estimating model developed here is based on three assumptions as follows:

1. Each Pareto point dominates a fraction of the dominated points, with the fraction $p \in [0,1]$
2. Which points are dominated by a Pareto point are random
3. Every Pareto point has a unique dominated point that only the specific Pareto point dominates

The first two assumptions imply that a dominated point can be dominated by more than one Pareto point, while the third assumption ensures that there will be dominated points remaining in the data set until the last pass of the algorithm. Based on these assumptions, a fraction of the remaining dominated points are removed at each pass through the algorithm. Letting n_i be the number of dominated points remaining after the ith pass of the algorithm, the expected number remaining after the next pass is $n_{i+1} = pn_i$. Starting from n_0, the number remaining after i steps is $p^i n_0$. The number of comparisons total for a run of the algorithm is

$$\frac{k^2-k}{2}+\sum_{i=1}^{k} p^{i-1}n_0 .$$

This expression, replacing n_0 with $N-k$, resolves to

$$\frac{k^2-k}{2}+\sum_{i=1}^{k} p^{i-1} n_0 = \left\{ \begin{array}{ll} \frac{k^2-k}{2}+(N-k)\left(\frac{1-p^k}{1-p}\right); & p \in [0,1) \\ \frac{k^2-k}{2}+k(N-k); & p=1 \end{array} \right\}. \tag{21}$$

This expression is a generalization of the upper and lower bounds, with the lower bound reached when $p=0$ while the upper bound results when $p=1$.

This type of model where the number of points dominated is reduced through a geometric progression is supported by observation, which shows that a reasonable choice for p is one where, at the last pass through the algorithm, the last Pareto point has only one point to remove. This leads to the third assumption. Since at the last pass, the algorithm will have made $k-1$ passes through the data, the expression for p can be determined as

$$p=\left(\frac{1}{N-k}\right)^{1/k-1}.$$

Inserting this value for p back into Eq. (21) gives the expression

$$\frac{k^2-k}{2}+(N-k)\left(\frac{1-\left(\frac{1}{N-k}\right)^{k/k-1}}{1-\left(\frac{1}{N-k}\right)^{1/k-1}}\right) \tag{22}$$

Comparing the upper and lower bounds, the relation captured in Eq. (22), and experimental results at multiple data dimensions, shows Eq. (22) is a reasonable fit, but that there is significant variance. The experimental data shown in Figure 41 is for a fixed lattice size of $N=100$, varying numbers of Pareto points, and for varying dimensions of the problem. Each plotted point is the result of one run of the SC algorithm, with the data generated using the GRCL algorithm.

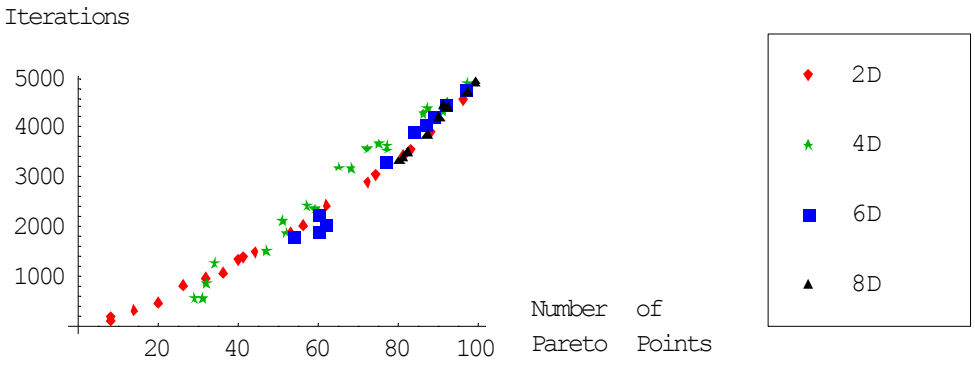

Figure 41: Experimental iterations

The same data shown is in Figure 42 with upper and lower bounds and the analytical estimate of the number of comparisons, shown as the dashed line.

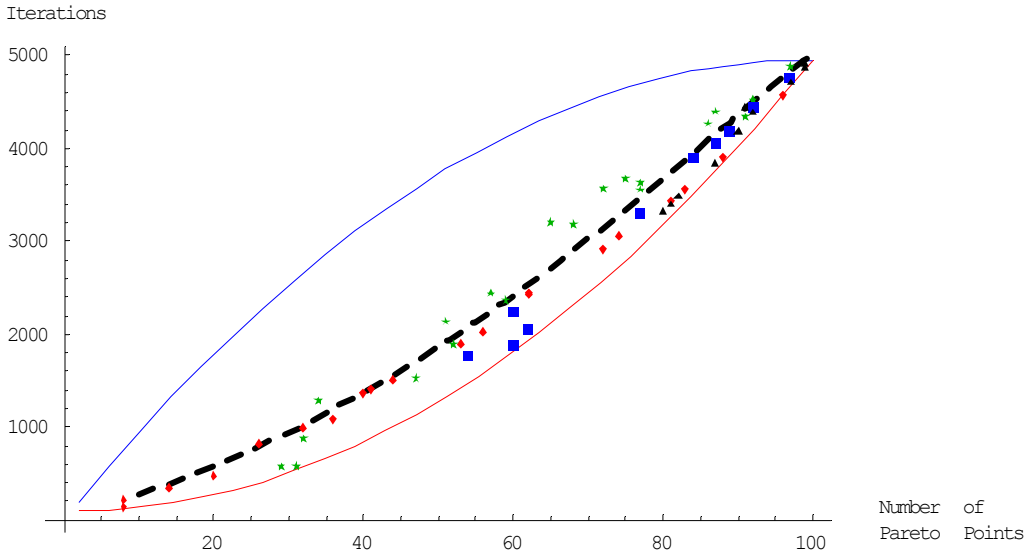

Figure 42: Experimental data, upper bound, lower bound, and analytical estimate

One immediate conclusion that can be drawn from the data is that—as expected—the number of comparisons in the algorithm is only a function of the number of points in the problem and of the number of Pareto points. The dimension of the problem does not figure into the run time complexity and only loosely affects the actual run time of the implemented algorithm.

5.2.2 Best Case Experiment

The best case performance for the SC algorithm is when there is a single Pareto point. An example of such a configuration of points is shown Figure 43 where the bottom left of the figure is the best solution. For this configuration, the algorithm will complete with one pass through the data, so for N points the algorithm will execute in $N-1$ steps.

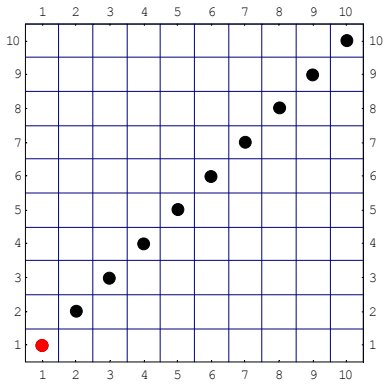

Figure 43: Best case configuration of points

One would expect the iterations to vary linearly with problem size, and this is what is observed in experiments as seen in Figure 44. In these experiments, data with a single Pareto point was generated using the RDP algorithm for each lattice size from 2000 to 10,000 points.

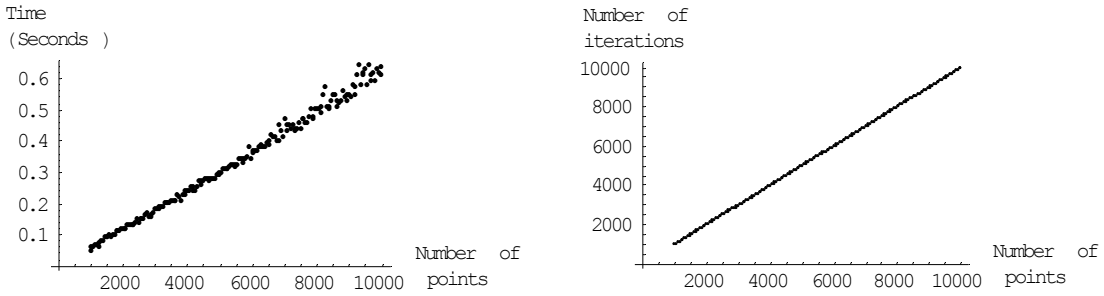

Figure 44: Run time for best case, SC

5.2.3 2D Worst Case Experiment

The worst case performance occurs when all points are Pareto, as in the 2D case of Figure 45.

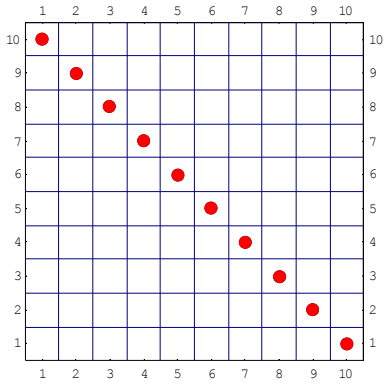

Figure 45: Worst case configuration of points

In this situation, the algorithm computes in N^2 iterations, as shown in Figure 46. Again, the RDP algorithm was used to generate lattices, this time ranging in size from 200 to 1000 points. The smaller lattice sizes were needed to make run times practical, due to the quadratic growth in run time with respect to size.

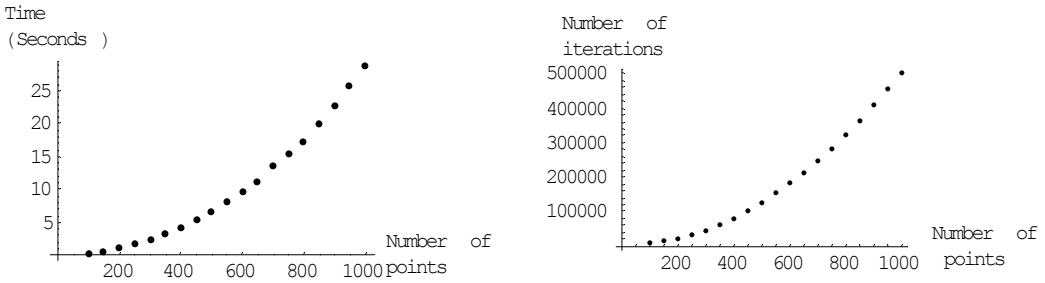

Figure 46: Worst case run time, SC

5.2.4 Random Experiment

Using the expression in Eq. (22) as a model, one can plot how the number of iterations grows for varying ratios of Pareto points to problem size (see Figure 42). Figure 47 shows the results of experiments where both the ratio of Pareto points to problem size and the problem size itself are varied. The different curves represent different ratios of problem size to number of Pareto points. In the experiment, the RDP algorithm was used to generate lattices with a known ratio of dominated points to total lattice size, with one lattice generated for each size ranging from 1 to 200.

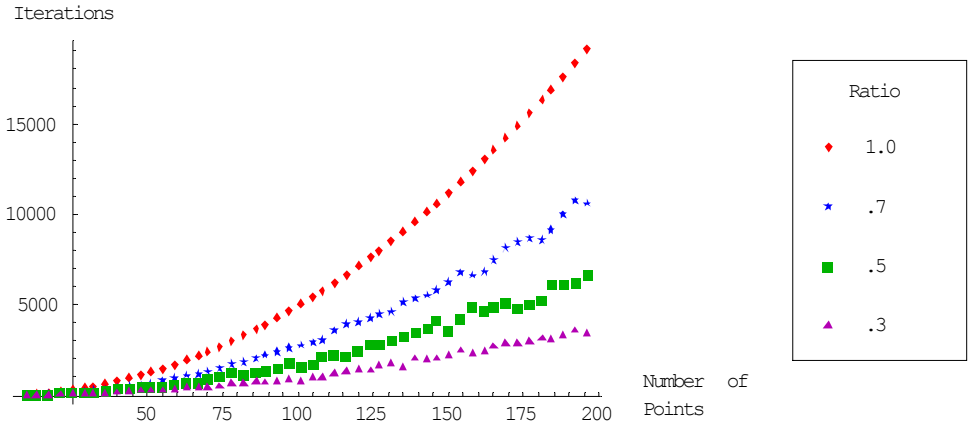

Figure 47: Run time for varying problem size and varying ratio of Pareto points to problem size

The figure indicates that, as expected, the ratio of run times between two data sets of the same size, but with differing percentages of Pareto points, is independent of the size of the data sets.

5.2.5 Simple Cull2

The second algorithm, Simple Cull 2 (SC2) is a minor variation of the first, included for completeness. Rather than compare the top ranking point in the list against those remaining on the list to cull the remaining list, the top point is compared to those points that are already identified as nondominated, to determine if the top point will be added to the Pareto list.

Algorithm SC2 Given a set **Z** find the nondominated points that are in the set

SC2-1	[Form list of scalar rankings] Form a new list **U** where **U** is a permutation of **Z** such that for $i < j, S(\mathbf{u}_i) < S(\mathbf{u}_j)$.
SC2-2	[Pick off first point of list] Remove the top element of **U**, add to list of Pareto points **P**.
SC2-3	[Pick off top point] Remove the top element of **U**, (refer to it as $\mathbf{u}'$).
SC2-4	[See if it is dominated by higher points] Compare $\mathbf{u}'$ to each of the elements in **P**. If nondominated, add to **P**.
SC2-5	[Any points left?] If **U** is empty, algorithm is complete and **P** contains points from **Z** that are Pareto.
SC2-6	[Repeat as necessary] Go back to SC2-4.

As per SC, the top point in the list **U** is nondominated. At each step, the point $\mathbf{u}'$ is not dominated by any points below it, does not dominate any points previously considered, and is only added to **P** if it is not dominated by any points having been previously considered.

SC2 again has a worst case complexity of $\mathrm{O}\left(n^2\right)$ since if each element of **Z** is Pareto, the collection of points **P** to be compared with grows linearly, and the number of comparisons is $1+2+\cdots+(N+1)=\left(N^2-N\right)/2$.

5.3 Diagonalizing Algorithm for Finding Pareto Points in 2D Case

Results in Section 3.4 showing that all points Pareto implies all points on the diagonal suggested an algorithm for finding the Pareto points in a two-dimensional problem. Stated simply, first place the points into LLH form. Then remove all points below the diagonal. Renormalize the remaining points into LLH form and repeat until no points are below the diagonal. This is the DIAG algorithm that follows.

Algorithm DIAG Given a set **Z** of points in a 2D space, find the nondominated points

DIAG-1	[Place into Latin Hypercube form] Form a new list **U** where **U** = T(**Z**) (not necessary if **Z** is already in form).
DIAG-2	[Remove elements below the diagonal] Remove the elements of **U** where $S(\mathbf{u}) < N+1$.
DIAG-3	[Repeat as necessary] If points were removed in DC-2, return to DC-1 and create a new list in Latin Hypercube form from the remaining elements of **U**.

As proven in Section 3.4, any dominated points will lie below the diagonal. When no points are below the diagonal, then all points are nondominated. The complexity of the algorithm, determined experimentally,

shows that the expected number of passes through the DC algorithm varies in proportion to $\log n$, implying the expected run time of the algorithm is $O(n \log n)$.

5.4 Nonrecursive Algorithm on 2D Data

This section introduces the linear 2D (L2D) algorithm as developed in (Kung et al. 1975). The algorithm operates on 2D data non-recursively in $O(n \log n)$ complexity, with the portion of the algorithm identifying the Pareto points operating in $O(n)$. If the data is already in lattice Latin hypercube form, then the algorithm can be done in $O(n)$.

Algorithm L2D Given a set $\mathbf{Z}$ of points in a 2D space, find the nondominated points

L2D-1	[Order the points by their first column] Form a new list $\mathbf{U}$ that is a permutation of $\mathbf{Z}$ such that it is sorted by the first column.
L2D-2	[Pick off first point of list] Remove the top element of $\mathbf{U}$, add to list of Pareto points $\mathbf{P}$.
L2D-3	[Set the value of $y_{\max}$] Set $y_{\max} = u_{1,2}$.
L2D-4	[Scan through the points] Compare $y_{\max}$ to the second element of each $\mathbf{u}$ in turn. If $u_2 \le y_{\max}$, add $\mathbf{u}$ to the Pareto points $\mathbf{P}$ and reset $y_{\max} = u_2$.

That this algorithm is correct can be seen by noting first that the top point in the sorted list is Pareto, as no point has a lower u_1 value. As all other points have greater u_1 values, a new point can be Pareto if and only if it has a lower u_2 value. This property holds at any Pareto point $\mathbf{u}_i$ in the list for all points above it.

The algorithm completes in one pass through the data, with the number of comparisons independent of the number of Pareto points. Therefore, for 2D data this algorithm clearly is computationally superior. It is $O(n \log n)$ only due to the need to sort the data first, being linear otherwise. If the data is in normalized form, then it can be sorted in linear time, and the algorithm becomes $O(n)$, and in fact takes exactly N-1 comparisons.

CHAPTER 6

Divide & Conquer Algorithm

This chapter presents and implements, and analyzes a divide & conquer (DC) algorithm, as developed by Bentley (Bentley 1980). He shows that for a problem of size N and dimension d that there is an algorithm of complexity $O(N \log^{d-1} N)$ that will identify the Pareto points. Since he assumed the need to do a comparison sort in the algorithm, and these sorts can be accomplished in linear time with the data in LLH form, the algorithm can be considered to run in $O(n \log^{d-2} n)$.

The presentation of the algorithm is first done geometrically, as from Bentley's paper. In order to better develop an analytical estimate of the run time of the algorithm, it is then developed in a form based on repeated operations on columns of a table. This second presentation is easier to interpret for high dimensional problems, and leads to both an infinite dimension estimate and a finite dimension estimate of the number of comparisons. The analytical estimates assume that all points in the data are Pareto. These analytical estimates are validated against experimental data.

As the DC algorithm is recursive, analytical estimates of the number of function calls in the recursion and the amount of data passed to each function call are developed. These are needed later in determining the switch points between the DC algorithm and the SC algorithm.

Unlike the SC algorithm, which has a deterministic number of comparisions when executed against a data set with all points Pareto, the DC algorithm's run time can vary by many orders of magnitude between data sets with all Pareto points. The sensitivity of the DC algorithm to the structure of the data is demonstrated thorugh a simple example.

To complete the analysis, a simple model for estimating the number of comparisons for the DC algorithm as a function of the percentage of points that are Pareto is developed. Again, while the SC algorithm is deterministic in its number of comparisons for a data set with a single point Pareto, the DC algorithm is not. The range of possible values for the number of comparisons for a data set with only one Pareto point spans orders of magnitude.

The basic approach in computing the Pareto points in a problem of size N and dimension d is to first solve two problems of size $N/2$ and dimension d and then solve a problem of size N and dimension $d-1$. The

process of dimension reduction continues recursively until $d = 2$, at which time the nonrecursive L2D algorithm L2D can be applied. As an example, consider Figure 48, with 160 points and 3 dimensions. The preference is for minimum values of coordinates, i.e., the lower left rear corner.

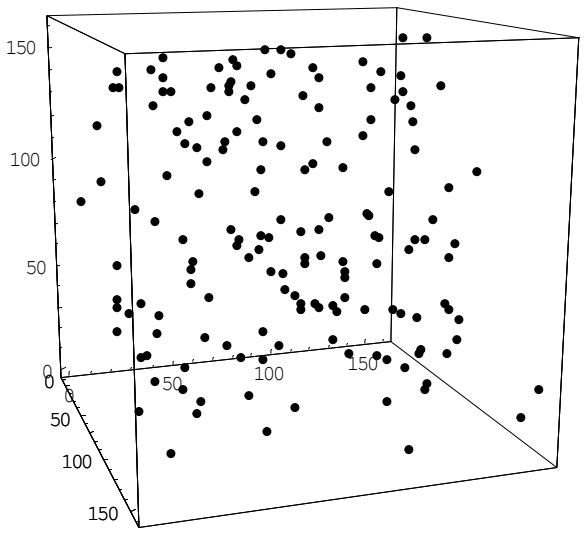

Figure 48: 160 points in 3 dimensions

Figure 49 shows both the results of problem size reduction and the reduction of dimension. The problem was first reduced in size by a cut plane into two sets of equal size, A and B so that no point in B dominates a point in A. This is accomplished by sorting the data along the appropriate dimension and taking all points into B that are above the median value. For Figure 49, A is the set of points on the left of the cutting plane and B is the set of points on the right. Each problem's Pareto points are calculated, with A's Pareto points in red, and B's in blue. As points that are Pareto in region B may not be Pareto in the total problem, they must be removed in a *marriage* step.

.

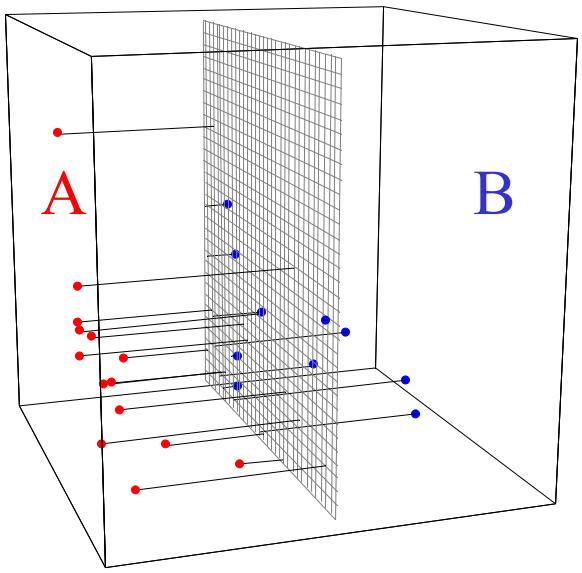

Figure 49: Subdivided problem

The Pareto points from each problem are now projected onto the cut plane between the regions, see Figure 50, and the Pareto points are identified on this cut plane. Blue points that are not dominated in the 2D problem (indicated by gray shading behind them) are not dominated in the 3D problem and are added to the Pareto points in set B, and the DC algorithm is finished.

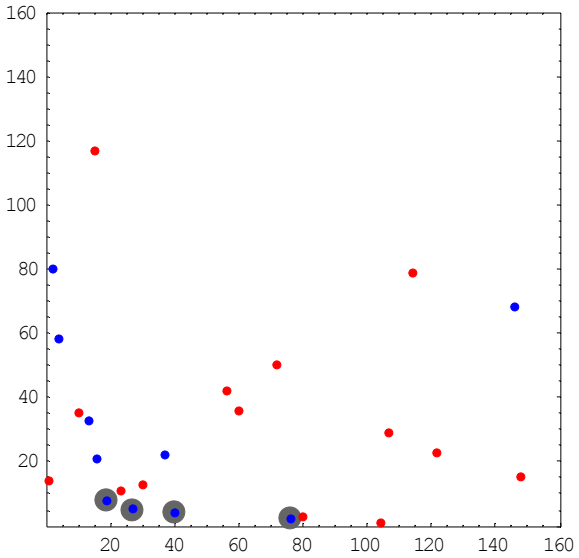

Figure 50: Projection onto cut plane

As stated above, the complexity of the DC algorithm is $O(n \log^{d-2} n)$. That this is true can be seen by first recognizing that for a 2D problem the complexity is $O(n)$, since the L2D algorithm is used. Now, assuming that N is a power of two, i.e., for some integer $N = 2^k$ for some integer k or conversely $k = \log N$, then the complexity of the 3D case is

$$C(N,3) = \sum_{k=1}^{\log N} N \,,$$
$$= N \log N$$

while the complexity for the 4D case is

$$C(N,4) = \sum_{k=1}^{\log N} 2^k \left[\frac{N}{2^k} \log\left(\frac{N}{2^k} \right) \right]$$
$$\leq N \sum_{k=1}^{\log N} \log N$$
$$= N \log^2 N$$

and similarly for higher dimensions.

The algorithm is actually a cooperating set of six algorithms, with DC being the main entry point, and L2D, SC, MARRY, MARRY2D, and MARRYDIRECT used at different points. L2D and SC have already been introduced. MARRY is a recursive algorithm that repeatedly reduces the problem size and reduces the dimension. MARRY2D is a modification of L2D for when the two data sets to be married are two dimensional. MARRYDIRECT is analogous to SC, in that it uses a brute force enumeration approach when the problem size is small. Each is presented in turn.

Algorithm DC Given a table **Z** of points in a d -dimensional space, find the nondominated points that are in the set.

DC-1	[If dimension of data is 2, call L2D on data and return results]
DC-2	[If size of problem is below limit, call SC on data and return results]
DC-3	[Else divide into two problems of equal size] Split **Z** into two sets of points **X** and **Y** by finding the median element of the first column, and taking points whose first element is greater the median and placing them in the inferior set **X** , and those points lower in superior set **Y** .
DC-4	[Call DC on subproblems] determine **XX** = DNC(**X**) and **YY** = DNC(**Y**) .
DC-5	[Reduce dimension] drop the first columns of **XX** and **YY** to form **XXX** and **YYY** .
DC-6	[Call MARRY to cull remaining dominated points in the inferior set] Determine **XXX'** = MARRY(**XXX**, **YYY**) .
DC-7	[return the results] Return **YYY** $\cup$ **XXX'** .

