MINIMUM-PERIMETER TILING IN PARALLEL COMPUTATION

by

Jonathan A. Yackel

Computer Sciences Technical Report #1170

August 1993

MINIMUM-PERIMETER TILING IN PARALLEL COMPUTATION

By

Jonathan A. Yackel

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy
(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

1993

Abstract

This thesis is primarily concerned with two combinatorial optimization problems that have applications in parallel computing. In each problem the goal (from an applications point of view) is to partition the cells of a grid (of two or more dimensions) evenly among a number of processors so as to minimize interprocessor communication subject to load balancing constraints. In the *perimeter minimization* problem the communication is measured by interprocessor "boundary", while in the *diversity minimization* problem communication is measured by summing the number of distinct processors in the "slices" of the grid. Besides the parallel computing applications, these problems are of intrinsic interest as combinatorial problems. For the two-dimensional versions of both these problems, a single theory based on geometric arguments provides good lower bounds on both objective functions and also characterizes the forms of optimal and nearly optimal solutions. The results include a method for generating provably optimal solutions for a large class of problems by "tiling" the grids with optimal shapes. This theory generalizes to arbitrary dimensions for the diversity minimization problem. A different theory provides lower bounds for many-dimensional versions of perimeter minimization.

A tiling-based heuristic for diversity minimization provides the basis for the empirical results in the the thesis. The heuristic, which incorporates a high-level genetic algorithm, is naturally parallel and has been implemented on a Thinking Machines CM-5. The code is efficient (spending less than 3% of the total computing time on interprocessor communication) and produced solutions of better quality than a previous heuristic developed by database researchers for diversity minimization. The code computed solutions within 4.2% of the best known lower bounds on all test problems, and solved a million-variable problem to optimality in 70 seconds.

Acknowledgements

I would