The procedure for marrying the points together is the crux of the algorithm. At the completion of calling DC on the two sets of data each N/2 in size, two sets of points $\mathbf{X}_P$ and $\mathbf{Y}_P$ of dimension d will remain with the property that all the points are Pareto in their respective sets and that no point in $\mathbf{X}_P$ dominates a point

in $\mathbf{Y}_P$. The marry algorithm must now compare the two sets and remove points in $\mathbf{X}_P$ that are dominated by points in $\mathbf{Y}_P$.

This step itself is a recursive process of problem reduction followed by dimension reduction and is illustrated in Figure 51. With the two sets, $\mathbf{Y}_P$ and $\mathbf{X}_P$, first divide them on a cut plane (dotted line) into $\mathbf{X}_1$ and $\mathbf{X}_2$ and $\mathbf{Y}_1$ and $\mathbf{Y}_2$ respectively. Call $\mathbf{X}'_2 = \text{MARRY}(\mathbf{X}_2, \mathbf{Y}_2)$ and $\mathbf{X}'_1 = \text{MARRY}(\mathbf{X}_1, \mathbf{Y}_1)$. In the dimension reduction step, drop the first columns of $\mathbf{Y}_1$ and $\mathbf{X}'_2$ (thereby taking the projection of the points onto the remaining $d-1$ dimensions), and determine $\mathbf{X}''_2 = \text{MARRY}(\mathbf{X}'_2, \mathbf{Y}_1)$. Note that $\mathbf{X}_1$ is not compared to $\mathbf{Y}_2$, as it is not possible for a point in $\mathbf{Y}_2$ to dominate a point in $\mathbf{X}_1$. If either $\mathbf{Y}_1$ or $\mathbf{X}''_2$ are empty, then the dimension reduction process terminates.

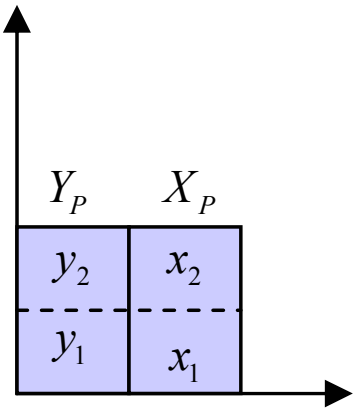

Figure 51: Divide and conquer in Marry algorithm

The MARRY algorithm must again track the dimension, and switch to a nonrecursive efficient algorithm (MARRY2D) if the dimension reaches two, or to a brute force algorithm (MARRYDIRECT) if the problem size reaches a critical point. For now, the critical size will be two, but this size will be adjusted based on dimension in the hybrid algorithm. The marrying algorithms follow.

Algorithm MARRY Given two sets of points $\mathbf{X}$ and $\mathbf{Y}$ in a d-dimensional space, with the properties that no point in $\mathbf{X}$ dominates another point in $\mathbf{X}$ or a point in $\mathbf{Y}$, and no point in $\mathbf{Y}$ dominates another point in $\mathbf{Y}$, determine the set of points $\mathbf{X}' \subseteq \mathbf{X}$ such that no point in $\mathbf{X}'$ is dominated by a point in $\mathbf{Y}$.

MARRY-1	[If data dimension is 2D, call MARRY2D and return results]
MARRY-2	[If size of problem is below limit, call MARRYDIRECT on data and return results]
MARRY-3	[Else divide into two problems of equal size, and recursively call MARRY] Choose a cut plane to divide $\mathbf{X}$ and $\mathbf{Y}$ into $\mathbf{X}_1, \mathbf{X}_2, \mathbf{Y}_1, \mathbf{Y}_2$ such that $\mathbf{X}_2$ is inferior to $\mathbf{X}_1$ and $\mathbf{Y}_2$ is inferior to $\mathbf{Y}_1$ as based on their first column, and so that $\lvert\mathbf{X}_1\rvert + \lvert\mathbf{Y}_1\rvert = \lvert\mathbf{X}_2\rvert + \lvert\mathbf{Y}_2\rvert$. Call $\mathbf{X}_1' = \text{MARRY}(\mathbf{X}_1, \mathbf{Y}_1)$ and $\mathbf{X}_2' = \text{MARRY}(\mathbf{X}_2, \mathbf{Y}_2)$.
MARRY-4	[Now drop a dimension, and recursively call MARRY] Drop the first columns of $\mathbf{X}_2'$ and $\mathbf{Y}_1$ to form $\mathbf{XX}_2'$ and $\mathbf{YY}_1$ and call $\mathbf{XX}_2'' = \text{MARRY}(\mathbf{XX}_2', \mathbf{YY}_1)$.
MARRY-5	[Form union of results and return] Return $\mathbf{X}_1' \cup \mathbf{X}_2''$.

Algorithm Marry2D Given two sets of points $\mathbf{X}$ and $\mathbf{Y}$ with properties as per the MARRY algorithm, but in a 2-dimensional space, determine the nondominated points in $\mathbf{X}$.

MARRY2D-1	[Mark the inferior and superior points, and form their union] Mark the elements of $\mathbf{X}$ and $\mathbf{Y}$ to distinguish between them, and form $\mathbf{W} = \mathbf{X} \cup \mathbf{Y}$.
MARRY2D-2	[Order the points by their first column] Sort $\mathbf{W}$ by its first column.
MARRY2D -3	[Augment sorted list with minimum element] Prepend to $\mathbf{W}$ the record $\{0, \lvert\mathbf{W}\rvert + 1\}$, marked as an element of $\mathbf{Y}$.
MARRY2D -4	[Set the value of q] Set $q = \lvert\mathbf{W}\rvert$.
MARRY2D -5	[Scan through the points, updating q only if point is in superior set] Compare q to the second element of each $\mathbf{w}$ in turn. If the second element of $\mathbf{w}$ is $\le q$, and if $\mathbf{w}$ is also an element of $\mathbf{X}$, add $\mathbf{w}$ to the inferior nondominated points $\mathbf{X}'$. If the second element of $\mathbf{w}$ is $\le q$ and $\mathbf{w}$ is also an element of $\mathbf{Y}$, reset q so that q equals the second element of $\mathbf{w}$.
MARRY2D-6	[Return the inferior points that survived] Return $\mathbf{X}'$.

Algorithm MarryDirect (MD) Given two sets of points **X** and **Y** in a *d*-dimensional space as per the MARRY algorithm, identify the points in **X**.

MD-1	[Directly compare each inferior point to each superior point] For reach point in **X**, directly compare it to all of the points in **Y**,and if it is not dominated by any of them, add it to the set **X'**.
MD-2	[Return nondominated inferior points] Return **X'**.

As each point in **X** must be compared with each point in **Y**, the complexity of this algorithm, given superior set **Y** of size N and inferior set **X** of size M is $O(NM)$.

6.1 Alternate Development of the DC Algorithm

The following sections develop a worst case analysis of the algorithm, which occurs when all points are nondominated. To develop a closed-form expression for the worst case run time, first this section develops a different approach to understanding the algorithm.

Again, start with a set of N points of dimension d. Recall that in the first step of the DC algorithm, the points are sorted based on their first dimension, split into two halves based on that dimension, and the nondominated points in each half are identified. As an example, consider a data set of 18 points shown in Figure 52. The data is split on the first dimension, and points 1-9 and 10-18 are treated as separate sets.

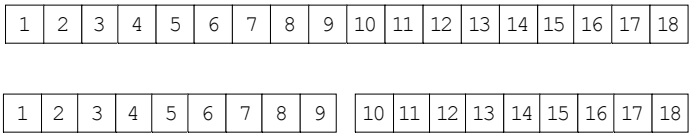

Figure 52: Data sorted on first dimension, and split

Once the nondominated points in each subset are identified, the two sets must be married via the MARRY algorithm. Let the points in the set of points 1-9 that are nondominated be collectively labeled as A, and the points from the set 10-18 be collectively labeled as B. Each of the A points are nondominated within their set, and they also cannot be dominated by a point in the set of B points, since the value for of the first dimension for every point in B is greater than the value of the first dimension for every point in A. Each of the B points are nondominated within their set, but they can be dominated by points in A. The object of the MARRY algorithm is to remove the B points that are dominated by points in A.

Figure 53: After culling the two subsets

In the data above, assuming that for each set of 9 points only 8 of them are nondominated within their set, the total number of points remaining is now 16; each set having lost one point. In the next step, the points

are resorted, this time on the second dimension, and again broken into two sets (see Figure 54). Each set, on the left and the right, contains A and B points. Label the left set as L and the right set as R.

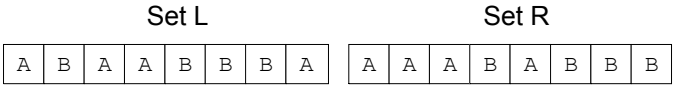

Figure 54: Points resorted on the second dimension

The algorithm then compares the B points in set L with the A points in set L and any B points that are dominated are removed, similarly for R. In this case, assume that all points remaining are nondominated, and therefore no points are removed from either side.

At this point, what is known is that all of the B points in set L are nondominated by any A points, left or right. Also, no A points in set R dominate any B points, left or right. What is not known is whether any A points on the left dominate any of the B points on the right. Mark the left B points and the right A points in gray (see Figure 55). In the next step, since left B points and right A points cannot dominate or be dominated, they do not be to be considered in the algorithm and can be dropped prior to the next step.

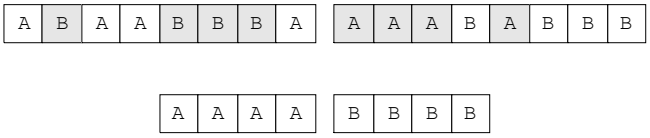

Figure 55: Left B's and right A's marked, and then removed

Again resort the data, now on the third dimension, and repeat. Keep doing so until no points remain.

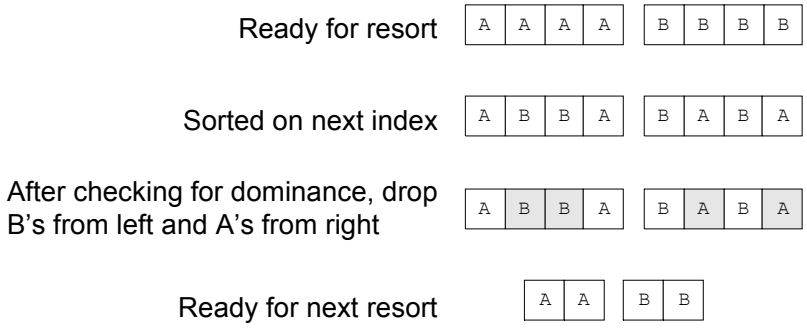

Figure 56: Cycle of resort, check for dominance left and right, drop left Bs and right As

There are three possible situations one ends up with in the resort. The first case, shown in Figure 52 - Figure 56, is where the A's and B's are distributed on both sides of the split. The second case is where the left side has all B's and the right side has all A's after the resort. This indicates that all points are nondominated with respect to each other, and the MARRY algorithm can return immediately. The final

case is where, after the resort, the left side contains all A's and the right side all B's. In this situation, the algorithm will immediately resort on the next chosen dimension and continue.

The critical point to note is that as long as the nondominance of any two points with respect to each other can be established in a finite number of steps of the algorithm, the algorithm will complete *even for a set of points of infinite dimension*. The result stems from the fact that the left B points and right A points are discarded each time the algorithm steps to resort on the next dimension. Given that this is true, it is possible to determine the worst case run time for a set of points of infinite dimension with all points nondominated. From there, the finite dimensional problem can then be described and solved.

6.2 Worst Case - Infinite Dimension Run Time

Assuming the problem is of infinite dimension but that the nondominance of any two points with respect to each other can be established in a finite number of steps of the algorithm, worst case run times of the algorithm can be determined. Three variations of the infinite dimension worst case will be considered here: a *balanced case*, a *maximum unbalanced case*, and a *generalized unbalanced case*. The balance refers to the distribution of A and B points in the L and R sets when each step in dimension is made. The balanced case is when, at each step in dimension, A and B points are equally distributed in sets L and R (see Figure 57).

Figure 57: Balanced case, same number of A and B points in the left and right sets

The maximum unbalanced case is where exactly one B point is in the left set and exactly one A point is in the right set after each resort (see Figure 58).

Figure 58: Maximum unbalanced case, with one B point in left set, and one A point in right set

The generalized unbalanced case is where the ratio of A points on the left (B points on the right) to total points on the left (right) varies by some ratio α. Consider the case in Figure 59 where α is equal to 3/8.

Figure 59: Generalized unbalanced case with ratio = 3/8

Note that the generalized case where $\alpha = 1/2$ is equivalent to the balanced case. Consider first the balanced case in the analysis.

6.2.1 Balanced Case

Although the algorithm developed here is a divide & conquer algorithm, conceptually it is easier to consider it as an initial divide followed by repeated marrying of fewer but larger sets. Define the function $\text{mbi}[n]$ (mbi is short for Marry Balanced Infinite) as a function that determines the number of comparisons required to marry n points together (i.e., marry two sets of $n/2$ points). Define a function $\text{pbi}[n]$ (pbi is short for Pareto Balanced Infinite) to determine the number of comparisons required to verify that a set of n points are nondominated. Then there are two recursive relationships, one for each of the two functions. For the marrying of balanced infinite dimension points the recursion is

$$\text{mbi}[n] = 3\text{mbi}\left[\frac{n}{2}\right].$$

This recursion derives from the fact that the Marry algorithm initially splits the problem of size n into two subproblems of size $n/2$, and then resorts on the next index and calls the Marry algorithm again on the remainder. Since the problem is balanced, half of the points are dropped prior to the final call, so three calls are made on data sets of size $n/2$.

When determining the Pareto points for a balanced case of infinite dimension, the recursion is

$$\text{pbi}[n] = 2\text{pbi}\left[\frac{n}{2}\right] + \text{mbi}[n]. \tag{23}$$

For the DC algorithm, two recursive calls to the DC algorithm with data sets of size $n/2$ are followed by a call to the Marry algorithm with a data set of size n, leading to Eq. (23).

The base case for inductive purposes is the case $n = 2$, where $\text{mbi}[2] = 1$ and $\text{pbi}[2] = 1$. If one assumes that n is a power of 2, i.e., there exists an i such that $n = 2^i$, then the recursions can be resolved directly. For the marry portion, the recursion resolves as follows:

$$\begin{aligned}
\text{mbi}[n] &= 3\text{mbi}\left[\frac{n}{2}\right] \\
&= 9\text{mbi}\left[\frac{n}{4}\right] \\
&\vdots \\
&= 3^{\log n - 1}\,\text{mbi}[2] \\
&= \frac{1}{3}n^{\log 3} \\
\text{mbi}[n] &\simeq \frac{1}{3}n^{1.58}.
\end{aligned}$$

To compute the total comparisons in determining the nondominated points, use the following relation, which is illustrated in Figure 60. Assume again that n is a power of 2.

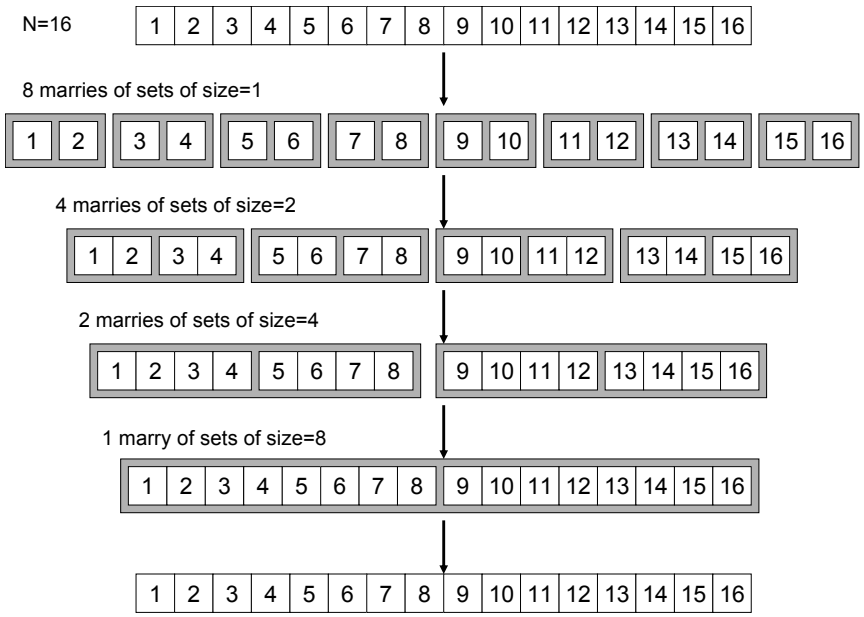

Figure 60: Process of initial divide, followed by repeated marrying of larger sets

$$\begin{aligned}\text{pbi}[n] &= 2\text{pbi}\left[\frac{n}{2}\right] + \text{mbi}[n] \\ &= 4\text{pbi}\left[\frac{n}{4}\right] + 2\text{mbi}\left[\frac{n}{2}\right] + \text{mbi}[n] \\ &\vdots \\ &= n \times \text{pbi}[1] + \frac{n}{2}\text{mbi}[2] + \frac{n}{4}\text{mbi}[4] + \cdots + 2\text{mbi}\left[\frac{n}{2}\right] + \text{mbi}[n]\end{aligned} \tag{24}$$

Recognizing that $\text{pbi}[1] = 0$ and $\text{mbi}[2] = 1$, the Eq. (24) can be recast as a summation,

$$\text{pbi}[n] = n\sum_{i=1}^{\log n} \frac{\text{mbi}[2^i]}{2^i}.$$

Replacing $\text{mbi}[2^i]$ gives

$$\text{pbi}[n] = \frac{n}{3}\sum_{i=1}^{\log_2 n} \frac{\left(2^i\right)^{\log 3}}{2^i}.$$

Resolving the summation in closed-form gives the final result,

$$\begin{aligned}\text{pbi}[n] &= n^{\log 3} - n \\ &= n^{1.58} - n.\end{aligned} \tag{25}$$

Note that although the assumption was made that n was a power of 2, the expression for $\text{pbi}[n]$, calculating the total number of comparisons for the balanced case of infinite dimension, does not depend on this fact, and experimental results show that the expression works for other values of n.

6.2.2 Generalized Unbalanced Case

Again for the generalized unbalanced case of infinite dimension define a function to determine the number of comparisons to identify the nondominated points, $\text{pgui}[n,\alpha]$ (Pareto Generalized Unbalanced Infinite), and a function for marrying sets together, $\text{mgui}[n,\alpha]$ (Marry Generalized Unbalanced Infinite). The α term in the functions represent the ratio of A to total points in the left set at each marry step. The recursion for the function $\text{pgui}[n,\alpha]$ remains the same as for the balanced case,

$$\text{pgui}[n,\alpha] = 2\text{pgui}\left[\frac{n}{2},\alpha\right] + \text{mgui}[n,\alpha]. \tag{26}$$

The recursion for the marry algorithm is, however, more complex. First define a function that takes as input the total number of points in a set (left or right) and the number of B points in that set, and determines the maximum number of comparisons to determine all B points are nondominated. Label the function $\text{unbi}[n,b]$. Assume the recursion

$$\text{unbi}[n,b] = 3\text{unbi}[\frac{n}{2},\frac{b}{2}]$$

holds, and that the base case is $\text{unbi}[n,1] = n-1$. Then the closed-form expression can be calculated as

$$\text{unbi}[n,b] = n\left(\frac{3}{2}\right)^{\log b} - 3^{\log b}. \tag{27}$$

To convert to a form that takes n and α as input, recognize that $b = (1-\alpha)n$ by definition and substitute in Eqn (27) to get

$$\text{unbi}[n,\alpha] = \left(\frac{\alpha(1-\alpha)^{\log 3}}{1-\alpha}\right) n^{\log 3}. \tag{28}$$

Interestingly, the shape of the function is as shown in Figure 61.

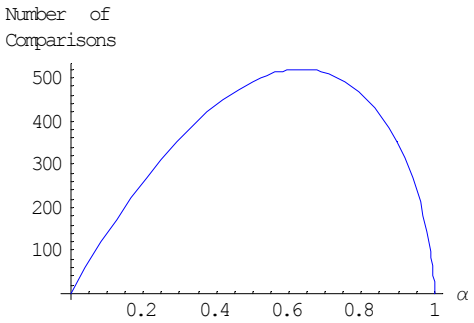

Figure 61: Shape of the unbalanced function, problem size of 100 points, varying alpha

Now the recursion for the Marry algorithm can be expressed as thus,

$$\text{mgui}[n,\alpha] = 2\text{unbi}\left[\frac{n}{2},\alpha\right] + \text{mgui}[\alpha n,\alpha]. \tag{29}$$

Note the αn term in the last part of the expression. This is due to the fact that after each marry step in a dimension, only $(1-\alpha)n$ points are removed, and αn points remain. In this case assume that n is equal to the expression

$$n = 4(1/\alpha)^k \tag{30}$$

for some integer value k, for reasons that will become apparent. Express the recursion in expanded form (suppressing the dependence on α for clarity) to see the relationship,

$$\begin{aligned}
\text{mgui}[n] &= \text{mgui}[\alpha n] + 2\text{unbi}\left[\frac{n}{2}\right] \\
&= \text{mgui}[\alpha^2 n] + 2\text{unbi}\left[\frac{\alpha n}{2}\right] + 2\text{unbi}\left[\frac{n}{2}\right] \\
&= \text{mgui}[\alpha^3 n] + 2\text{unbi}\left[\frac{\alpha^2 n}{2}\right] + 2\text{unbi}\left[\frac{\alpha n}{2}\right] + 2\text{unbi}\left[\frac{n}{2}\right] \\
&\vdots \\
&= \text{mgui}[4\alpha] + 2\text{unbi}\left[\frac{\alpha^{\log_{1/\alpha}(n/4)} n}{2}\right] + \cdots + 2\text{unbi}\left[\frac{\alpha^2 n}{2}\right] + 2\text{unbi}\left[\frac{\alpha n}{2}\right] + 2\text{unbi}\left[\frac{n}{2}\right]
\end{aligned} \tag{31}$$

The reason for $\log_{1/\alpha}(n/4)$ term in the exponent along with the requirement imposed by Eqn. (30) is to ensure that the input to the unbi[·] function in the final recursion is 2, as the function is undefined for smaller values. Also, the mgui[4α] term must be replaced with a closed-form, as repeating the recursion would violate the constraint on the input to the unbi[ ], and so the mgui[4α] function is replaced with the balanced version mbi[4α]. Finally, the recursion is placed into summation form to give

$$\text{mgui}[n] = mbi[4\alpha] + 2\sum_{i=0}^{\log_{1/\alpha}\frac{n}{4}} unbi\left[\frac{\alpha^i n}{2}\right]. \tag{32}$$

As with the balanced equation version of the marry algorithm, this can be resolved to an unwieldy closed-form the following solution:

$$\text{mgui}[n] = \frac{1}{3}(4\alpha)^{\log 3} + \left(\frac{2}{3}\right)\frac{\alpha(1-\alpha)^{\log 3 - 1}}{\alpha^{\log 3} - 1}\left(n^{\log 3} - (4\alpha)^{\log 3}\right). \tag{33}$$

This can be plugged back into the expression pgui[ ] from Eq. (26) and solved for in an analogous fashion to give the complete expression

$$\text{pgui}[n] = A\left(n^{\log 3} - n\right) + B(n-1) \tag{34}$$

where

$$A = \frac{2\alpha(1-\alpha)^{\log 3}}{(1-\alpha)(1-\alpha^{\log 3})}, \tag{35}$$

and

$$B = \frac{3(\alpha^{\log 3} - \alpha^{2\log 3} - \alpha^{3\log 3} + \alpha^{6\log 3}) - 6\alpha^{2\log 3}(1-\alpha)^{\log 3}}{(1-\alpha)(1-\alpha^{\log 3})}. \tag{36}$$

A plot of the number of iterations with varying α is shown in Figure 62, with the dashed horizontal line showing the value for the balanced case. Note that the two lines intersect at the value of $\alpha = .5$, as expected.

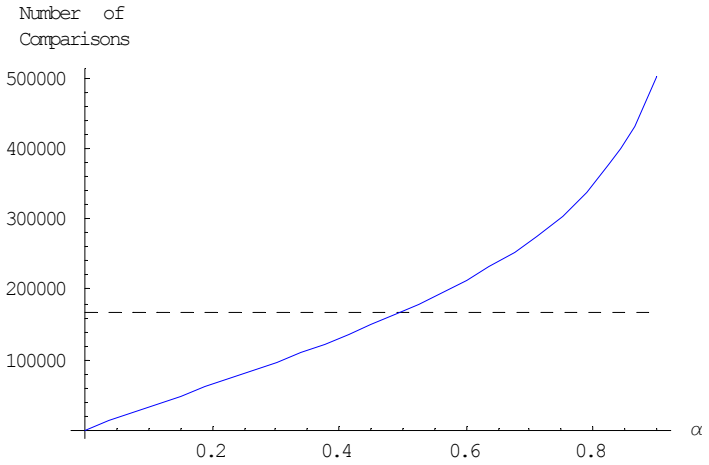

Figure 62: Plot of number of comparisons versus alpha, for 2000 points, infinite dimension, all nondominated. Dashed line is the balanced case, shown for reference.

Note that the number of comparisons increases monotonically with the percentage of imbalance, as expected.

6.2.3 Maximum Unbalanced Case

Define a Marry algorithm and a Pareto algorithm for the maximum unbalanced case of infinite dimension, $\text{mmui}[n]$ (Marry Maximum Unbalanced Infinite) and $\text{pmui}[n]$ (Pareto Maximum Unbalanced Infinite) respectively. Since this case results in only a single B point in the left set and a single A point in the right set at each iteration, the number of comparisons for the marry algorithm can be written as

$$\begin{aligned} \text{mmui}[n] &= 2\left(\frac{n}{2} - 1\right) + \text{mmui}[n-2] \\ &= n - 2 + \text{mmui}[n-2]. \end{aligned} \tag{37}$$

This recursion can be resolved to

$$\begin{aligned} \text{mmui}[n] &= 1 + 2\sum_{i=1}^{n/2-1} i \\ &= \frac{1}{4}\left(n^2 - 2n + 4\right). \end{aligned} \tag{38}$$

The recursive relationship for the Pareto algorithm is

$$\text{pmui}[n] = 2\text{pmui}\left[\frac{n}{2}\right] + \text{mmui}[n]. \tag{39}$$

This can be solved similar to the balanced case to give

$$\begin{aligned} \text{pmui}[n] &= n\sum_{i=1}^{\log n}\frac{\text{mmui}[2^i]}{2^i} \\ &= \frac{1}{2}(n^2+n-2-n\log n) \end{aligned} \tag{40}$$

.

It is interesting to note that while the balanced case and the generalized balanced case both vary in the number of comparisons by $O(n^{\log 3})$, the maximum unbalanced case varies by $O(n^2)$. The maximum unbalanced case is not a limit for the generalized balanced case as $\alpha \to 1$. Figure 63 is a plot of the three different cases.

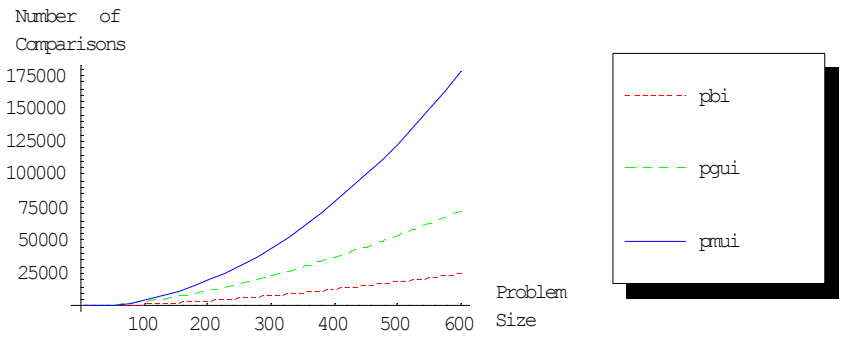

Figure 63: Plot of the balanced case, the generalized unbalanced case with alpha = .9, and the maximum unbalanced case for varying problem size

6.3 Worst Case - Finite Dimension Run Time

The finite dimension computations are simpler to pose than the infinite dimension case developed in the previous section, but the resulting expressions are more complex. First the balanced case is presented, then the generalized unbalanced case, and finally the maximum unbalanced case.

6.3.1 Balanced Case

Define two functions for the Marry algorithm and the Pareto algorithm, $\text{mbf}[n,d]$ (Marry Balanced Finite) and $\text{pbf}[n,d]$ (Pareto Balanced Finite). The recursion for the marry algorithm is now

$$\text{mbf}[n,d] = 2\text{mbf}\left[\frac{n}{2},d\right] + \text{mbf}\left[\frac{n}{2},d-1\right]. \tag{41}$$

There are two base cases to consider, $\text{mbf}[2,d]=1$ and $\text{mbf}[n,2]=n-1$. The second case derives from the fact that for a 2D problem a linear time algorithm is available. One can recursively define in a manner analogous to the infinite case the relationship

$$\text{mbf}[n,d] = \frac{n}{2} + n\sum_{i=2}^{\log n}\left(\frac{\text{mbf}\left[2^{i-1},d-1\right]}{2^i}\right). \tag{42}$$

This can be solved for recursively in d resulting in Table 7.

Table 7: Value of mbf[ ] function for varying dimension

d	mbf[n,d]
2	$n-1$
3	$\frac{1}{2}(n\log n - n + 2)$
4	$\frac{1}{8}(n\log^2 n - 3n\log n + 10n - 8)$
5	$\frac{1}{48}(n\log^3 n - 6n\log^2 n + 35n\log n - 30n + 48)$
6	$\frac{1}{384}(n\log^4 n - 10n\log^3 n + 83n\log^2 n - 194n\log n + 504n - 384)$

The Pareto function can be expressed as two base cases and a recursive relationship,

$$\begin{aligned}&\text{pbf}[2,d] = 1,\\ &\text{pbf}[n,2] = n-1,\\ &\text{pbf}[n,d] = 2\text{pbf}\left[\frac{n}{2},d\right] + \text{mbf}[n,d-1]\end{aligned}$$

Using the base cases, the recursion can be expressed as a summation,

$$\text{pbf}[n,d] = \frac{n}{2} + \sum_{i=2}^{\log n}\frac{\text{mbf}[2^i,d-1]}{2^i}.$$

Again, this can be solved recursively, generating an example table of solutions, Table 8.

Table 8: Value of pbf[ ] for various dimensions

d	pbf[n,d]
2	$n-1$
3	$n\log n - n + 1$
4	$\frac{1}{4}(n\log^2 n - n\log n + 4n - 4)$
5	$\frac{1}{24}(n\log^3 n - 3n\log^2 n + 26n\log n - 24n + 24)$
6	$\frac{1}{192}(n\log^4 n - 6n\log^3 n + 59n\log^2 n - 54n\log n + 192n - 192)$

One should note that, as expected, the number of comparisons varies with $O(n \log^{d-2} n)$. What is interesting is a plot of the number of comparisons required versus dimension of the problem for different fixed numbers of points, as shown in Figure 64.

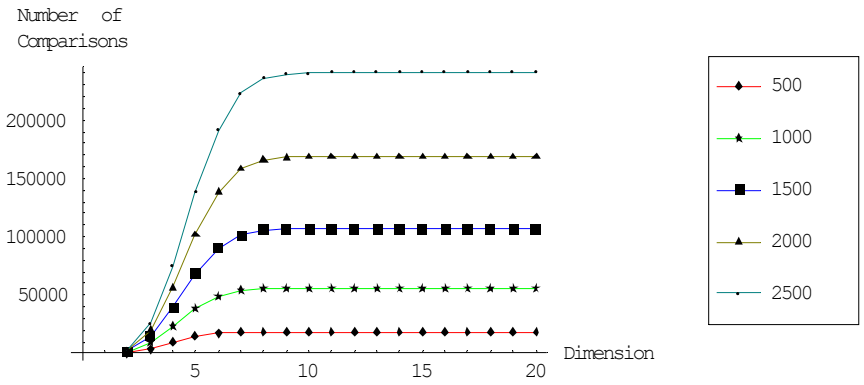

Figure 64: Number of comparisons as a function of dimension for fixed number of points in problem

Note that as the dimension increases, the number of comparisons plateaus. The plateau corresponds to the limiting case of infinite dimension, where the algorithm repeatedly divides the data until only two points are remaining, compares them, and steps to the next dimension in the data set. For the case of high dimensionality or few points, the algorithm completes prior to ever stepping through all of the dimensions. Figure 65 shows just the curve for a problem with 10,000 points and varying dimension, along with a flat line showing the value for infinite dimension.

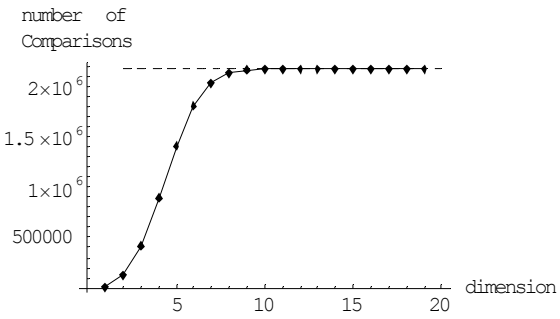

Figure 65: Plot of number of comparisons versus dimension for 10,000 point problem

The dimension at which the two lines join for a problem of size n is approximately equal to $d = \log n$, for a problem of size n, since the mbf$[n, d]$ algorithm steps down through $\log[n]$ dimensions before exhausting all of the points, and in order to not reach the linear case of $d = 2$, must start with a sufficient initial dimension. Figure 66 shows a contour curve for this expression.

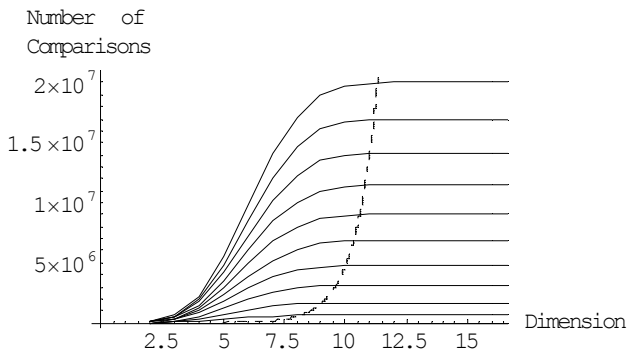

Figure 66: Plot of comparisons versus dimension for varying problem size, with dashed line indicating the point where limiting number of comparisons reached

6.3.2 Generalized Unbalanced Case

Define two functions for the Marry algorithm and the Pareto algorithm for the generalize unbalanced case, finite dimension, $\text{mguf}[n,\alpha,d]$ (Marry Generalized Unbalanced Finite) and $\text{pguf}[n,\alpha,d]$ (Pareto Generalized Unbalanced Finite). Also define a function for computing the number of comparisons in the left and right sets during the marry computation, $\text{unbf}[b,\alpha,d]$ (Unbalanced Finite), where b is the number of B points in the left set, which will be equal to $(1-\alpha)n$. Start the analysis with the $\text{unbf}[n,\alpha,d]$ function. Suppress the dependency on α in the formulas to simplify the presentation.

The $\text{unbf}[b,d]$ function has the following relationships:

$$\begin{aligned}
\text{unbf}[1,d] &= \frac{1}{\alpha} - 1 \\
\text{unbf}[b,2] &= \frac{b}{\alpha} - 1 \\
\text{unbf}[b,d] &= \left(\frac{b}{\alpha} - b\right) + b\sum_{i=1}^{\log b} \frac{\text{unbf}[2^{i-1}, d-1]}{2^i}
\end{aligned} \tag{43}$$

As for the finite dimension balanced problem, this can be solved recursively in dimension to get the desired functional forms. Some examples are given in Table 9, rewriting $\text{unbf}[b,d]$ as $\text{unbf}[n,d]$ from here by replacing b with αn in the original version.

Table 9: Value of unbf[ ] function for various dimensions

d	unbf$[n,d]$
2	$n-1$
3	$\frac{1}{2}(n\log n + n(\log\alpha - 4\alpha + 2) + 2)$
4	$\frac{1}{8}(n\log^2 n + n\log n(2\log\alpha - 8\alpha + 3) + n(\log^2\alpha - 8\alpha\log\alpha + 3\log\alpha + 8) - 8)$

Analogous to the infinite dimensional case, define the base case and recursive relation for $\text{mguf}[n,\alpha,d]$ as

$$\begin{aligned} &\text{mguf}[n,2] = n-1, \\ &\text{mguf}[2,d] = 1, \\ &\text{mguf}[n,d] = 2\,\text{unbf}\left[\frac{n}{2},d\right] + \text{mguf}[\alpha n, d-1]. \end{aligned} \tag{44}$$

With a closed-form expression for $\text{unbf}[n,d]$, the $\text{mguf}[n,d]$ function can be recursively defined on d and values determined. Two examples are given in Table 10 for dimensions 2 and 3, as the expressions become unwieldy with higher dimensions.

Table 10: Values for mguf[ ] at 2 and 3 dimensions

d	mguf$[n,d]$
2	$n-1$
3	$\frac{1}{2}n\log\left(\frac{n}{2}(1-\alpha)\right) - n(1-\alpha) + 1$

A plot of the function versus α for varying dimensions in Figure 67 shows that the function reaches a peak between 0.5 and 1, rapidly declining as it approaches 1. This is due to the effect of finite dimension, where the algorithm terminates early due to reaching dimension size of 2 and completing linear from there. The number of dimensions the algorithm will use if unrestricted is proportional to $\log_{1/\alpha} n$. As $\alpha \to 1$, the term $\log_{1/\alpha} n \to \infty$.

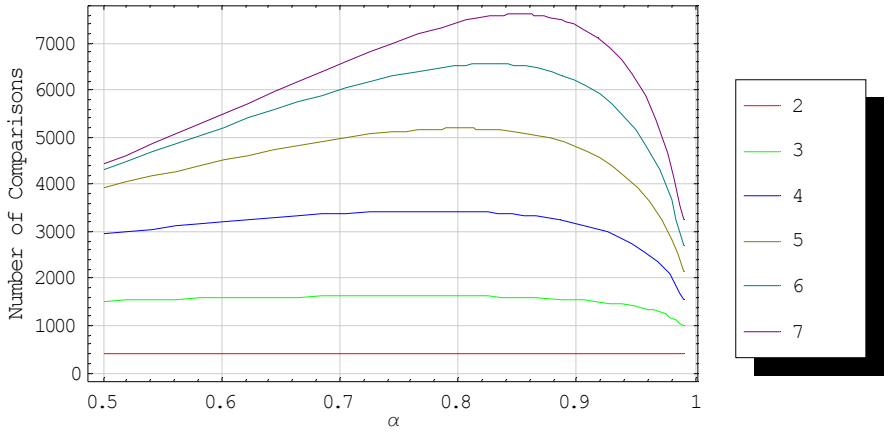

Figure 67: Plot of the mguf[n,d] function for varying dimension and alpha

Finally, the number of comparisons to compute the full function pguf$[n,d]$ can be determined in a manner directly analogous to the balanced finite dimension case, resulting in

$$\text{pguf}[n,d] = \frac{n}{2} + n\sum_{i=2}^{\log n} \frac{\text{mguf}[2^i, d-1]}{2^i} \tag{45}$$

This can be used to recursively determine the functional form with increasing d. Some examples are shown in Table 11.

Table 11: Values of pguf[ ] for dimensions 2 through 4

d	pguf$[n,d]$
2	$n-1$
3	$n\log n - n + 1$
4	$\frac{1}{4}(n\log^2 n + n\log n(2\log(1-\alpha)+12\alpha-5) + n(8-2\log(1-\alpha)-12\alpha)-4)$

Plotting this function with varying α and dimension reveals a similar behavior, where again the number of computations rises to a peak, then rapidly declines with increasing α (see Figure 68).

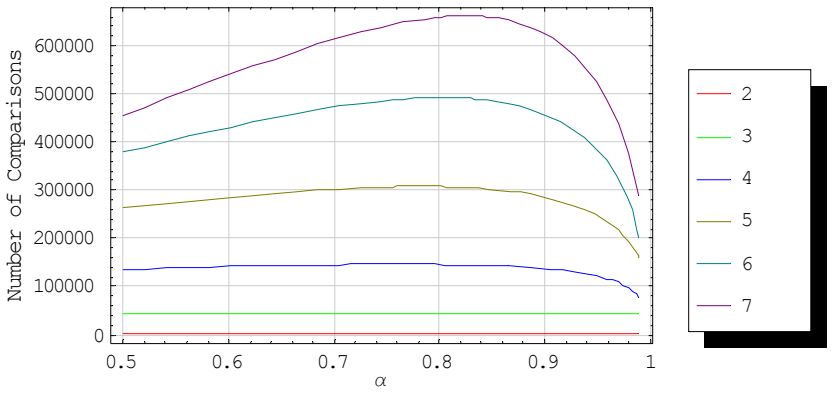

Figure 68: Number of comparisons versus alpha and dimension for a problem size of 4000 points

6.3.3 Maximum Unbalanced Case

Based on the behavior of the generalized balanced case with finite dimension, one would expect the maximum unbalanced case to terminate rapidly as the dimensions are exhausted. This proves to be true. Define functions for the marry and the Pareto algorithms, $\text{mmuf}[n,d]$ (Marry Maximum Unbalanced Finite) and $\text{pmuf}[n,d]$ (Pareto Maximum Unbalanced Finite) as

$$\begin{aligned} \text{mmuf}[n,2] &= n-1, \\ \text{mmuf}[n,d] &= (n-2)+\text{mmuf}[n-2,d-1] \end{aligned} \tag{46}$$

which can be simplified to

$$\begin{aligned} \text{mmuf}[n,d] &= \sum_{i=1}^{d-1} n-2i \\ &= (n-d)(d-1)+1. \end{aligned} \tag{47}$$

Equation (47) assumes that $n/2 \geq d-1$. For values below this threshold, the $\text{mmui}[n]$ function must be used. From this expression the $\text{pmuf}[n,d]$ function can be derived to be

$$\text{pmuf}[n,d] = n\sum_{i=1}^{\log n} \frac{\text{mmuf}[2^i,d]}{2^i} . \tag{48}$$

Due to the switching between marry algorithms depending on the size of n, Eq. (48) cannot be written in closed-form. An approximate form for $n \gg d$ is

$$\text{pmuf}[n,d] \simeq (d-1)n\log n-(n-1)(d^2-d-1) . \tag{49}$$

The dependency is now $O(n\log n)$. The function has a strong dependency on d, as seen in Figure 69, where, multiple curves for different dimensionality are shown, along with a curve for infinite dimension.

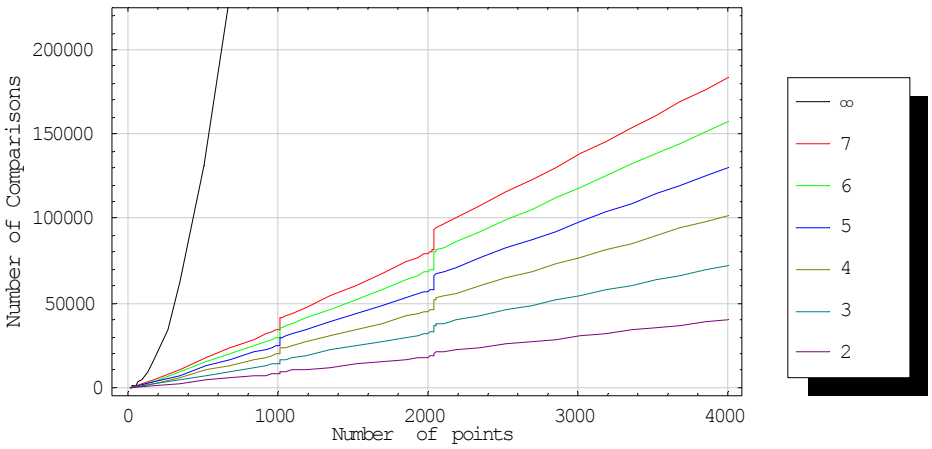

Figure 69: Number of comparisons as a function of number of points and dimension. Also, the infinite dimension function is shown.

Finally the maximum unbalanced case can be compared to the balanced case for varying dimensions and numbers of points in Figure 70.

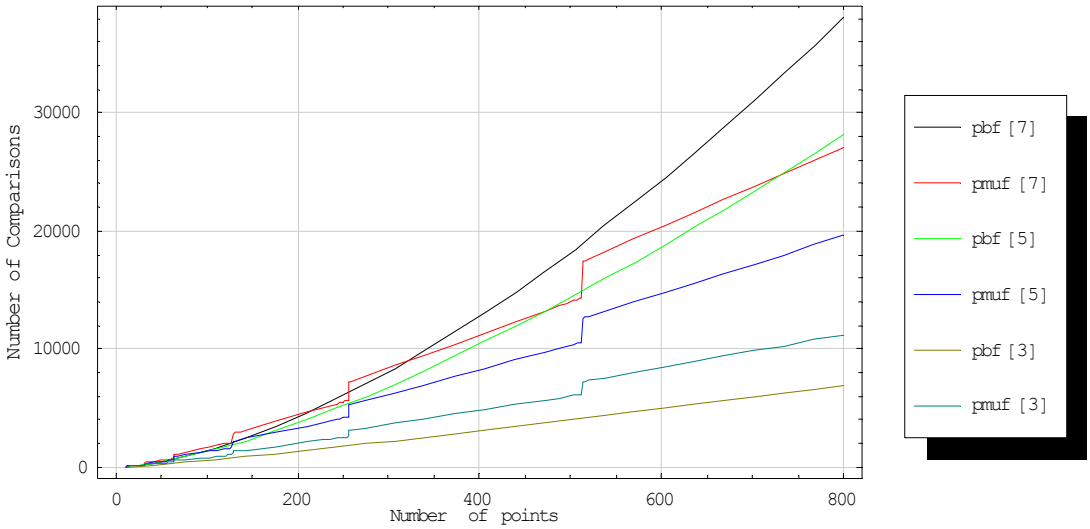

Figure 70: Comparison of the pbf[ ] function and the pmuf[ ] function for varying problem size and dimension

6.4 Expected Performance of the All Points Pareto Case

While clearly the worst case performance is encountered in the generalized unbalanced case with $\alpha \simeq 0.9$, in reality if the points are distributed randomly, the theoretical worst case will not be encountered. One can compute, for randomly distributed points, the distribution of α for a fixed problem size. For example, let $n = 20$ and calculate the distribution of α assuming randomly distributed points. The distribution is hypergeometric and is shown in Figure 71.

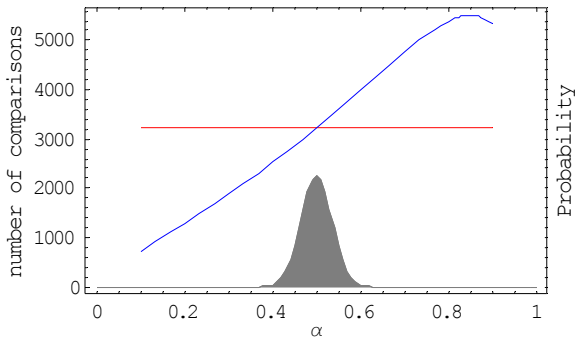

Figure 71: Comparing the pguf[ ] function and the pbf[ ] function for varying alpha and problem size of 200, d=7, with probability distribution of alpha shown in gray.

There are two things to note. First, the support for the distribution of α is primarily in the vicinity of the value of 0.5 and is symmetrical. Second, the function $\text{mguf}[n,\alpha,d]$ is approximately linear about the point $\alpha = 0.5$. This implies that a reasonable approximation for the expected number of comparisons is

$$\begin{aligned} E_\alpha[\text{pguf}[n,\alpha,d]] &\simeq \text{pguf}[n,E[\alpha],d] \\ &= \text{pguf}[n,.5,d] \\ &= \text{pbf}[n,d]. \end{aligned} \tag{50}$$

Another argument in favor of the approximation is that the worst case would require every run of the Marry algorithm to occur with $\alpha \simeq 0.9$, which would have a small probability of occurrence with any randomness in the distribution of the points.

6.5 Comparison of Empirical Results with Analytical Results

Figure 72 shows results of both experiments and analyses in estimating the number of comparisons for problems with all points nondominated, varying problem size, and varying dimension. For the analytical results the balanced case function $\text{pbf}[n,d]$ was used. Experimental results were derived by using the RDP algorithm to generate data of dimension ranging from 2 to 20. The results show that the experimental values approach the analytical values as the dimension increases.

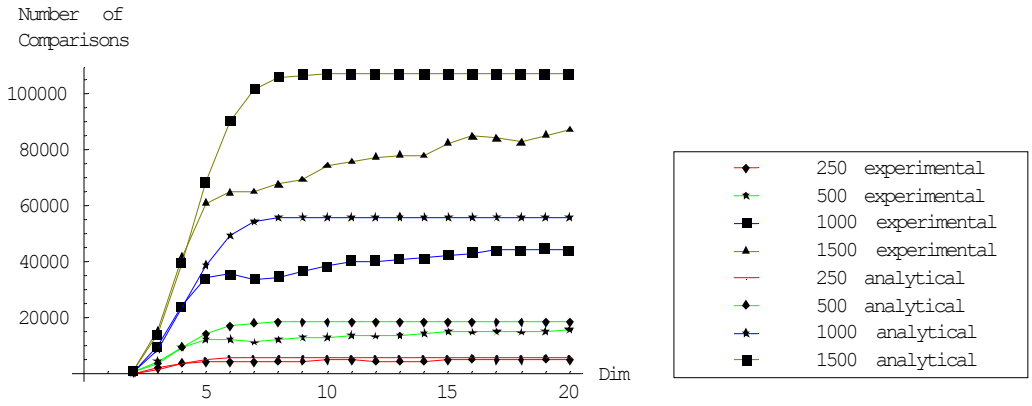

Figure 72: Comparison of experimental and analytical estimates of number of comparisons for different problem sizes and different dimensions

Extending into the higher dimensions shows that the experimental results continue to approach the analytical worst case balanced results. Figure 73 shows the approach of the variables to the worst case result. Again, the RDP algorithm was used to generate the test data.

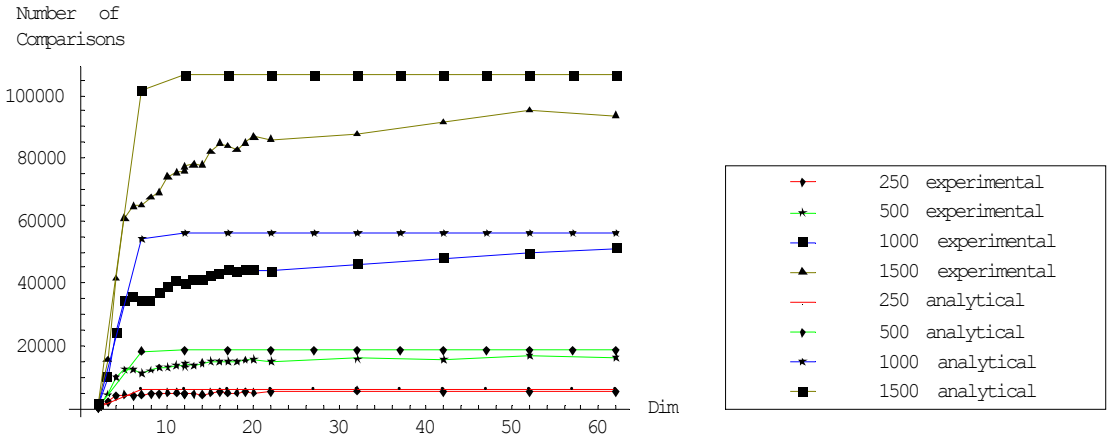

Figure 73: Experimental and analytical results up to dimension of 60

By generating data sets with all points Pareto and of sizes ranging from 10 to 600 and dimension from 20, one can estimate a scaling coefficient α to multiply with pbf[ ] function to improve the fit for each dimension. Appendix A lists the results of the curve fits. Figure 74 shows the fit between the pbf[ ] function scaled by α and the data used to fit the α coefficient. Shown are dimensions 3 through 18.

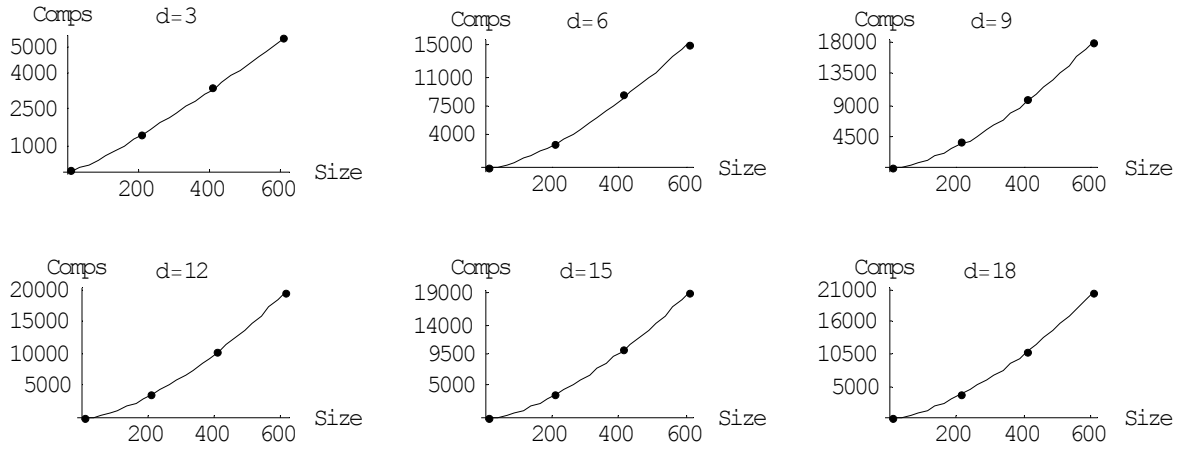

Figure 74: Data versus curve fits for the $\alpha\,\text{pbf}[\bullet]$ function, with varying dimension

Finally, Figure 75 repeats Figure 73 but with the $\text{pbf}[n,d]$ value multiplied by the mean value of 0.78, resulting in better correspondence with the experimental data. This estimate can be used as an approximate value of the number of comparisons required for random data with known dimension and all points nondominated.

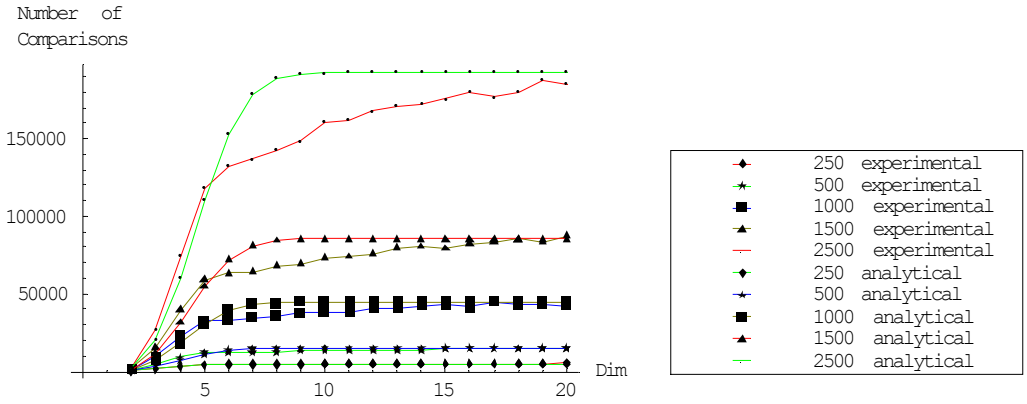

Figure 75: Comparison of empirical data with the analytical estimate gained by multiplying pbf[n,d] by .85

As for reasons for the difference between the balanced worst case estimates and actual experiments, one key source of difference is due to the fact that most function calls to the Marry algorithm are in fact unbalanced. Figure 76 shows a histogram with the relative imbalance of the input to the Marry algorithm plotted versus the number of function calls. The imbalance I was calculated via the following formula:

$$I = 1 - \frac{2x}{n} \tag{51}$$

where x is the number of points entered as the first argument, and n is the sum of the two arguments. In this figure, only function calls with a total data input greater than a size of 35 are shown. The data is from a run with 1000 point, 6 dimensional input data that has all points Pareto. Recalling that the Marry algorithm takes as arguments two sets of points, an imbalance of 1 indicates that of n points provided as argument to the Marry algorithm, all of them were to the second argument. An imbalance of -1 indicates all points were to the first argument. An imbalance of 0 indicates the two arguments were equal in size. Any function call for which the imbalance is 1 or -1 requires no comparisons.

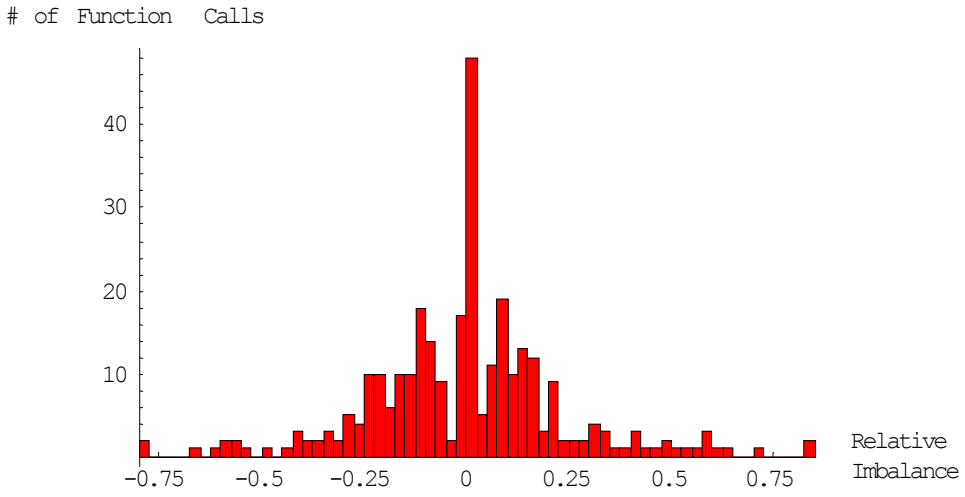

Figure 76: Plot of the relative imbalance of a Marry function call versus the number of calls, for a data set with 1000 points, 6 dimensions, all Pareto, only functions with data input size > 35

Figure 77 shows the histogram from the same data set but looking at function calls with arguments of all sizes. Notice that the number of function calls for the functions with smaller input is far greater and that the dispersion for the smaller calls is broader. There are spikes at 0, -1, +1, and also +/- ½ and +/- 1/3. These are due to the preponderance of function calls with arguments of total size only 2 or 3.

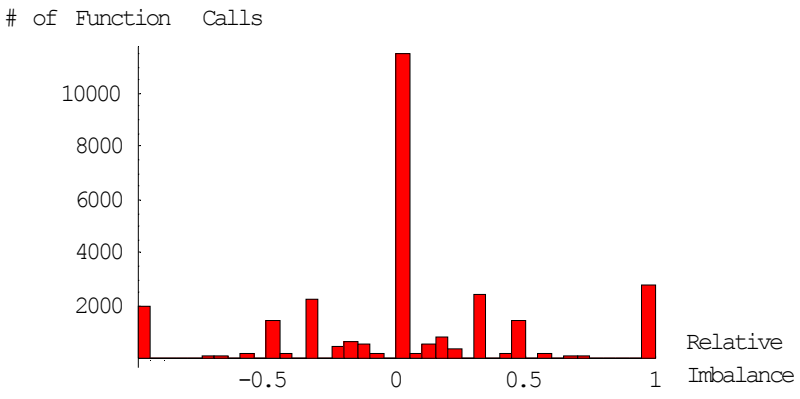

Figure 77: Plot of the relative imbalance of a Marry function call versus the number of calls, for a data set with 1000 points, 6 dimensions, all Pareto, functions of all sizes

6.5.1 Model of Run Time for DC Algorithm

A call to the DC algorithm involves recursive calls to the DC algorithm with diminishing problem size and/or lower dimension. This recursion results in more complex calculations to determine the number of subroutine calls and the size of the data passed to those subroutine calls. In a manner similar to the process for computing the number of comparisons in the DC algorithm, consider the Marry algorithm first.

Define the calculation mff[ ] (Marry Function Finite). The number of function calls needed in the Marry algorithm with a data set of size n evenly balanced into sets A and B can be determined via the recursion

$$\begin{aligned} \text{mff}[2,d] &= 1 \\ \text{mff}[n,2] &= 1 \\ \text{mff}[n,d] &= \frac{n}{2} + n\sum_{i=2}^{\log n} \frac{\text{mff}[2^{i-1}, d-1]+1}{2^i} \end{aligned} \quad . \tag{52}$$

The recursion can be resolved in closed-form and plotted, as shown in Figure 78.

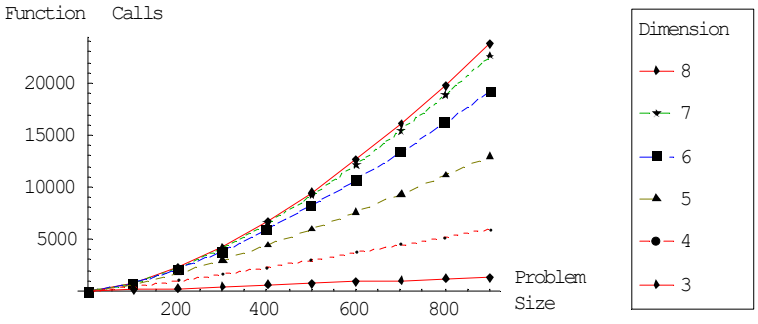

Figure 78: Number of routine calls in the Marry algorithm as a function of dimension and data size

The mff[ ] function overestimates the actual number of function calls as determined by experiment, and so one can adopt a simple model for the corrected function, $\alpha_d \times \text{mff}[n,d]$ and fit α for the different dimensions. Doing so with experimental data ranging in size from 10 to 600 points and dimension 3 to 20 results in a series of fitting coefficients listed in Appendix A. One reason for this overestimate is the assumption of balance in the calls, which was shown in the previous section (Figure 76 and Figure 77) to be unrealistic.

Defining pff[ ] (Pareto Function Finite), the number of function calls for the Pareto algorithm is then calculated through the recursion:

$$\begin{aligned} \text{pff}[2,d] &= 1 \\ \text{pff}[n,2] &= 1 \\ \text{pff}[n,d] &= \frac{n}{2} + n\sum_{i=2}^{\log n} \frac{\text{mff}[2^i, d-1]+1}{2^i} \end{aligned} \quad . \tag{53}$$

This can be plotted also, this time with dimension on the horizontal axis in Figure 79. One can see how for increasing dimension the number of function calls plateaus in a manner analogous to how the number of comparisons plateaus.

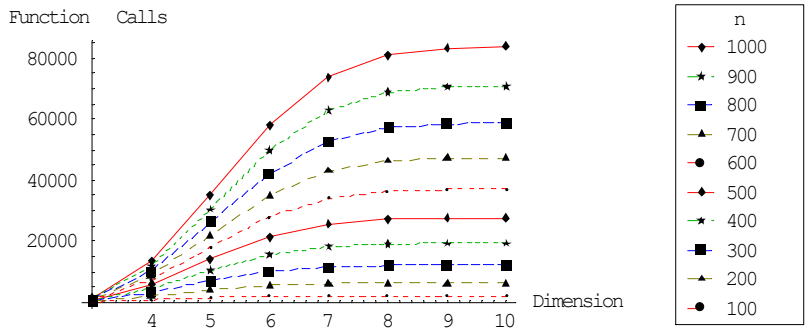

Figure 79: Analytical estimate of number of routine calls for Pareto algorithm as a function of problem size and dimension

The estimated number of function calls differs from the experimentally observed results, in that the analytic result overestimates the number of function calls. Again, coefficient multipliers can be determined to fit the pff[ ] functions to the data, as listed in Appendix A.

The functions for determining the amount of data passed in the Marry algorithm, mdf[ ] and in the Pareto algorithm, pdf[ ] are calculated in a similar manner:

$$
\begin{aligned}
&\mathrm{mdf}[2,d] = 2 \\
&\mathrm{mdf}[n,2] = n \\
&\mathrm{mdf}[n,d] = n\log n + n\sum_{i=2}^{\log n} \frac{\mathrm{mdf}[2^{i-1}, d-1]}{2^i} \\
&\mathrm{pdf}[2,d] = 2 \\
&\mathrm{pdf}[n,2] = n \\
&\mathrm{pdf}[n,d] = n\log n + n\sum_{i=2}^{\log n} \frac{\mathrm{mff}[2^{i}, d-1]}{2^i}
\end{aligned}
\qquad (54)
$$

The coefficients for each function are in Appendix A. The plot for the pdf[ ] function is shown in Figure 80.

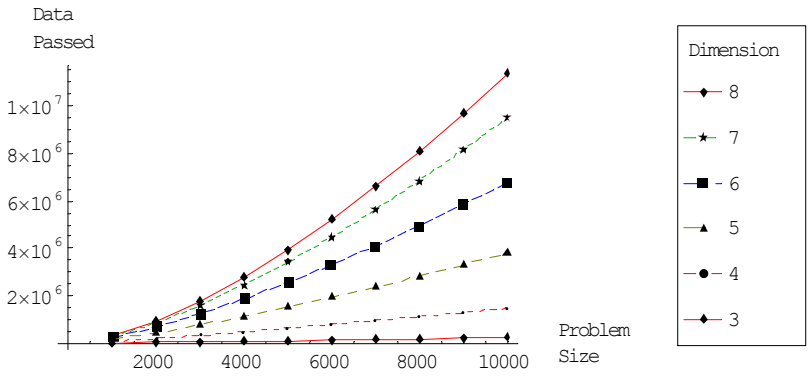

Figure 80: total amount of data passed to subroutines as a function of dimension and data size

The total time τ to run the DC and Marry algorithms, respectively, are then estimated to be

$$
\begin{aligned}
\tau_{mDC} &= k_S \times \mathrm{mff}[n,d] + k_D \times \mathrm{mdf}[n,d] + k_C \times \mathrm{mbf}[n,d] \\
\tau_{pDC} &= k_S \times \mathrm{pff}[n,d] + k_D \times \mathrm{pdf}[n,d] + k_C \times \mathrm{pbf}[n,d]
\end{aligned}
\qquad (55)
$$

The values for k_S, k_D and k_C will vary from one environment to another, both in their relative and absolute values. One must experimentally determine them for the best results. Using the Sun Java compiler on a Windows XP platform resulted in relative values between the parameters of:

$$
\begin{aligned}
k_S &= 1.2 \\
k_D &= 1 \\
k_C &= 1
\end{aligned}
\quad .
$$

Although the values of the k_D and k_C parameters will not be exactly the same for the SC and DC algorithms, they are approximately similar. Fixing them to be the same, one can plot the run times for the two types of algorithm to identify the break points (see Figure 81).

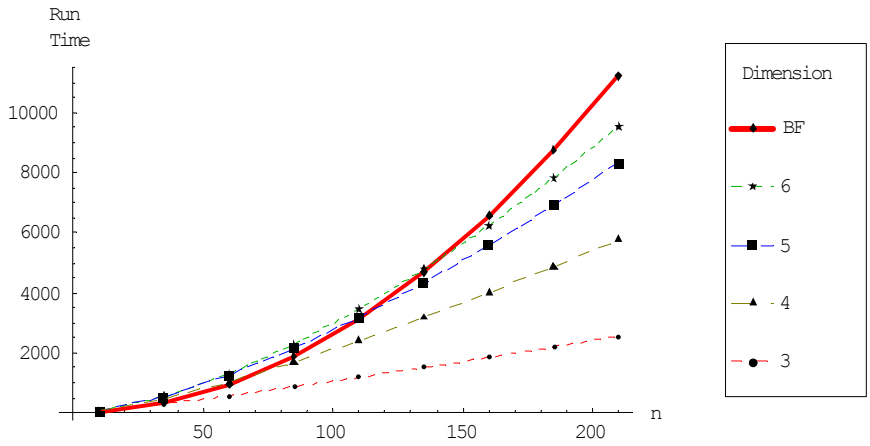

Figure 81: SC estimated run time and DC estimated run time for varying dimension

6.5.2 Variation of Number of Comparisons with Percentage of Points Nondominated

Experiments show that the relationship between the number of comparisons and the percentage of points that are nondominated is approximately linear. This holds for each of the three algorithms for generating random lattices. Figure 82 shows this approximately linear relationship, with individual plots for problems of dimensions 2 through 6. Note that the lines for dimension 5 and 6 overlay each other. This corresponds to the flat range in the curve of Figure 72, where the dimension has reached the limiting point such that the infinite dimension result holds.

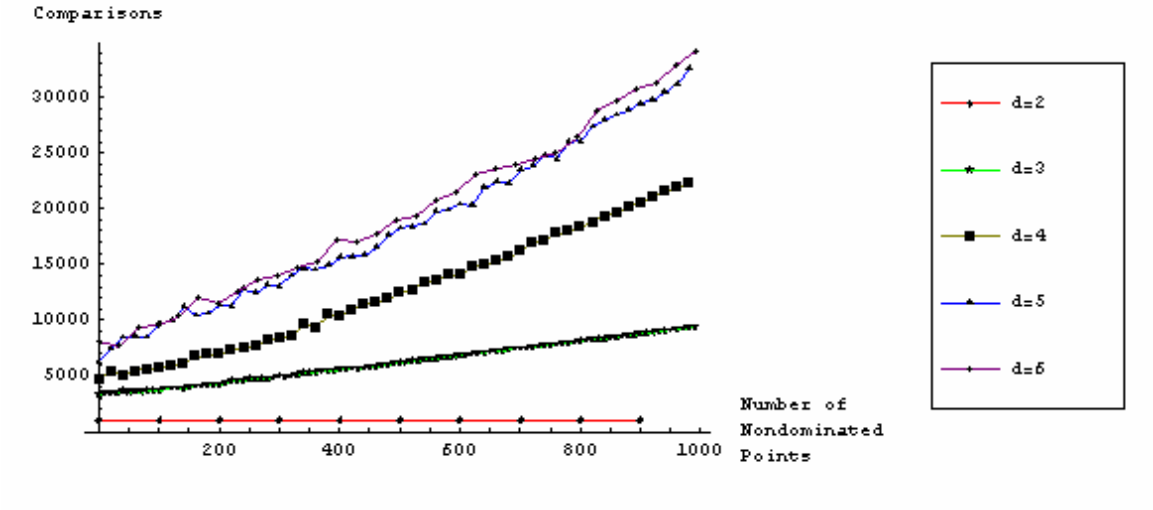

Figure 82: For problem size of 1000 points, the number of comparisons as a function of dimension and of the number of nondominated points

This trend holds for larger data sets. Figure 83 shows a data set of size 10,000 and dimension 4.

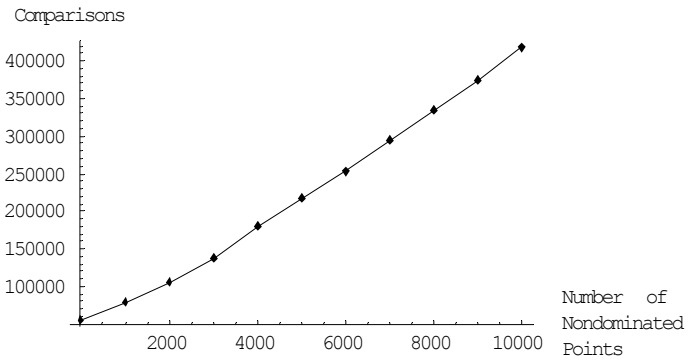

Figure 83: Number of comparisons for 10,000 points and varying nondominated points, d=4

An approximation, then, for the number of comparisons needed in the DC algorithm is

$$\frac{4}{5}\left(\frac{p}{n}\,\mathrm{pbf}[n,d]+\frac{n-p}{n}\,\mathrm{spp}[n,d]\right) \tag{56}$$

where p is the number of nondominated points, and the function spp[n,d] is defined to compute the number of comparisons required to work through a data set of size n, dimension d, and only one point nondominated. The 4/5 is the approximate fractional version of .78.

The analytical estimate of the spp[ ] function proves to be very sensitive to assumptions about the distribution of the data. The two limiting cases are (1) that all points are otherwise nondominated if the single dominant point is removed, and (2) that the data has the form $\{\{1,\ldots,1\},\ldots,\{N,\ldots,N\}\}$ so that every subset of the data has a single dominant point. The expression for the first case takes the form

$$\begin{aligned}&\mathrm{spp1}[n,d]=\mathrm{spp1}\left[\frac{n}{2},d\right]+\mathrm{pbf}\left[\frac{n}{2},d\right]+\frac{n}{2}\\&\mathrm{spp1}[2,d]=1\end{aligned} \tag{57}$$

By observation, it can be seen that spp1[ ] will be more than one half the value for pbf[ ].

Assuming that the data is randomly distributed independently in each of the dimensions, one can compute the probability that a data set would arise that has a single nondominated point, and this probability proves to be very small. The probability for a data set of size N and dimension d equals

$$\left(\frac{1}{N}\right)^{d-1}. \tag{58}$$

For a data set of 100 points and three dimensions, the probability of a randomly selected distribution having only one Pareto point is 1 in 10,000. More likely, the distribution of data for a data set with one dominant point would show correlation between each of the dimensions. The assumptions made in Eq. (57) err to the

other side, in that except for the single dominant point the other points are inversely correlated. Therefore the spp1[ ] model most likely overestimates typical data sets.

Looking at the second case where the dimensions are exactly correlated with each other, one can derive the recursive relationship

$$\begin{aligned} \text{spp}\,2[n,d] &= 2\,\text{spp}\,2\left[\frac{n}{2},d\right]+1 \\ \text{spp}\,2[2,d] &= 1 \end{aligned} \tag{59}$$

which simplifies to the closed-form expression

$$\begin{aligned} \text{spp}\,2[n,d] &= n-1 \\ &\simeq n \end{aligned}. \tag{60}$$

The assumptions implicit in Eq. (59) are more realistic than those made for Eq. (57), and so spp2[ ] is a better candidate function for spp[ ]. Note that the data sets used for generating Figure 82 and Figure 83 are intermediate in their distribution form, as can be seen in Figure 84. This results in experimental run times well above the estimates made using spp2[ ].

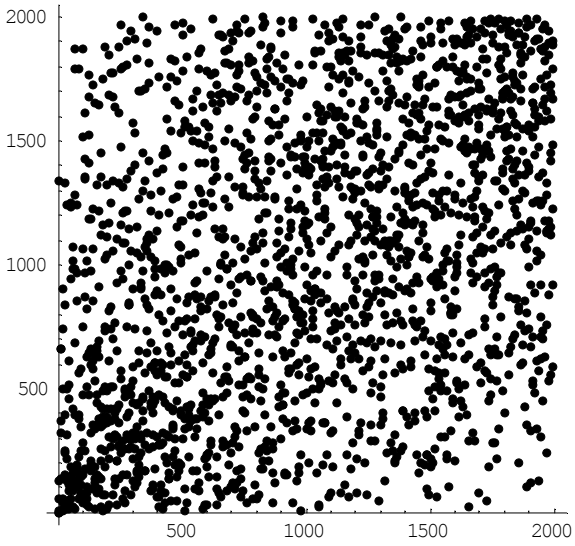

Figure 84: Data set with single Pareto point, as generated by the RDP algorithm

Therefore the final form of estimator for the run time of data sets with N points of which p are Pareto is

$$\frac{4}{5}\left(\frac{p}{n}\,\text{pbf}[n,d]+(n-p)\right), \tag{61}$$

with the caveat that the actual run time experienced depends heavily on the structure of the data.

6.6 The Dependence of DC Algorithm Performance on the Ordering of Dimensions

Interestingly, it is possible to get vastly different performance out of two data sets that both are constituted of only nondominated points. Look at the following data set of 6 dimensions:

$$\begin{Bmatrix} \{1,1,1,1,1,1000\} \\ \{2,2,2,2,2,999\} \\ \vdots \\ \{1000,\cdots,1\} \end{Bmatrix} \tag{62}$$

The number of comparisons needed to execute the algorithm is 11,207. Conversely, look at this data set, identical to the previous except for permuting the order of the columns:

$$\begin{Bmatrix} \{1000,1,1,1,1,1\} \\ \{999,2,2,2,2,2\} \\ \vdots \\ \{1,\cdots,1000\} \end{Bmatrix} \tag{63}$$

The number of comparisons for this data set is only 536, which is less than the number of points in the problem. Also, this is much less than the set in Eq. (62), with the exact same number of Pareto points, by a factor of 20. The reason for the improved performance against the data set lies in the marriage step of the algorithm. For the second data set, upon entry into the marriage step, the two sets $\mathbf{Y}_1$ and $\mathbf{X}_2$ will be empty, and the algorithm stops at that point. For the first data set, during the marriage steps the sets $\mathbf{Y}_1$ and $\mathbf{X}_2$ will be occupied, and the sets $\mathbf{Y}_2$ and $\mathbf{X}_1$ will be empty until the 5^{th} dimension is reached. This results in the large increase in the number of comparisons.

In order to overcome this lack of determinism and disparity in run time, the algorithm can be modified to randomly choose the next dimension (column of data) upon which to split. The revised, randomized versions of the DC and Marry algorithms are shown with changes in the algorithm highlighted in bold and italics.

Algorithm RandomDC Given a set **Z** of points in a d -dimensional space, find the nondominated points that are in the set.

RDC-1	[If dimension of data is 2, call L2D on data and return results]
RDC-2	[If size of problem is below limit, call SC on data and return results]
RDC-3	[Else divide into two problems of equal size] Split **Z** into two sets of points **X** and **Y** on the median element of ***a randomly chosen column***.
RDC-4	[Call DC on subproblems] Determine **XX** = *RDNC*(**X**) and **YY** = *RDNC*(**Y**)
RDC-5	[Reduce dimension] Drop the ***chosen*** columns of **XX** and **YY** to form **XXX** and **YYY**
RDC-6	[Call MARRY to cull remaining dominated points in the inferior set] Determine $\mathbf{XXX}' = \text{MARRY}(\mathbf{XXX}, \mathbf{YYY})$.
RDC-7	[return the results] Return $\mathbf{YYY} \cup \mathbf{XXX}'$.

Similarly, the MARRY[ ] algorithm is modified as follows.

Algorithm RandomMarry

RMARRY-1	[If data dimension is 2D, call MARRY2D and return results]
RMARRY-2	[If size of problem is below limit, call MARRYDIRECT on data and return results]
RMARRY-3	[Else divide into two problems of equal size, and recursively call MARRY] Choose a cut plane based on ***a randomly chosen column***, and so that $\lvert\mathbf{X}_1\rvert + \lvert\mathbf{Y}_1\rvert = \lvert\mathbf{X}_2\rvert + \lvert\mathbf{Y}_2\rvert$. Call $\mathbf{X}_1' = \text{RMARRY}(\mathbf{X}_1, \mathbf{Y}_1)$ and $\mathbf{X}_2' = \text{RMARRY}(\mathbf{X}_2, \mathbf{Y}_2)$.
RMARRY-4	[Now drop a dimension, and recursively call MARRY] Drop the ***chosen columns*** of $\mathbf{X}_2'$ and $\mathbf{Y}_1$ to form $\mathbf{XX}_2'$ and $\mathbf{YY}_1$ and call $\mathbf{X}_2'' = \text{RMARRY}(\mathbf{XX}_2, \mathbf{YY}_1)$.
RMARRY-5	[Form union of results and return] Return $\mathbf{X}_1' \cup \mathbf{X}_2''$.

The other supporting algorithms remain unchanged. The results of randomizing the algorithm can be shown against the first data set in Eq. (62), as shown in Figure 85. While there is still significant variance in the number of comparisons required to complete the algorithm, the difference between best and worst case is much less than a factor of 20.

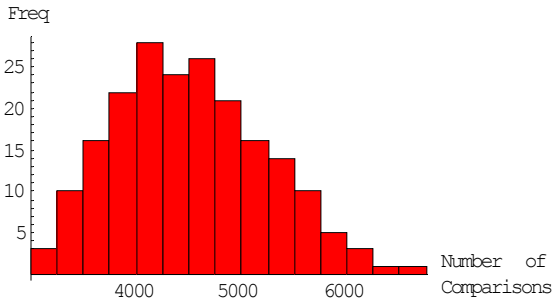

Figure 85: Performance of the RandomDC algorithm with 200 runs

The disparity in run times for the deterministic algorithm suggests that prior analysis of the data could result in an optimum choice of columns to cut on. The approach would be to determine, for each possible plane in the *d*-dimensional problem, the correlation coefficient of the data. Then choose the plane that has the correlation coefficient closest to -1, and use the two axes that form the plane as the first two columns in the DC algorithm. This is not explored further in the thesis, instead remaining as future possible research, discussed in the final chapter. Meanwhile, in the next chapter the hybrid algorithm is introduced next.

CHAPTER 7

Hybrid DC Algorithm

This chapter develops the hybrid divide & conquer (HDC) algorithm that combines aspects of both the SC algorithm and the DC algorithm. The first motivation in assembling the hybrid algorithm is that, while the DC algorithm is asymptotically more efficient than the SC algorithm, it also has a larger computational overhead, and so for smaller problem sizes it is better to switch to the simpler algorithm. The second motivation is that initial analysis of the data can indicate whether to first use a pass of the SC algorithm to efficiently eliminate from consideration a large number of points at the outset, then switch to the DC algorithm if warranted.

In the hybrid algorithm, the basic flow is to

1. Immediately remove all points that are guaranteed dominated;
2. Apply the SC algorithm if there are points guaranteed to dominate more than some cut off limit;
3. Apply DC algorithm to remaining points;
4. At each branching of the DC algorithm, if the problem goes below a certain size, switch to the brute force algorithm, either SC or MarryDirect.

Each of these is described in the following sections.

7.1 Immediately Remove All Points Guaranteed Dominated

Previously it was noted by using the LLH form that it is possible to identify points that must dominate others. This property is mirrored in that it is possible to identify points that must be dominated by others. In particular, for a problem of size N and dimension d, a point must be dominated if the sum of each of its dimensions is less than $N+d-1$ if larger values are preferred, or greater than $(d-1)(N+1)$ if smaller values are preferred. One can look at the expected percentage of points that meet this criteria as a function of the dimension of the problem. This percentage is approximately equal to the volume of the portion of a hyper lattice that satisfies the property that the sum of each point's dimensions is less than $N+d-1$, which is itself approximately equal to the ratio of the volume of a right simplex of dimension d and unit edge length to a hypercube of dimension d and unit edge length. This ratio is $1/d!$. Figure 86 shows the experimental results based on a data set of $N=10,000$ and plot of the function $1/d!$ together.

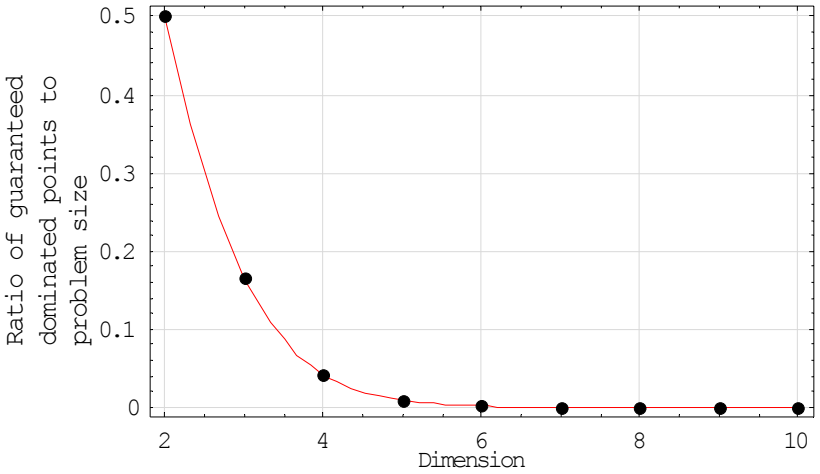

Figure 86: Fraction of points guaranteed dominated as a function of dimension (points) and the function 1/d! (line)

Although Figure 86 shows that the likelihood for finding points that satisfies this property goes down rapidly with increasing dimensionality of the data, even for high dimensional data it still merits a test on the chance that the data has structure, especially due to the low computational cost of implementing it.

7.2 Breakpoint for Switching to Simpler Algorithm

To determine the breakpoint for switching to a simpler algorithm, one must first model the operating time of the algorithm. Three prime contributors to the algorithm run time are considered. The first is the overhead time of starting and ending a subroutine $\tau_f = f \times 1$, where f is a measure of the time to instantiate a function call. The second is the time that scales linearly with the amount of data passed to a subroutine, $\tau_h = h \times n$, where h is a scaling factor for internal data handling and linear sorting. The third is the time that scales proportional to the number of comparisons, $\tau_c = c \times$ #comparisons, where c is a scaling factor for comparisons. The values for f, h and c will vary from one environment to another, both in their relative and absolute values. One must experimentally determine them for the best results. Using the Sun Java compiler on a Windows XP platform resulted in relative values between the parameters of:

$$f = 1.2$$
$$h = 1$$
$$c = 1$$

In building the models for identifying the breakpoint, the worst case scenario of all points Pareto will be used. This is justified first by the fact that the hybrid algorithm will first run a Brute Force pass and remove the majority of points dominated, resulting in a data set with a high proportion of points nondominated. Second, the relationship between number of comparisons and proportion of points nondominated is

reasonably linear for both the DC and SC algorithms for proportions close to one. For the DC algorithm, the balanced case developed in Chapter 6 is used as the model.

Starting with the Marry algorithm with infinite dimensional data, define the function bfm[ ] (Brute Force Marry) to compute the run time for the Brute Force algorithm:

$$\text{bfm}[n] = c\frac{n^2}{4} + hn + f \ . \tag{64}$$

The $n^2/4$ term is predicated on the Marry algorithm having two sets of data input of $n/2$. If we first divide once, then run the brute force algorithm on the two subproblems, and the Brute Force algorithm on the marrying of the results of the subproblems (dcbfm[ ] = DC Brute Force Marry) we get a run time estimate of

$$\begin{aligned}\text{dcbfm}[n] &= 3\times\left(c\frac{(n/2)^2}{4} + h\frac{n}{2} + f\right) + hn + f \\ &= \frac{3c}{16}n^2 + \frac{5h}{2}n + 4f\end{aligned} \ .$$

The goal is to identify the point at which these two infinite dimension run time estimates are equivalent. So solving for the equation $\text{bfm}[n] = \text{dcbfm}[n]$ results in a quadratic equation in n that resolves to

$$n_{M\infty} = \frac{4\left(3h + \sqrt{3cf + 9h^2}\right)}{c} \ . \tag{65}$$

Replacing the scaling values with their numerical estimates gives $n_{M\infty} \simeq 26$.

The three dimensional case can be solved similarly. While the brute force time remains the same, the time for running a single iteration of the divide first gives

$$\begin{aligned}\text{dcbfm}[n,3] &= 2\,\text{bfm}[n/2,3] + \text{bfm}[n/2,2] \\ &= \frac{c}{8}n^2 + (2h + c)n + h + 4f - c.\end{aligned} \tag{66}$$

This change is due to the fact that $\text{bfm}[n/2,2]$ runs in linear time. Solving for the breakpoint for the 3D marry case leads to

$$n_{M3} = \frac{4\left(c + h + \sqrt{\frac{1}{2}c(c+3f) + \frac{5ch}{2} + h^2}\right)}{c}, \tag{67}$$

or replacing the scaling coefficients with their numerical values gives $n_{M3} \simeq 18$. So for data sets of size less than 18 points and of dimension equal to 3, one should execute the Brute Force algorithm.

To look at the 4D case, one has two choices of possible methods to compute the run time, both shown here.

$$\begin{aligned} \text{dcbfm}[n,4] &= 2\,\text{bfm}[n/2] + \text{dcbfm}[n/2,3] \\ \text{dcbfm}[n,4] &= 3\,\text{bfm}[n/2] \end{aligned} \tag{68}$$

The top option uses the 3D version of the bfm[ ] function for the dimension reduction step, while the bottom uses the brute force option for the dimension reduction step, making it identical to the infinite dimension case.

Since the dcbfm[n,4] function's breakpoint will be between the 3D and the infinite dimension case, then in the top option the value of $n/2$ is guaranteed to be less than 26/2=13, which is itself less than the breakpoint for the 3D function, and so the bottom option dominates. The result is that for 4D problems and higher, the breakpoint to use is the infinite dimension value of 26. For 3D, a value of 18 should be used.

For the DC algorithm, the 3D case is considered first. The Brute Force algorithm (bfp[n] = Brute Force Pareto) results in a run time of

$$\text{bfp}[n] = c\frac{n^2 - n}{2} + hn + f\,, \tag{69}$$

while the approach that has a single divide stage results in an estimate for the function dcbfp[n] (DC then Brute Force Pareto)

$$\begin{aligned} \text{dcbfp}[n,3] &= 2\,\text{bfp}\left[n/2\right] + \text{bfm}[n,2] \\ &= \frac{c}{4}n^2 + \left(\frac{c}{2} + 3h\right)n + 4f - c. \end{aligned} \tag{70}$$

The breakpoint is the value of n where the two functions are equivalent. Solving the equations gives

$$n_{P3} = \frac{2\left(c + 2h + \sqrt{3cf + 4ch + 4h^2}\right)}{c} \simeq 13\,. \tag{71}$$

So for the 3D case, if the data set is less than 13, the SC algorithm should be run.

The infinite dimension case is more complicated. At first glance, the approach would be to compare the brute force run time $\text{bfp}[n]$ with a divided approach $\text{dcbfp}[n] = 2\,\text{bfp}[n/2] + \text{bfm}[n]$. However, when comparing the two it turns out that the $\text{bfp}[n]$ approach dominates for all values of n. The reason is that the $\text{bfm}[n]$ approach is suboptimal for the values of n considered. Instead, the $\text{bfm}[n]$ function should be replaced with one that divides at least once, but possibly more times. Define the following functions for computing various run times for the Marry algorithm:

$$\begin{aligned} \text{dcbfm2}[n] &= 3\,\text{dcbfm}[n/2] + hn + f \\ \text{dcbfm3}[n] &= 3\text{dcbfm2}[n/2] + hn + f\,. \end{aligned} \tag{72}$$

Now the candidate run times to compare for the DC algorithm are

$$
\begin{aligned}
&\text{bfp}[n] \\
\text{dcbfp}[n] &= 2\,\text{bfp}[n/2] + \text{bfm}[n] + hn + f \\
\text{dcbfp2}[n] &= 2\,\text{bfp}[n/2] + \text{dcbfm}[n] + hn + f \\
\text{dcbfp3}[n] &= 2\,\text{bfp}[n/2] + \text{dcbfm2}[n] + hn + f \\
\text{dcbfp4}[n] &= 2\,\text{bfp}[n/2] + \text{dcbfm3}[n] + hn + f .
\end{aligned}
\tag{73}
$$

Plotting the difference between each of these (using the scaling coefficient values provided above) and the value of bfp$[n]$ over varying n shows, for each one, at what point the function becomes preferred to the strictly brute force approach. Candidate breakpoints are where the X-axis is crossed (Figure 87). The figure indicates that the dcbfp3$[n]$ function has the lowest breakpoint, since its point of crossing the X-axis is closest to the origin. Setting bfp$[n]$ = dcbfp 3$[n]$ gives a breakpoint of $n_{P\infty} \simeq 55$, so for the limiting infinite dimensional case one would use the SC algorithm if the data set has fewer than 55 points.

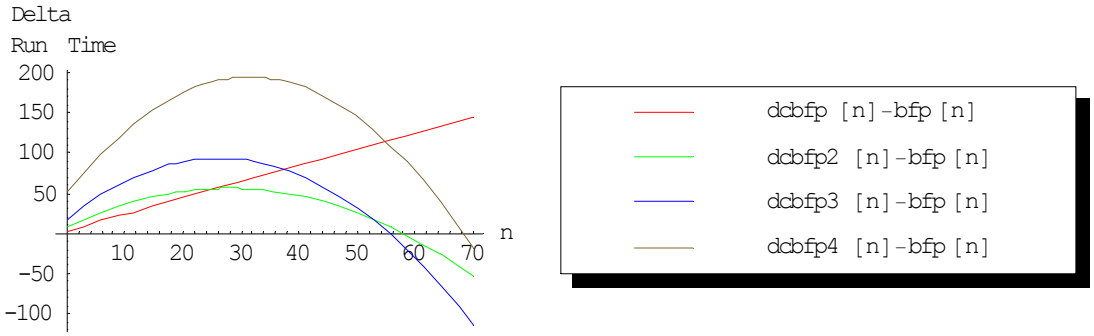

Figure 87: Comparison of Δ run times for differing numbers of divisions of the Marry algorithm

For the 4D case with the DC algorithm, a similar process of comparing differing amounts of dividing in the Marry portion of the algorithm leads to Figure 88, with a calculated value of $n_{P4} \simeq 33$.

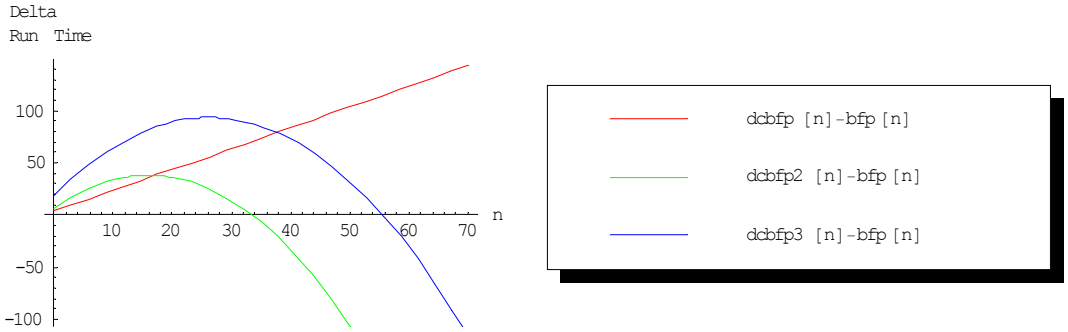

Figure 88: Comparison of Δ run times for the 4D case

For the 5D case, we know that the value of n_{P5} must be between the value for the 4D case of 33 and the limiting infinite dimension case of 55. Writing

$$
\text{dcbfp}[n,5] = 2\,\text{bfp}[n/2] + \text{marry}[n,4] + hn + f \tag{74}
$$

the question is, how to implement the marry$[n,4]$ function? Since the value of n will be greater than 26, the breakpoint for the 4D marry algorithm, we know it will divide itself at least once, and so

$$\text{marry}[n,4] = 2\,\text{bfm}[n/2] + \text{marry}[n/2,3] + hn + f \ . \tag{75}$$

The bfm$[n/2]$ terms are dominant since we know that $n/2 < 55/2 \simeq 27$, which is close enough to the breakpoint of 26 to assume the bfm procedure will have the same run time as one that might divide again. As for the marry$[n/2,3]$ term, since the divide point for the 3D marry algorithm is 18, and we know that $16.5 < n/2 < 27.5$, the marry term will involve another divide, so that

$$\text{marry}[n/2,3] = 2\,\text{bfm}[n/4] + \text{marry}[n/4,2] + hn + c \ . \tag{76}$$

The marry$[n/4,2]$ term will use the linear algorithm, and so the full run time estimate of the dcbfp$[n,5]$ algorithm is

$$\text{dcbfp}[n,5] = \frac{13c}{32}n^2 + \left(\frac{21h-c}{4}\right)n + 10f - c \ . \tag{77}$$

Solving for bfp$[n]$ = dcbfp$[n,5]$ and using the coefficient values give a result of $n_{P5} = 50$. For higher dimensions than 5, the infinite dimension value for the breakpoint should be used.

To summarize, Table 12 shows the breakpoints for the Marry and the DC algorithms for the varying dimensions.

Table 12: Breakpoints for Marry and DC algorithms

Dimension	**Marry**	**DC**
3	18	13
4	26	33
5	26	50
6+	26	55

Experimental results indicate that using the hybrid algorithm results in significant savings in total run time. Figure 91 shows results for 6-dimensional data of varying size.

These are for the total number of points in the data set, recalling that the relationships used to create the table assume that all points are Pareto. The values for breakpoints should approximately hold even if only subsets of the points are Pareto, since the relationship of comparisons to percentage of points Pareto are approximately linear for both the SC and DC algorithms.

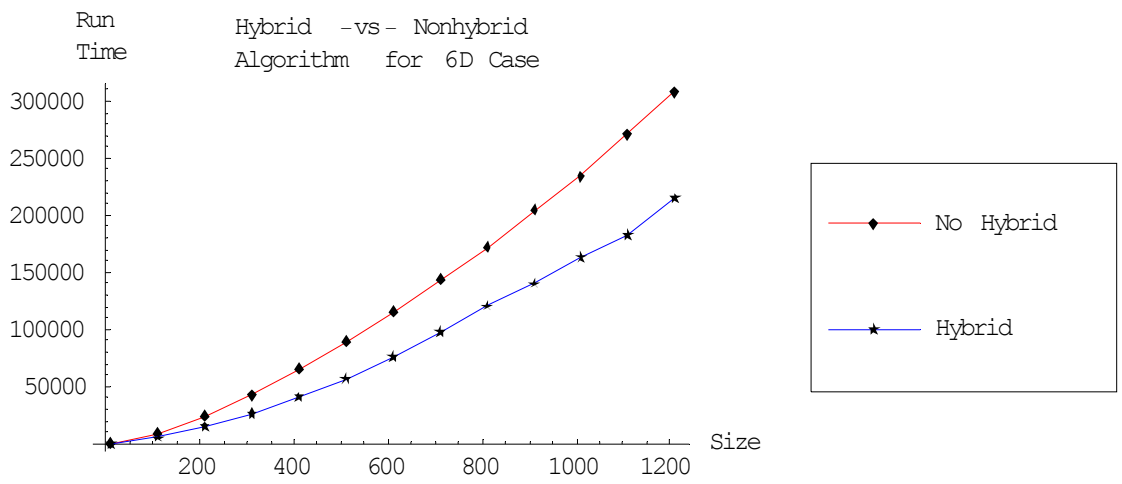

Figure 89: Comparison of hybrid and non-hybrid algorithm for 6Dcase

The percent reduction in run time is shown in Figure 90. The reduction appears to approach a value of approximately 30%.

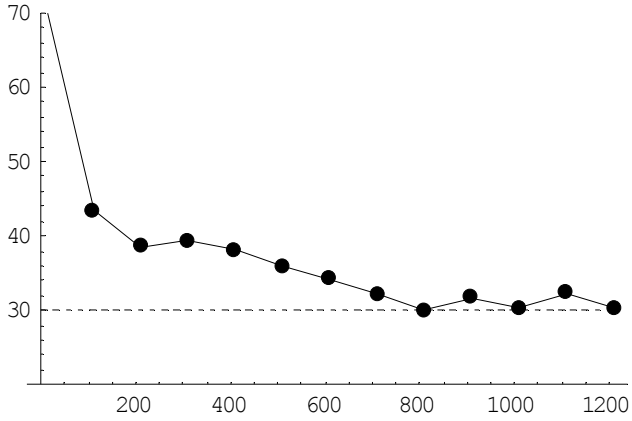

Figure 90: Percent reduction in run time for 6D data, varying data set sizes

Fixing the size of the data set at 600 points and varying the dimension provides the graph shown in Figure 91.

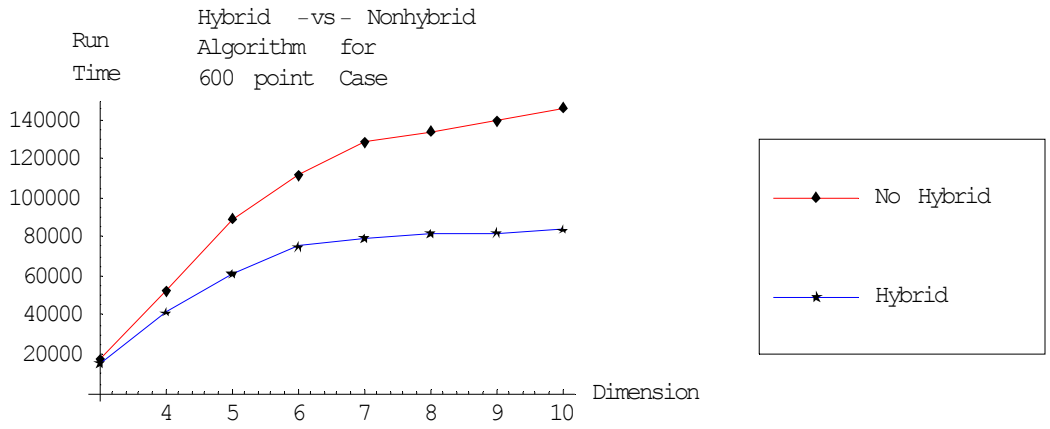

Figure 91: Comparison of run times of hybrid and nonhybrid algorithms for problem size of 600 and varying dimension

The corresponding percent reduction is shown in Figure 92 for each dimension.

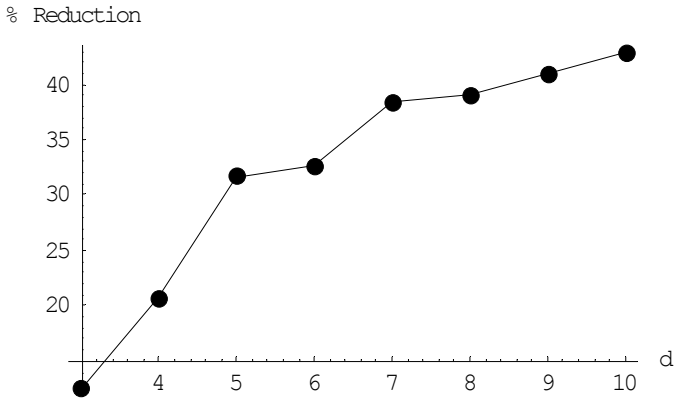

Figure 92: Percent reduction in run time of hybrid with respect to nonhybrid algorithm for problem size of 600 and varying dimension

7.3 First Pass Trade

The last relationship to build is for the tradeoff between either a single pass of the SC algorithm followed by the DC algorithm, or directly going to the DC algorithm. The tradeoff between these two occurs when the run time for a single pass of the SC algorithm is less than the run time expected to be saved by its execution.

The SC algorithm takes $n-1$ comparisons for a single pass through data of size n. If we indicate the number of points to be removed as Δ, then we can estimate the number of comparisons required by running the DC algorithm with a full data set versus running with a data set of $n-\Delta$, and if the difference is greater than the cost of a single pass through the SC algorithm, then the single pass is warranted. To compute this breakpoint, define a DC run time estimator to be

$$\text{pdc}[n,d] = 0.66\left(0.73\,\text{pff}[n,d] + 0.77\,\text{pdf}[n,d] + 0.78\,\text{pbf}[n,d]\right), \tag{78}$$

where the coefficients to the estimates for the number of function calls, data passed, and comparisons are derived by using $k_S = 1.2$ and $k_d = k_C = 1$. The multipliers to each of the functions are taken from the Appendix A, while the leading .66 is due to the application of the mixed DC and Brute Force algorithm, which is developed in the previous section. Since the relationship of the run time with respect to the number of Pareto points is approximately linear, the number of comparisons that would be avoided is approximately

$$C = \frac{\Delta}{n}\text{pdc}[n,d] \tag{79}$$

Since it will cost one function call and $n-1$ comparisons and $n-1$ data passed to remove Δ points, we can solve for Δ to get

$$\Delta = \frac{2(n-1)n}{\text{pdc}[n,d]} \tag{80}$$

The minimum value for Δ is $\Delta = n - S(\mathbf{z}) + d - 1$. Replacing Δ in Eqn. (80) results in the inequality that determines whether or not to perform the single pass, where if the inequality holds true, a single pass is merited:

$$S(\mathbf{z}) \le n - \frac{2(n-1)n}{\text{pdc}[n,d]} + d - 1. \tag{81}$$

Figure 93 shows the break-even point as a function of data set size and dimension. So, for example, with $d = 4$ and $n = 8000$, the break-even point is approximately 150.

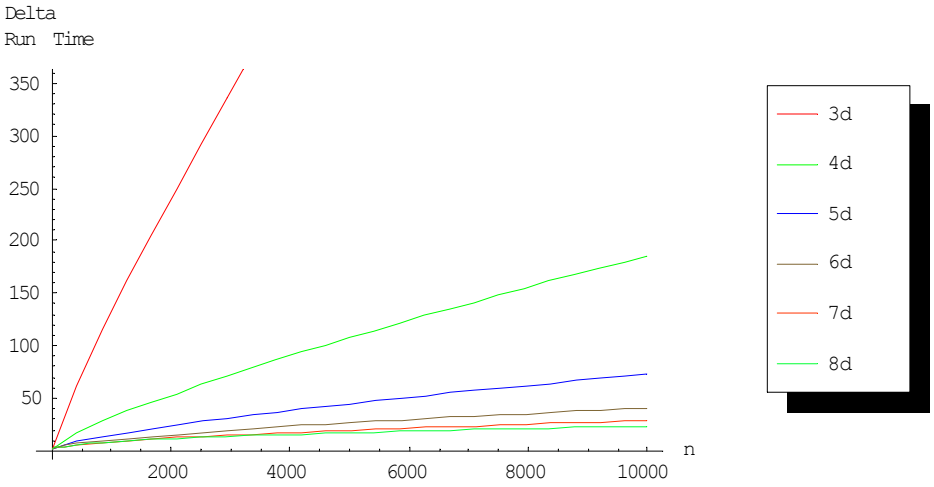

Figure 93: Plot of the break-even point for varying data size and dimension, above which one should first run the SC algorithm

Figure 94 shows the same results but with dimension on the X axis.

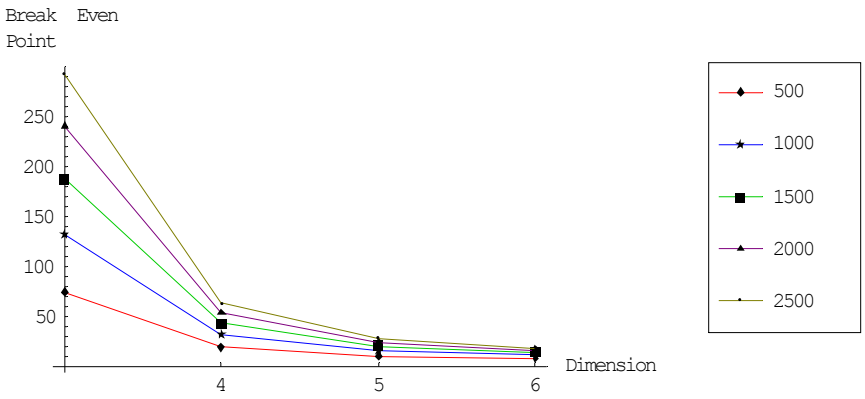

Figure 94: Break-even point as a function of dimension and data set size

Figure 95 shows results of the DC algorithm alone, the hybrid algorithm without using a single pass of SC, and the full hybrid algorithm using a single pass. All runs were with 400 points, of which only 5 were Pareto. The horizontal axis is a *data dispersion factor*, which is one of the inputs to the RDP algorithm used to generate the test data. Recall from Section 4.3 that as the dispersion factor grows larger, that probability that a dominated point is dominated by more than one point increases, and for very high dispersions each Pareto point dominates all dominated points. Note the independence of the DC and the hybrid to this factor.

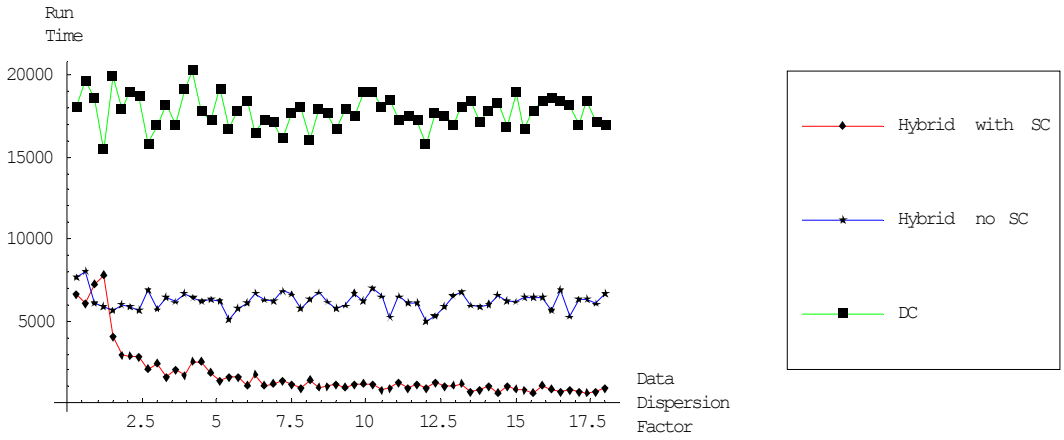

Figure 95: Comparison of DC, hybrid with single pass of SC, and hybrid with single pass of SC, for 5D data set of 400 points, only 5 points Pareto. The horizontal axis correlates to the number of points each Pareto points dominates, with more points as you move right on the axis.

Figure 96 shows a comparison between the three algorithms for a data set of 5D, with a ratio of 1/20 of the points Pareto, with a high data dispersion value, and varying numbers of points in the data.

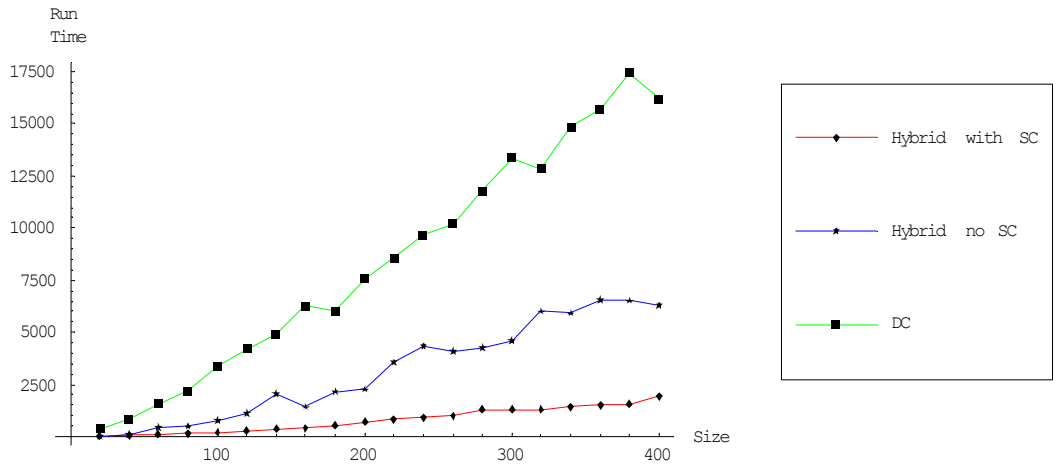

Figure 96: Varying the number of points, for 5D data, 1/20 of points dominant, high dispersion of data

Figure 97 shows a 5D data set with 500 points, high dispersion, and varying number of Pareto points. Interestingly, the hybrid with one pass of SC performs better for a significantly wider range of possible values than would be predicted by the analytical model, indicating that additional effects are at work, discussed below.

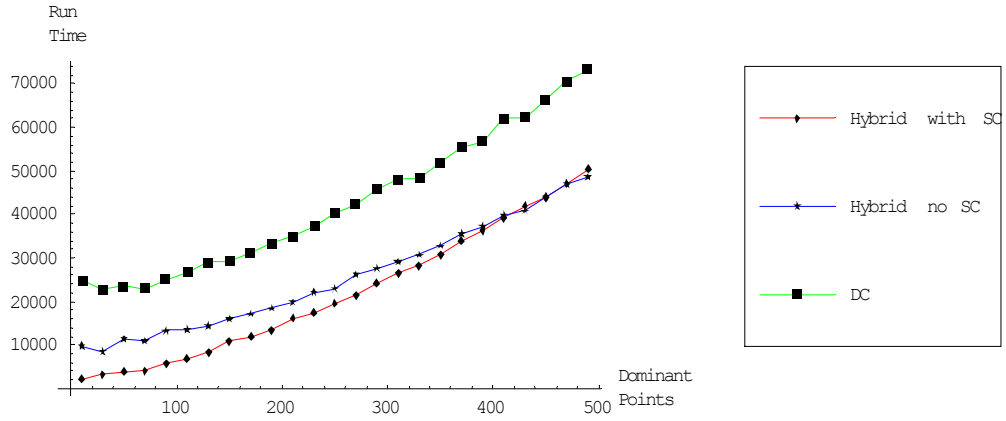

Figure 97: 5D data set with 500 points, varying number of dominating points, high dispersion in data

Figure 98 is a repeat of Figure 97, but with lower dispersion. Again, for all but the data sets with almost all points Pareto, the first pass of the SC algorithm is merited.

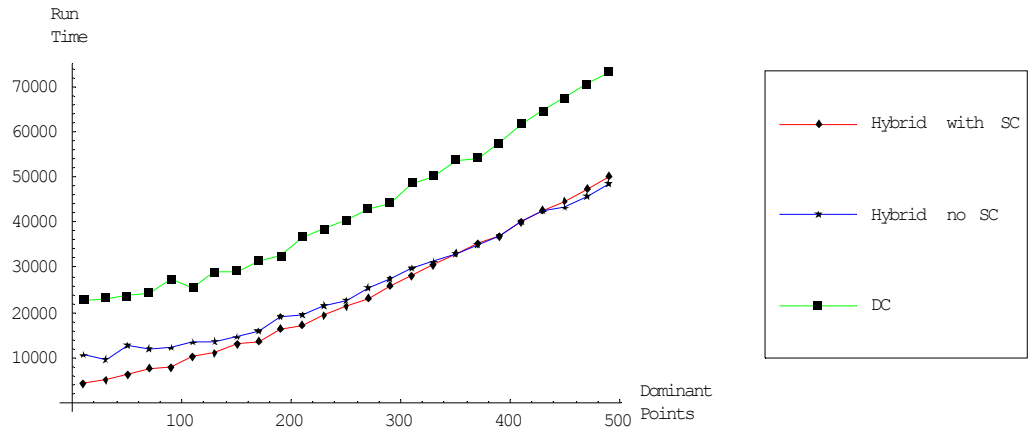

Figure 98: 5D data set with 500 points, varying number of dominating points, low dispersion in data

While the rationale for this unexpected performance is not investigated in detail, a cursory examination of the data sets as the algorithm executes suggests that a single pass of the SC algorithm shapes the data in such a way that the DC algorithm performs especially well. If the data is initially in the shape of a hypersphere or hypercube, then a single pass of the SC algorithm can be expected to remove a center core of points, leaving the remaining points distributed in a hypertoroid. This shape possibly lends itself to the DC algorithm by allowing it to avoid the comparison of points that are on opposite sides of the toroid.

7.4 Full Description of Hybrid algorithm

The hybrid algorithm is presented here in three parts. The first is the HYBRID algorithm itself, which then serves as a wrapper for the HYBRID_DC algorithm and the HYBRID_MARRY algorithm.

Algorithm HYBRID Given a set **Z** of points in a d -dimensional space, find the nondominated points that are in the set.

HYBRID-1	[If dimension of data is 2, call LC1 on data and return results]
HYBRID -2	[Remove all points guaranteed dominated] Test each point, if the sum of its elements are greater than $(d-1)(N+1)$ delete the point.
HYBRID -3	[Test to determine single pass of the cull algorithm] Define S as the sum of each of a point's elements, and $S_{\max}$ as the maximum S from among all points. Then if, $k_S + (k_D + k_C)(N-1) < \tau_{pDC}(n,d) - \tau_{pDC}(n - S(\mathbf{z}^*) + d, d)$ do one pass of the SC algorithm.
HYBRID - 4	[Call HYBRID_DC] Call the hybrid DC algorithm.
HYBRID - 5	[Return the results] Return.

The HYBRID_DC algorithm receives data of at least 3 dimensions.

Algorithm HYBRID_DC Given a set **Z** of points in a 3-dimensional or greater space, find the nondominated points that are in the set.

HYBRID_DC - 1	[Call the HYBRID_DC or SC algorithm] Chose the HYBRID_DC or the SC algorithm based on which has a lower value of its estimated run time. If HYBRID_DC, split the data as per the DC algorithm and call recursively.
HYBRID_DC - 2	[Call HYBRID_MARRY to cull remaining dominated points in the inferior set] Call the HYBRID MARRY algorithm on the remaining data.
HYBRID_DC - 3	[Return the results] Return.

The HYBRID_MARRY algorithm receives data of at least 2 dimensions.

Algorithm HYBRID_MARRY

HYBRID_MARRY - 1	[Use linear algorithm if 2D problem] If problem is of dimension 2, use MARRY2D algorithm.
HYBRID_MARRY - 1	[Call the HYBRID_MARRY or MD algorithm] Chose the HYBRID_MARRY or the MD algorithm based on which has a lower value of its estimated run time. If HYBRID_MARRY, split the data as per the MARRY algorithm and call recursively.
HYBRID_MARRY - 3	[Return the results] Return

CHAPTER 8

Test of Algorithm against Satellite Model Data

While the algorithms developed in the thesis have been comprehensively tested using random test data, the test data is by nature of a particular form that is derived from the algorithm to generate it, and so the possibility exists that the algorithm's performance is tied to the structure of the data. A concern is that real world test data that has a different structure than the test data may cause the algorithms to perform differently. Therefore, this chapter exercises the model created in Chapter 2 to generate a single data set of 10,000 sample points in the 8D space. The inputs, listed in Table 13, were randomly sampled via a distribution uniformly spread between their upper and lower bounds. The input and resulting output values were recorded to form the data set. This data set is then decomposed into multiple data sets of varying numbers of dimensions, resulting in a total of 247 distinct data sets available for testing the hybrid algorithm. The preference for each dimension is also noted in the table.

The experiments were run against the entire 8D data set, and also against all possible choices of dimensions with at least 2 dimensions. For the original data set of dimension d there are a total of $2^d - d - 1$ possible choices of dimensions to pick in creating subsets of the problem (neglect the picks that have a single dimension, and also the empty set.) For d=8, there are 247 possible combinations. Of those there are 28 2D combinations, 56 3D combinations, 70 4D combinations, 56 5D combinations, 28 6D combinations, 8 7D combinations, and just one 8D combination (i.e., the whole data set).

Each of these data sets are run through the hybrid algorithm, and compared with the analytical predictions. The parameters are listed in Table 13 along with whether they were to be maximized or minimized.

Table 13: List of dimensions from satellite test problem

Parameter	**In or out**	**Preference**	**Range of Values**
Propellant mass (PropMass)	Input	Minimize	500 – 3000 kg
Payload mass (payload)	Input	Maximize	300 – 1,100 kg
Orbit radius (radius)	Input	Minimize	4000 – 4,400 km
Thrust (thrust)	Input	Minimize	20 – 1,000 Newtons
Delta angle of burn (del)	Calculated	Minimize	-1.18 – 0 Radians
Velocity of trajectory (DV)	Calculated	Maximize	0 – 5,000 km/s
Total mass (mass)	Calculated	Minimize	1,473 – 4,452 kg
Total cost (cost)	Calculated	Minimize	1,400 – 10,600 (dimensionless)

An example of a projection of the 8D space into a 3D space is shown in Figure 99. Noting that DV is to be maximized and thrust and propellant mass are to be minimized, it can be observed that most of the points in this plot would be Pareto points.

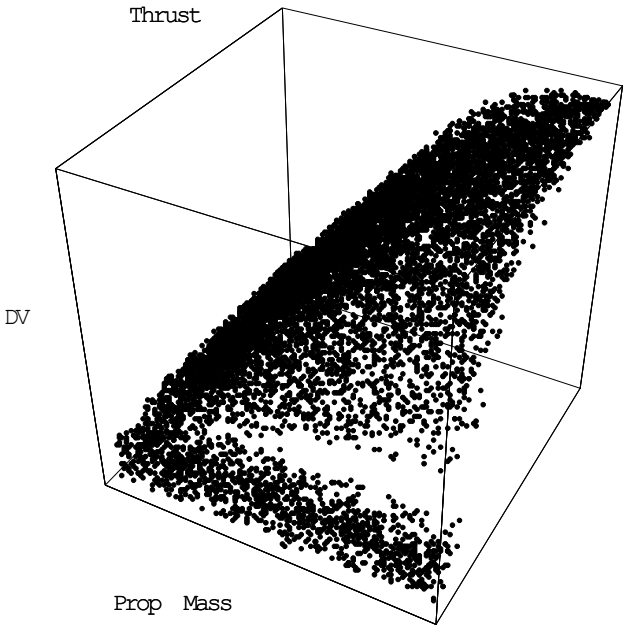

Figure 99: Plot of Propellant mass, delta velocity, and thrust of the engine. Dimensions have been normalized and scaled for proportion

Figure 100 shows the full 8-by-8 scatter plot matrix of the data. The matrix is symmetric about the diagonal, with the diagonal plots just showing the variables plotted againsts themselves. Note the tight correlation between propellant mass and total mass, indicative of the fact that a spacecraft's total mass is typically dominated by the propellant mass. ΔV correlates with propellant mass also, as expected. As cost and total mass are be definition correlated, cost then correlates with propellant mass and ΔV. The scatter matrix does not reflect the trends of design variables such as payload and radius on cost, as their effect is dwarfed by the effect of the mass variables.

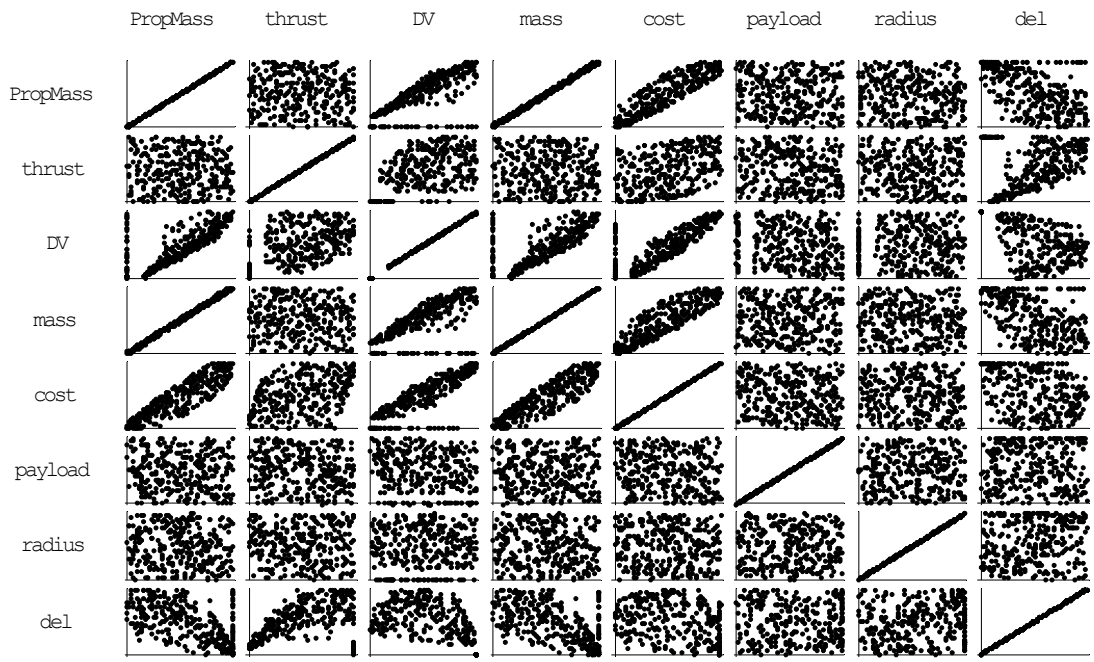

Figure 100: Scatter matrix of the 8D data

By decomposing the data set, a broad variation in dimensions and number of Pareto points is possible. Figure 101 shows the model-derived data in red and the analytical estimates of run time in blue for each dimension from 3 to 8. The hybrid algorithm with a first pass of the SC was used. The figure shows that for the 3D and 4D case the analytical model underestimates the total run time but that the analytical model is a good fit for higher dimensions. The underestimates for the lower dimensions is most probably a reflection of the sensitivity of the run time estimates to the structure of the data, as discussed in Section 6.5.2.

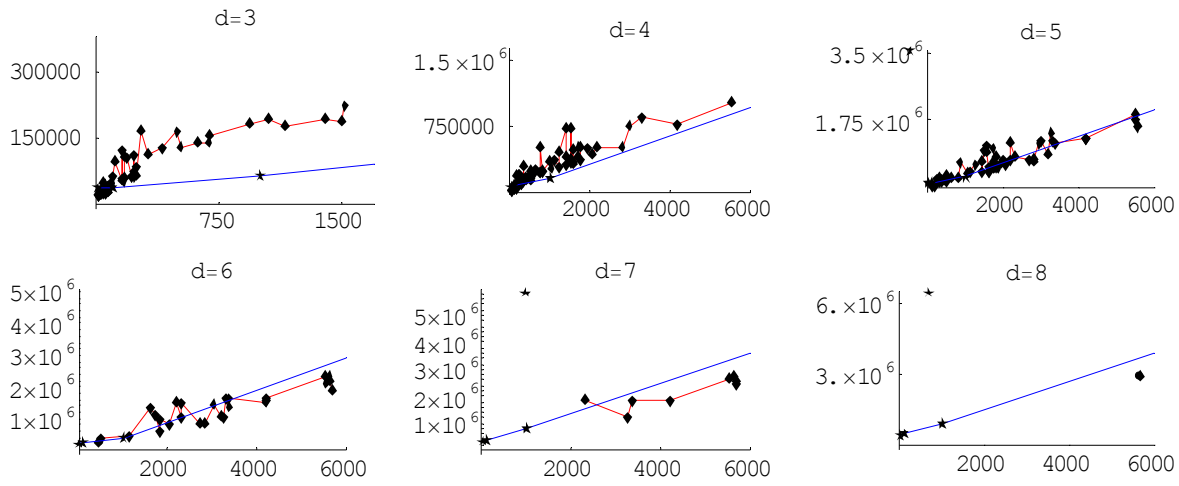

Figure 101: Comparison of data derived from the model (red with diamonds) and analytical estimates of run time (blue with stars) for dimensions ranging from 3 to 8

As there was only one data set for the 8D case, the preference structure was modified to allow for an additional 27 8D data sets. For the data set in Figure 101, the columns of data were multiplied by 1, -1, 1, 1, 1, 1 ,-1 and 1 respectively, in order to reflect whether the attribute was to be minimized or maximized. In order to generate new 8D data sets, all possible permutations of preferences {1, 1, -1, -1, -1, -1, -1, -1} and {1, -1, -1, -1, -1, -1, -1, -1}, which maximizes either two parameters or one parameter, respectively, were applied to the original 8D data set prior to determining the number of Pareto points and the run time. The results are shown in Figure 102.

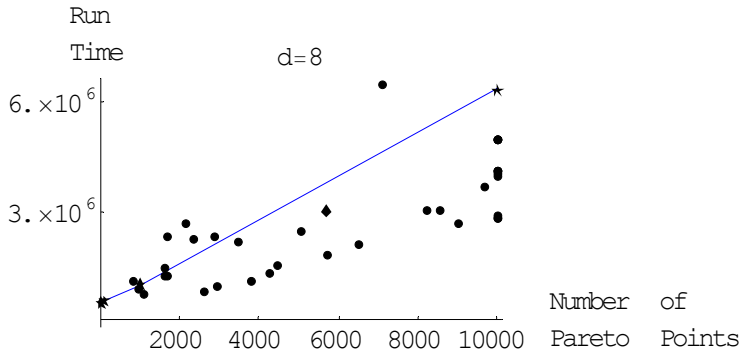

Figure 102: Number of Pareto points versus run time for 8D data with 10,000 points, derived from the satellite model. Experimental results are marked with diamonds; analytical is blue line with stars

Comparing the performance of the algorithm against both the satellite model derived data and the test data shows that the run time estimates derived from the test data are applicable to other data sets. One difference is the dispersion in the run times for the satellite data is much greater than from the test data. Since many of the parameters of the satellite model are directly correlated, e.g., propellant mass, total mass and cost, we are probably seeing the effects that the ordering of the dimensions in the divide & conquer steps has on

total performance, as per Section 6.6. This would lead to shorter than estimated run times. Figure 103 shows the results of experimentally determining the effect of permuting the columns of the 8D data on the run time of the algorithm. In this experiment, 40 different permutations of the columns were selected at random from the 8! possible choices, and the hybrid algorithm was executed. The results show significant dispersion in the run times, quantified by Figure 103 as being greater than a factor of 2, and reinforce the point that column ordering has a significant effect on the efficiency of the algorithm.

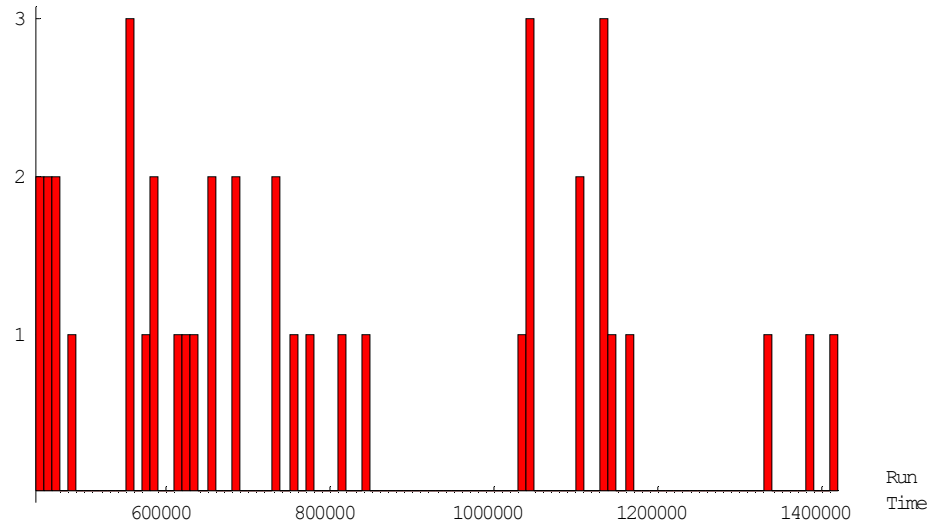

Figure 103: Histogram showing the run times for the 8D spacecraft data, 10,000 points, 712 Pareto points, with the columns permuted in 40 different possible orderings

CHAPTER 9

Conclusions and Future Research

9.1 Conclusions

The main goal in this research has been to develop a hybrid algorithm that would be efficient in terms of run time in identifying Pareto points from multi-dimensional data sets. This task was accomplished, resulting in an algorithm that demonstrates an average of 40% reduction in run time over a pure divide & conquer (DC) algorithm, and that can adapt to the data to operate most efficiently. The hybrid algorithm is analogous to hybrid algorithms for sorting, which for example often employ QuickSort for initial stages, switching to a less efficient but lighter weight algorithm for subproblems below a critical size.

In developing the algorithm, analytical estimates of upper bounds on the expected worst case run times were developed for data sets where all points are Pareto. These upper bounds were shown to be typically 125% of the actual run times of the algorithm against experimental data. The analytical estimates also show that for data sets of very high dimension or small number of points, the run time is bounded by the number of points in the data set only. This result is not obvious from the initial theoretical computational complexity for the DC algorithm of $O(n \log^{d-2} n)$, which contains a dependence on the dimension.

For data sets with all points Pareto, the ordering of the columns in the data was shown to have significant impact on the run time of the DC algorithm, possibly affecting it by orders of magnitude. This is in contrast to the Simple Cull (SC) algorithm, which has a deterministic run time for all points Pareto.

For data sets with few points Pareto, the structure of the remaining dominated points strongly impacts the run time of the DC algorithm. This effect is mitigated in the hybrid algorithm by running a single pass of the SC algorithm first.

While the experimental data used to validate the analytical models of run time resulted in fairly tight groupings of run time versus number of Pareto points, the data set generated from the satellite design model had much more dispersion. This suggests that more aggressive adaptation in the algorithm to take advantage of structure in the data would be beneficial.

Fast, efficient algorithms to identify the Pareto set are enabling technologies to the larger goal of trade space exploration. With these algorithms one can consider larger data sets of higher dimensionality, thereby

improving the a decision maker's understanding of impact of requirements, constraints, and preferences on the trade space, ultimately resulting in a better product.

9.2 Limitations

An assumption underlying much of the development of the SC and DC algorithms in this thesis is that each element in the initial table **T** was unique in its column, which may not be true if input or output values of a model are restricted to take one of a finite set of values. The transformation to LLH form, the ranking of points in the SC algorithm, and the fundamental operation of the DC algorithms were all predicated on this assumption.

Another assumption made is that all data is transformed to LLH form. This is clearly justified for the use of the algorithm to support visualization, as one is operating against a fixed data set throughout, so the penalty for transforming to the LLH form is offset by the savings in future sorting operations and in the Pareto algorithm. For problem domains where each data set is only operated on once, the gains may not necessarily offset the up-front computation. While the DC algorithm and the hybrid algorithm without the first pass do not depend on the LLH form, identifying the point to use for a single first pass in the hybrid does. Also, the analytical estimates of run times would no longer hold true if each divide step of the DC algorithm required a comparison sort.

9.3 Future Research

There are a number of future research topics stemming from this thesis, which would either complete the work done here or would represent new thrusts that are motivated by this work. They are

1. Adaptive ordering of dimensions in the hybrid algorithm
2. Allowing for duplicate elements in the columns of tables
3. Defining multiple levels of dominance for visualization

Each of these is elaborated upon in the following sections.

9.3.1 Adaptive Ordering Of Dimensions in the Hybrid Algorithm

As introduced in Section 6.6 and seen in Chapter 8, the ordering of columns in the table can change the run time of the DC algorithm by orders of magnitude. Section 6.6 introduced a method to mitigate this effect, by randomly choosing the next column/dimension on which to operate. This random approach is reminiscent of a mini-max approach to decision making in that it minimizes the worst expected run time that the algorithm could experience, rather than minimize the run time. The random selection of columns not only minimizes the worst expected run time but also maximizes the minimum expected run time. An ideal solution would be to take advantage of the disparity in run times between different column orderings, and order them so as to minimize the run time. This section sketches out an approach to doing so.

Recall the previously introduced Figure 51, repeated here for reference as Figure 104. In the figure, the lower left corner is the preferred direction. The figure represents a projection of a higher dimensional space onto a 2D space. These two dimensions will be the next two chosen in the MARRY algorithm for cutting.

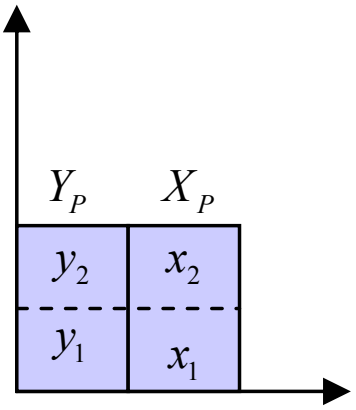

Figure 104: Dividing of space in DC algorithm

If the regions y_1 or x_2 are unoccupied as in the projection of points onto the plane as shown in Figure 104 then the algorithm will terminate early, greatly reducing the number of comparisons required. Figure 105 shows two different projections of a 5D data set onto a 2D plane. The first plot clearly has the X and Y points inversely correlated, and so would have regions x_2 and y_1 relatively empty. One would expect the Marry algorithm to perform well if these two dimensions are chosen as the first two to operate on. The second projection does not share this property, and the performance of the Marry algorithm, would suffer accordingly, even though both cases are with the same data. The key research question in support of adaptively selecting the ordering of the dimensions is how to most accurately analyze the best choice of dimensions to order on which to order, while minimizing extra computational time?

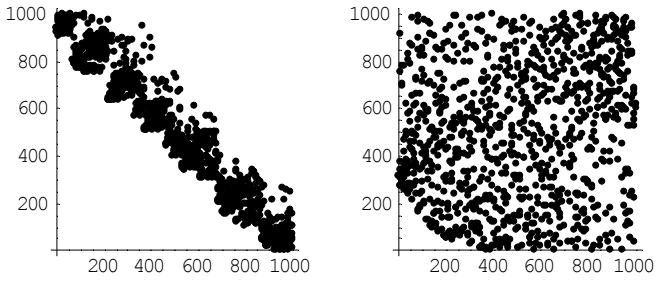

Figure 105: Two different 2D projections of a hypothetical 5D data set with some dimensions correlated and other not

Assuming the table **Z** is of size N and dimension d, then there will be $(d^2 + d)/2$ possible choices of 2D projections from which to choose. One possible approach that suggests itself is to compute the correlation coefficients for each of the possible projections, and then choose the one that most nearly approaches -1. A

negative to this approach is that it would require $N(d^2 + d)/2$ multiplications to compute all of the coefficients. This could be mitigated by only taking a sample of the points, assuming that if randomly chosen then the correlation coefficients calculated from the samples would be sufficient on which to make a decision. The absolute values of the coefficients are not as important as their relative values. Another issue is that the correlation coefficient is skewed by points that lie in the upper left or lower right quadrant, when the key to algorithm efficiency is strictly based on which quadrants are occupied, not where they are occupied.

Another approach would be to set up a parallel data structure where, for each point, its position would be either 1 or -1, reflecting whether it was in the upper or lower half of values within its column. Then one can compute the correlation coefficient. This approach would not skew the coefficient based on point position, and it also would have the advantage in that the multiplication step could be replaced by a more efficient comparison of the two values. Again, a statistically representative sample of the points could be used to reduce the time to compute the coefficients.

Analysis and experiments would be needed to fix the number of samples of the data set to best estimate the points. An interesting issue is that, since the goal is to estimate the relative ordering of the coefficients rather than there absolute values, the sampling itself may need to be adaptive. For cases as in Figure 105 it would require relatively few points to establish that the first ordering clearly is better than the second, while for projections that are similar, many more samples might be needed to establish the best ordering, while the gains expected from a proper ordering would diminish. If no one coefficient is clearly dominating or if perhaps two are dominating but similar and three are clearly deficient, the algorithm for choosing dimensions would stop and randomly pick one of the two remaining projections.

9.3.2 Allowing For Duplicate Elements in the Columns of Tables

An assumption underlying much of the development of the SC and DC algorithms in this thesis is that in the initial table **T**, each element was unique in its column. The transformation to LLH form, the ranking of points in the SC algorithm, and the fundamental operation of the DC algorithms were all predicated on this assumption. This section discusses issues in relaxing the assumption, and sketches out methods for handling duplicate elements in the SC and DC algorithms.

To provide an example for discussion, two tables are given. **T** (see Table 14) is a table of designs that has duplicate elements in the cost column, while **Z** (see Table 15) is the data transformed to LLH form, with the transform arbitrarily assigning an ordering within the cost column. To reflect the preference of more speed and range and less mass, diameter, length and cost, the values for speed and range are given in negative numbers.

Table 14: Design instances with duplicate elements in the cost column

T	Speed ↑	Range ↑	Mass ↓	Diameter ↓	Length ↓	Cost ↓
$\mathbf{t}_1$	-25	-13000	2000	13.2	230	**1,000,000**
$\mathbf{t}_2$	-23.2	-12980	2050	15.0	231	**1,000,000**
$\mathbf{t}_3$	-22.4	-12700	2543	13.1	229	1,200,000
$\mathbf{t}_4$	-28.25	-14075	2100	14.7	215	950,000

Note that design $\mathbf{t}_1$ dominates $\mathbf{t}_2$. In the conversion to LLH form, however, an arbitrary ranking is placed between the duplicates, and now $\mathbf{z}_1$ does not dominate $\mathbf{z}_2$. This simple example shows that one cannot simply transform the table to LLH form and have the proper dominance relationships preserved.

Table 15: Transformation to LLH form, with arbitrary ranking of duplicate items in cost column

Z						
$\mathbf{z}_1$	2	2	1	2	3	**3**
$\mathbf{z}_2$	3	3	2	4	4	**2**
$\mathbf{z}_3$	4	4	4	1	2	4
$\mathbf{z}_4$	1	1	3	3	1	1

The key to modifying the process is to recognize that the data converted to LLH form is used for two different functions in the SC and DC algorithms: (1) it is used to order the points, and (2) it is used to compare the points. The key to allowing duplicate elements is to recognize the two different functions of the data and adjust accordingly. The recommended approach is to still transform the data to LLH form and use the LLH data for ordering the points, and to use either the original data or additionally modified LLH data with duplicate elements to compare the points. As the only way points are removed in either the SC or DC algorithms is through a direct comparison with another point, this will ensure that no nondominated point is inadvertently removed. The issue is to determine how the conversion to LLH form and how the SC and DC algorithms need to be modified to ensure that all dominated points are removed. There are three candidate approaches, enumerated as follows:

1. Use LLH form to order points, but compare points using original data

2. Use a modified LLH form that allows for duplicate elements, to both order and compare points
3. Use LLH form to order points, and use a modified LLH form with duplicate elements to compare points

The advantage to the first approach is that it requires minimal modification to the existing algorithm and no additional storage. A disadvantage is that comparing real numbers is more time consuming than comparing integers.

The advantage to the second approach is that no additional storage is required and that comparisons will be between integers. A disadvantage is that the DC algorithm may not perform optimally if, when dividing the data sets, the divide is imbalanced.

The third approach has the advantage of simplicity of ordering and comparing, but it requires either additional storage for the modified LLH data or additional bookkeeping to track common elements. Both increase the complexity of the algorithm.

While it appears that all three approaches applied to the SC and DC algorithms should result in algorithms that are correct, it remains to be rigorously proven.

9.3.3 Defining Multiple Levels of Dominance for Visualization

Since the current prime use of the algorithms developed here is to support multi-dimension visualization, issues that arise in the visualization reflect back on the algorithms. One of the issues is on identifying more points than those strictly considered nondominated, in order to "thicken" the boundary region of nondominated points or to identify points that are "almost nondominated". Figure 106 shows the Pareto frontier (red points) for an 80 point, 2D data set. For this data set, down and left are preferred. One can see in looking at the figure that there are a number of other points that are very close to the frontier also, although the algorithm does not identify them as such.

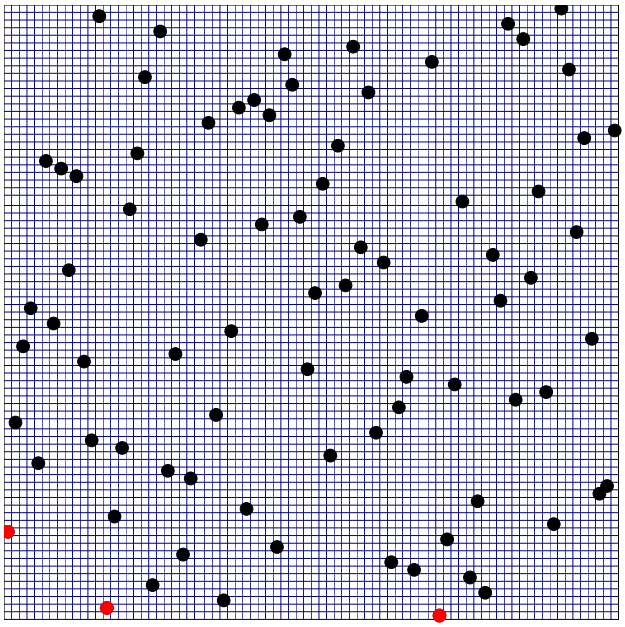

Figure 106: Pareto frontier for 80 points

An alternate approach, for example, would be to break points up into “tiers” of Pareto points (i.e., multiple Pareto fronts), where for example Figure 107 shows a plot with 70 points, down and left preferred, with three tiers of Pareto points. The red points are Pareto in the entire set. The blue points are Pareto in the set remaining if the red points are removed. The green points are Pareto if the red and blue points are first removed. Another candidate would be to show in the second tier points that only have one dominating point, and show in the third tier points that have two dominating points, etc. One can easily come up with additional candidate methods for showing both the Pareto points and points that are nearly Pareto.

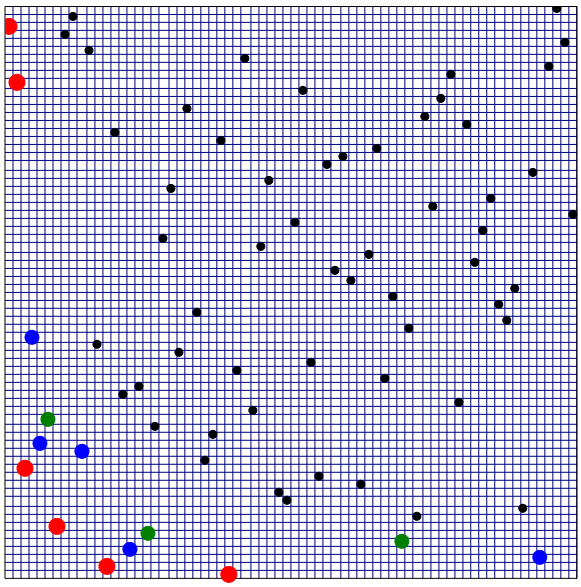

Figure 107: 70 points with three tiers of Pareto points

This problem is exactly analogous to the current problem in evolutionary computing, where researchers are attempting to develop efficient algorithms to identify multiple layers of Pareto frontiers, also known as Pareto sorting (Murata and Ishibuchi 1995; Zitzler and Thiele 1999; Knowles and Corne 2000; Zitzler et al. 2000; Deb 2001; Deb et al. 2002; Jensen 2003). Whatever method is developed to mark additional points, it is likely that the SC and DC algorithms can be improved upon to best identify them. The methods to assign the tiers of Pareto points, and how to modify the SC and DC algorithms to best identify these points, are open research questions.

References

Abraham, L. H. (1965). *Spacecraft Systems,* Washington: Scientific and Technical Information Division National Aeronautics and Space Administration; [for sale by the Superintendent of Documents U. S. Govt. Print. Off.], 1965

Alexandrov, N. M. and S. Kodiyalam (1998). "Initial Results of an MDO Method Evaluation Study." *7th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Vol. 2, St. Louis, MO: AIAA, 2-4 September, pp. 1315-1327

Alexandrov, N. M. and R. Lewis (2000). "Analytical and computational aspects of collaborative optimization," NASA NASA/TM-2000-210104, Hampton, VA, April 2000

Alexandrov, N. M. and R. M. Lewis (1999). "Comparative Properties of Collaborative Optimization and Other Approaches to MDO," NASA NASA/CR-1999-209354, Hampton, VA, July 1999

Allen, B. (2001). "On the Aggregation of Preferences in Engineering Design." *ASME Design Engineering Technical Conferences - Design Automation Conference*, ed. by A. Diaz, Pittsburgh, PA: ASME, September 9-12

Balling, R. (1999). "Design by Shopping: A New Paradigm?" *Proceedings of the Third World Congress of Structural and Multidisciplinary Optimization (WCSMO-3)*, ed. by C. L. Bloebaum, K. E. Lewis and R. W. Mayne, Vol. 1, Buffalo, NY: University at Buffalo, May 17-21, pp. 295-297

Barton, R. R., M. Meckesheimer, et al. (2000). "Experimental Design Issues for Simultaneous Fitting of Forward and Inverse Metamodels." *ASME 2000 Design Engineering Technical Conference and Computers and Information in Engineering Conference*

Becker, R. and W. Cleveland (1987). "Brushing Scatterplots." *Technometrics* 29: 127-142

Bentley, J. (1980). "Multidimensional Divide-and-Conquer." *Communications of the ACM* 23, no. 4 (April 1980): 214-229

Bentley, J., K. Clarkson, et al. (1990). "Fast linear expected-time algorithms for computing maxima and convex hulls." *Annual ACM-SIAM Symposium on Discrete Algorithm*, Jan 1990, pp. 509-517

Box, G. E. P. and N. R. Draper (1959). "A Basis for the Selection of a Response Surface Design." *Journal of the American Statistical Association* 54: 622-654

Brown, C. D. (1996). *Spacecraft Propulsion,* Washington, DC: AIAA, 1996

Brown, C. D. (1998). *Spacecraft Mission Design,* 2nd ed., Washington, DC: AIAA, 1998

Cheng, F. Y. and D. Li (1997). "Multiobjective Optimization Design with Pareto Genetic Algorithm." *Journal of Structural Engineering* 123, no. 9: 1252-1261

Conway, J. H. and R. K. Guy (1996). *The Book of Numbers,* New York, NY: Copernicus, 1996

Cormen, T. H., C. E. Leiserson, et al. (1994). *Introduction to Algorithms,* Cambridge, Massachusetts: The MIT Press, 1994

Cramer, E. J., J. E. Dennis, Jr., et al. (1994). "Problem Formulation for Multidisciplinary Optimization." *SIAM Journal of Optimization* 4, no. 4: 754-776

Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms,* New York: Wiley, 2001

Deb, K., S. Pratab, et al. (2002). "A Fast and Elitist Multiobjective Genetic Algorithm: NGSA-II." *IEEE Transactions on Evolutionary Computing* 6 (Apr 2002): 182-197

Ehrgott, M. and X. Gandibleux (2000). "An Annotated Bibliography of Multiobjective Combinatorial Optimization," Universitat Kaiserslautern NR 62/2000, Kaiserslautern, Germany, 13 April 2000

Fleeter, R. (2000). *The Logic of Microspace* ed. J. R. Wertz, Space Technology Library ; v 9, El Segundo, Calif., Microcosm Press, 2000

Fortescue, P. W. and J. P. W. Stark (1995). *Spacecraft Systems Engineering,* 2nd ed., Chichester ; New York: Wiley, 1995

Gavanelli, M. (2001). "Partially Ordered Constraint Optimization Problems." *Principles and Practice of Constraint Programming, 7th International Conference-CP2001*, ed. by T. Walsh, Vol. 2239, Paphos, Cyprus: Springer Verlag

Gavanelli, M. (2002). "An Implementation of Pareto optimiality in CLP(FD)." *CP-AI-OR-International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, ed. by N. Jussein and F. Laburthe, Le Croisic, France: Ecole des Mines de Nante, pp. 49-64

Goebel, M. and L. Gruenwald (1999). "A Survey of Data Mining and Knowledge Discovery Software Tools." *SIGKDD Explorations* 1, no. 1 (June 1999): 21-33

Golub, G. H. and C. F. Van Loan (1996). *Matrix Computations,* 3rd ed., Baltimore: Johns Hopkins University Press, 1996

Hazelrigg, G. A. (1996). *Systems Engineering : an Approach to Information-Based Design,* Upper Saddle River, NJ: Prentice Hall, 1996

Jeffrey, A. (2000). *Handbook of Mathematical Formulas and Integrals,* 2nd ed., New York: Academic Press, 2000

Jensen, M. T. (2003). "Reducing the Run Time Complexity of Multiobjective EAs: The NGSA-II and Other Algorithms." *IEEE Transactions on Evolutionary Computing* 7, no. 5 (October 2003): 503-515

Jones, D. R., M. Schonlau, et al. (1998). "Efficient Global Optimization of Expensive Black-Box Functions." *Journal of Global Optimization* 13: 455-492

Keim, D. A. (2002). "Information Visualization and Visual Data Mining." *IEEE Transactions on Visualization and Computer Graphics* 8, no. 1: 1-8

Knowles, J. D. and D. W. Corne (2000). "Approximating the Nondominated Front Using the Pareto Archived Evolutionary Strategy." *Evolutionary Computing* 8 (2000): 149-172

Knuth, D. E. (1992). *Literate Programming,* Stanford, CA: Center for the Study of Language and Information, 1992

Knuth, D. E. (1997). *The Art of Computer Programming,* 3rd ed., Reading, Mass.: Addison-Wesley, 1997

Kung, H. T., F. Luccio, et al. (1975). "On Finding the Maxima of a Set of Vectors." *Journal for the Association of Computing Machinery* 22, no. 4: 469-476

Messac, A., J. G. Sundararaj, et al. (1999). "The Ability of Objective Functions to Generate Non-Convex Pareto Frontiers." *40th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Vol. 1, St. Louis, MO: AIAA, April 12-15, pp. 78-87

Murata, T. and H. Ishibuchi (1995). "MOGA: multi-Objective Genetic Algorithms." *IEEE Conference on Evolutionary Computation*: IEEE, 1 Nov 29-Dec 1 1995, pp. 289-294

Nagel, H., E. Granum, et al. (2001). "Methods for Visual Mining of Data in Virtual Reality." *International Workshop on Visual Data Mining at ECML/PKDD 2001*, San Francisco: Simeon Simoff, 7 Sep 2001, pp. 13-27

O'Rourke, J. (1993). *Computational Geometry in C,* Cambridge: Cambridge University Press, 1993

Rogers, J. and C. L. Bloebaum (1994). "Ordering Design Tasks Based on Coupling Strengths," NASA TM-109137, Langley, VA, Sept. 1994

Rogers, J., C. M. McCulley, et al. (1999). "Optimizing the Process Flow of Complex Design Projects." *Design Optimization, International Journal for Product and Process Improvement* 1, no. 3 (1999): 281-292

Rosenthal, R. (1985). "Principles of Multiobjective Optimization." *Decision Sciences* 16: 133-152

Scott, M. J. and E. K. Antonsson (1999). "Arrow's Theorem and Engineering Design Decision Making." *Research in Engineering Design* 11, no. 4: 218-228

Shah, J. J. and P. K. Wright (2000). "Developing Theoretical Foundations for DfM." *ASME Design Engineering Technical Conferences - Design for Manufacturing*, Baltimore, MD: ASME, September 10-13

Simpson, T. W., A. J. Booker, et al. (2003). "Approximation Methods in Multidisciplinary Analysis and Optimization: A Panel Discussion." *Structural and Multidisciplinary Optimization* (In Press)

Simpson, T. W., J. Peplinski, et al. (2001). "Metamodels for Computer-Based Engineering Design: Survey and Recommendations." *Engineering with Computers* 17, no. 2 (2001): 129-150

Sobieszczanski-Sobieski, J. and R. T. Haftka (1997). "Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments." *Structural Optimization* 14: 1-23

Stump, G., M. Yukish, et al. (2002). "Multidimensional Visualization and Its Application to a Design by Shopping Paradigm." *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Washington, DC: AIAA, 4-6 Sept.

Stump, G., M. Yukish, et al. (2004). "Trade Space Exploration of Satellite Datasets Using a Design by Shopping Paradigm." *IEEE Aerospace Conference*, New York: IEEE, 7-13 March 2004

Stump, G. M., M. Yukish, et al. (2002). "Multidimensional Visualization and Its Application to a Design by Shopping Paradigm." *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA: AIAA, September 4-6, 2002

Sutton, G. P. and O. Biblarz (2001). *Rocket Propulsion Elements,* 7th ed., New York: John Wiley & Sons, Inc., 2001

Tang, X. and S. Krishnamurty (2000). "On Decision Model Development in Engineering Design." *Journal of Engineering Valuation & Cost Analysis, Special Edition on Decision-Based Design* 3(2/3): 131-150

Tappeta, R. V. and J. E. Renaud (1999). "Interactive Multiobjective Optimization Design Strategy for Decision Based Design." *Advances in Design Automation*, Las Vegas, NV: ASME, September 12-15

Tappeta, R. V. and J. E. Renaud (1999). "Interactive Multiobjective Optimization Procedure." *40th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, Vol. 1, Washington, DC: AIAA, April 12-15, pp. 27-41

Thurston, D. L. (2001). "Real and Misconceived Limitations to Decision Based Design with Utility Analysis." *ASME Journal of Mechanical Design* 123, no. 2: 176-182

Wertz, J. R. and W. J. Larsen (1999). *Space Mission Analysis and Design*, 3rd ed., ed. J. R. Wertz, Space Technology Series, El Segundo, CA, Microcosm Press, Kluwer Academic Publishers, 1999

Yukish, M. and E. Harris (2002). "Formulations To Support the Conceptual Design of Satellites." *World Space Congress 2002*, Houston, TX: AIAA, 18 Oct 2002

Zitzler, E., K. Deb, et al. (2000). "Comparison of Multiobjective Evolutionary Algorithms: Emprical Results." *Evolutionary Computing* 8, no. 2 (2000): 173-195

Zitzler, E. and L. Thiele (1999). "Multiobjective Evolutionary Algorithms: a Comparative Case Study and the Strength Pareto Approach." *IEEE Transactions on Evolutionary Computing* 3 (November 1999): 257-271

APPENDIX A

Fitting Parameters for Estimators

Table 16: Coefficients for estimating functions

Dimension	Pbf[ ]	Pff[ ]	Pdf[ ]	Mbf[ ]	Mff[ ]	Mdf[ ]
3	1.05913	1.19093	1.03041	1.02115	0.634506	0.918568
4	0.872525	0.63957	0.798846	0.946639	0.603141	0.853297
5	0.70962	0.544582	0.698684	0.805111	0.57684	0.793526
6	0.657503	0.525402	0.687411	0.71493	0.544081	0.75109
7	0.659375	0.518626	0.696837	0.682871	0.499059	0.698144
8	0.681469	0.508409	0.690546	0.697301	0.508089	0.708953
9	0.710089	0.534579	0.7184	0.711502	0.506027	0.70492
10	0.736185	0.547895	0.730274	0.708196	0.50814	0.694486
11	0.75745	0.562509	0.744163	0.755914	0.546505	0.745668
12	0.772541	0.57927	0.758007	0.757405	0.550855	0.747881
13	0.774487	0.575436	0.76189	0.794202	0.57917	0.78029
14	0.795029	0.584329	0.771653	0.775273	0.560446	0.757071
15	0.750314	0.557616	0.734427	0.756452	0.530607	0.722296
16	0.81629	0.615247	0.797987	0.824969	0.596409	0.789224
17	0.819606	0.608744	0.79109	0.842754	0.605894	0.809232
18	0.808472	0.589733	0.778427	0.867312	0.619583	0.827767
19	0.833996	0.62287	0.805897	0.863022	0.63231	0.831596
20	0.822385	0.610228	0.797431	0.868603	0.62227	0.830964
Mean	**0.779804**	**0.606443**	**0.766243**	**0.799645**	**0.567996**	**0.775832**

APPENDIX B

Solving Recursions

This section shows how to solve the recursions that appear throughout the thesis. The general form of the recursion is $f(n) = 2f(n/2) + g(n)$. This can be converted to a summation as follows:

$$\begin{aligned}
f(n) &= 2f(n/2) + g(n) \\
&= 4f(n/4) + 2g(n/2) + g(n) \\
&= 2^3 f(n/2^3) + 2^2 g(n/2^2) + 2^1 g(n/2^1) + 2^0 g(n/2^0) \\
&= \frac{n}{2^{\log n-3}} g(2^{\log n-3}) + \frac{n}{2^{\log n-2}} g(2^{\log n-2}) + \frac{n}{2^{\log n-1}} g(2^{\log n-1}) + \frac{n}{2^{\log n}} g(2^{\log n}) \\
&\vdots \\
&= n\left(\frac{g(2^1)}{2^1} + \frac{g(2^2)}{2^2} + \cdots + \frac{g(2^{\log n-1})}{2^{\log n-1}} + \frac{g(2^{\log n})}{2^{\log n}} \right) \\
&= n\sum_{i=1}^{\log n} \frac{g(2^i)}{2^i}
\end{aligned}$$

A commonly occurring form for $g(n)$ is $g(n) = n\log^d n$ for some $d \in \{0,1,2,\ldots\}$. This can be solved for in closed form.

$$n\sum_{i=1}^{\log n} \frac{(2^i)\log^d(2^i)}{2^i} = n\sum_{i=1}^{\log n} i^d$$

Using the result for the generalized summing of powers of integers (Jeffrey 2000), one can write as

$$n\sum_{i=1}^{\log n} i^d = n\frac{\log^{q+1} n}{q+1} + n\frac{\log^q n}{2} + \frac{1}{2}\binom{q}{1} B_2 n\log^{q-1} n + \frac{1}{4}\binom{q}{3} B_4 n\log^{q-3} n + \frac{1}{6}\binom{q}{5} B_6 n\log^{q-5} n + \cdots$$

where B_i are the Bernoulli numbers, and the series terminates at either $\log n$ or $\log^2 n$.

APPENDIX C

Expressions for Comparisons, Function Calls, and Data Inputs

The following figures show the expressions for the analytical estimates for number of comparisons pbf[ ] and mbf[ ], the analytical estimates for number of function calls pff[ ] and mff[ ], and the analytical estimates for amount of data passed ptf[ ] and mtf[ ].

The Mathematica code for generating the analytical estimates of the run time for the Marry algorithm are in Figure 108. The code generates estimates up to a dimension of 20, but can be modified to generate estimates for any range. The code is exercised for d=5.

```
In[9]:= mbf[n_, 2] := n - 1;
        mbf[2, d_] := 1;
        Table[mbf[n_, d] := Evaluate[ n (1/2 + Sum[(mbf[2^i/2, d - 1])/2^i, {i, 2, Log[2, n]}])], {d, 3, 20}];
        mbf[n, 5]
```

$$\text{Out[12]= } n\left(\frac{1}{2}+\frac{48\,\text{Log}[2]^3-54\,n\,\text{Log}[2]^3+35\,n\,\text{Log}[2]^2\,\text{Log}[n]-6\,n\,\text{Log}[2]\,\text{Log}[n]^2+n\,\text{Log}[n]^3}{48\,n\,\text{Log}[2]^3}\right)$$

Figure 108: Mathematica code for generating analytical estimate of number of comparisons for the Marry algorithm

Note in the result for mbf[n,5] in Figure 108 the expression Log[2] appearing in the numerator and the denominator. This is because Mathematica converts all Base 2 logarithms such as $\log_2 n$ into the equivalent form $\ln n / \ln 2$. In Mathematica Log[n] means the natural logarithm of n. To maintain consistency within this thesis and simplify presentation of the results below, Mathematica's output has been modified so that log[n] (note the lower case 'l' in the function) indicates the logarithm Base 2 of n.

Figure 109 shows the estimates for the Marry algorithm for dimensions 2 through 8. All logarithms are in Base 2.

Dimension	mbf[n,d]
2	$-1+n$
3	$1-\frac{n}{2}+\frac{1}{2}n\log[n]$
4	$-1+\frac{5n}{4}-\frac{3}{8}n\log[n]+\frac{1}{8}n\log[n]^2$
5	$1-\frac{5n}{8}+\frac{35}{48}n\log[n]-\frac{1}{8}n\log[n]^2+\frac{1}{48}n\log[n]^3$
6	$-1+\frac{21n}{16}-\frac{97}{192}n\log[n]+\frac{83}{384}n\log[n]^2-\frac{5}{192}n\log[n]^3+\frac{1}{384}n\log[n]^4$
7	$1-\frac{21n}{32}+\frac{1537n\log[n]}{1920}-\frac{47}{256}n\log[n]^2+\frac{11}{256}n\log[n]^3-\frac{1}{256}n\log[n]^4+\frac{n\log[n]^5}{3840}$
8	$-1+\frac{85n}{64}-\frac{2087n\log[n]}{3840}+\frac{181}{720}n\log[n]^2-\frac{43n\log[n]^3}{1024}+\frac{59n\log[n]^4}{9216}-\frac{7n\log[n]^5}{15360}+\frac{n\log[n]^6}{46080}$

Figure 109: Expresssions for mbf[ ]

Figure 110 shows the code for generating the estimates of number of comparisons for the DC algorithm.

Figure 111 shows pbf function for dimensions 2 through 8.

```
In[51]:= pbf[n_, 2] := n - 1;
         pbf[2, d_] := 1;
         Table[pbf[n_, d] := Evaluate[ n (1/2 + Sum[(mbf[2^i, d - 1])/2^i, {i, 2, Log[2, n]}])], {d, 3, 20}];
```

Figure 110: Mathematica code for generating analytical estimate of number of comparisons for the DC algorithm

Dimension	pbf[n,d]
2	$-1+n$
3	$1-n+n\log[n]$
4	$-1+n-\frac{1}{4}n\log[n]+\frac{1}{4}n\log[n]^2$
5	$1-n+\frac{13}{12}n\log[n]-\frac{1}{8}n\log[n]^2+\frac{1}{24}n\log[n]^3$
6	$-1+n-\frac{9}{32}n\log[n]+\frac{59}{192}n\log[n]^2-\frac{1}{32}n\log[n]^3+\frac{1}{192}n\log[n]^4$
7	$1-n+\frac{263}{240}n\log[n]-\frac{29}{192}n\log[n]^2+\frac{23}{384}n\log[n]^3-\frac{1}{192}n\log[n]^4+\frac{n\log[n]^5}{1920}$
8	$-1+n-\frac{55}{192}n\log[n]+\frac{3677n\log[n]^2}{11520}-\frac{21}{512}n\log[n]^3+\frac{41n\log[n]^4}{4608}-\frac{n\log[n]^5}{1536}+\frac{n\log[n]^6}{23040}$

Figure 111: Expressions for pbf[ ]

Figure 112 shows the code for generating the estimates of number of function calls for the Marry algorithm. Figure 113 shows mff function for dimensions 2 through 8.

```
In[48]:= mff[n_, 2] := 1;
         mff[2, d_] := 1;
         Table[mff[n_, d] := Evaluate[ n/2 + n (Sum[(mff[2^i/2, d - 1] + 1)/2^i, {i, 2, Log[2, n]}])], {d, 3, 20}];
```

Figure 112: Mathematica code for generating analytical estimate of number of function calls for the Marry algorithm

Dimension	mff[n,d]
2	1
3	$-2+\frac{3n}{2}$
4	$1-\frac{3n}{4}+\frac{3}{4}n\log[n]$
5	$-2+\frac{15n}{8}-\frac{9}{16}n\log[n]+\frac{3}{16}n\log[n]^2$
6	$1-\frac{15n}{16}+\frac{35}{32}n\log[n]-\frac{3}{16}n\log[n]^2+\frac{1}{32}n\log[n]^3$
7	$-2+\frac{63n}{32}-\frac{97}{128}n\log[n]+\frac{83}{256}n\log[n]^2-\frac{5}{128}n\log[n]^3+\frac{1}{256}n\log[n]^4$
8	$1-\frac{63n}{64}+\frac{1537n\log[n]}{1280}-\frac{141}{512}n\log[n]^2+\frac{33}{512}n\log[n]^3-\frac{3}{512}n\log[n]^4+\frac{n\log[n]^5}{2560}$

Figure 113: Expressions for mff[ ]

Figure 114 shows the code for generating the estimates of number of function calls for the DC algorithm. Figure 115 shows pff function for dimensions 2 through 8.

```
In[45]:= pff[n_, 2] := 1;
         pff[2, d_] := 1;
         Table[pff[n_, d] := Evaluate[ n/2 + n (Sum[(mff[2^i, d - 1] + 1)/2^i, {i, 2, Log[2, n]}])], {d, 3, 20}];
```

Figure 114: Mathematica code for generating analytical estimate of number of function calls for the DC algorithm

Dimension	pff[n,d]
2	1
3	$-2+\frac{3n}{2}$
4	$1-\frac{3n}{2}+\frac{3}{2}n\log[n]$
5	$-2+\frac{3n}{2}-\frac{3}{8}n\log[n]+\frac{3}{8}n\log[n]^2$
6	$1-\frac{3n}{2}+\frac{13}{8}n\log[n]-\frac{3}{16}n\log[n]^2+\frac{1}{16}n\log[n]^3$
7	$-2+\frac{3n}{2}-\frac{27}{64}n\log[n]+\frac{59}{128}n\log[n]^2-\frac{3}{64}n\log[n]^3+\frac{1}{128}n\log[n]^4$
8	$1-\frac{3n}{2}+\frac{263}{160}n\log[n]-\frac{29}{128}n\log[n]^2+\frac{23}{256}n\log[n]^3-\frac{1}{128}n\log[n]^4+\frac{n\log[n]^5}{1280}$

Figure 115: Expressions for pff[ ]

Figure 116 shows the code for generating the estimates of the amount of data passed in the Marry algorithm. Figure 117 shows pff function for dimensions 2 through 8.

```
In[42]:= mtf[n_, 2] := n;
         mtf[2, d_] := 2;
         Table[mtf[n_, d] := Evaluate[ n (Log[2, n] + Sum_{i=2}^{Log[2,n]} (mtf[2^i/2, d - 1] / 2^i))], {d, 3, 20}];
```

Figure 116: Mathematica code for generating analytical estimates of the amount of data passed in the Marry algorithm

Dimension	mtf[n,d]
2	n
3	$-\frac{n}{2} + \frac{3}{2} n \log[n]$
4	$\frac{n}{4} + \frac{3}{8} n \log[n] + \frac{3}{8} n \log[n]^2$
5	$-\frac{n}{8} + \frac{17}{16} n \log[n] + \frac{1}{16} n \log[n]^3$
6	$\frac{n}{16} + \frac{43}{64} n \log[n] + \frac{35}{128} n \log[n]^2 - \frac{1}{64} n \log[n]^3 + \frac{1}{128} n \log[n]^4$
7	$-\frac{n}{32} + \frac{567}{640} n \log[n] + \frac{25}{256} n \log[n]^2 + \frac{13}{256} n \log[n]^3 - \frac{1}{256} n \log[n]^4 + \frac{n \log[n]^5}{1280}$
8	$\frac{n}{64} + \frac{987 n \log[n]}{1280} + \frac{781 n \log[n]^2}{3840} + \frac{3 n \log[n]^3}{1024} + \frac{23 n \log[n]^4}{3072} - \frac{3 n \log[n]^5}{5120} + \frac{n \log[n]^6}{15360}$

Figure 117: Expressions for mtf[ ]

Figure 118 shows the code for generating the estimates of the amount of date passed in the DC algorithm. Figure 119 shows ptf function for dimensions 2 through 8.

```
In[36]:= ptf[n_, 2] := n;
         ptf[2, d_] := 2;
         Table[ptf[n_, d] := Evaluate[ n (Log[2, n] + Sum_{i=2}^{Log[2,n]} (mtf[2^i, d - 1] / 2^i))], {d, 3, 20}];
```

Figure 118: Mathematica code for generating analytical estimates of the amount of data passed in the DC algorithm

Dimension	ptf[n,d]
2	n
3	$-n + 2 n \log[n]$
4	$-n + \frac{5}{4} n \log[n] + \frac{3}{4} n \log[n]^2$
5	$-n + \frac{3}{2} n \log[n] + \frac{3}{8} n \log[n]^2 + \frac{1}{8} n \log[n]^3$
6	$-n + \frac{45}{32} n \log[n] + \frac{35}{64} n \log[n]^2 + \frac{1}{32} n \log[n]^3 + \frac{1}{64} n \log[n]^4$
7	$-n + \frac{231}{160} n \log[n] + \frac{15}{32} n \log[n]^2 + \frac{11}{128} n \log[n]^3 + \frac{1}{640} n \log[n]^5$
8	$-n + \frac{457}{320} n \log[n] + \frac{1937 n \log[n]^2}{3840} + \frac{29}{512} n \log[n]^3 + \frac{17 n \log[n]^4}{1536} - \frac{n \log[n]^5}{2560} + \frac{n \log[n]^6}{7680}$

Figure 119: Expressions for ptf[ ]

VITA

Dr. Michael A. Yukish

Michael Yukish is the Head of the Manufacturing Product and Process Design Department at the Applied Research Laboratory of The Pennsylvania State University. He has been employed as a Research Assistant at ARL since 1993. The focus of his research at ARL is in the areas of multidisciplinary design optimization, simulation based design, and conceptual trade space exploration for complex systems.

He received a B.S. in Physics from Old Dominion University in 1983, an M.S. in Mechanical Engineering from The Pennsylvania State University in 1997, and a Ph.D. in Mechanical Engineering from The Pennsylvania State University in 2004.

From 1985 to 1992 Michael Yukish served on active duty in the United States Navy as a Naval Aviator. He has over 1800 flight hours in the E-2C Hawkeye and the T-2C Buckeye, and made multiple deployments to North Atlantic, the Mediterranean, and the Indian Ocean. He is now a Commander in the US Naval Reserve.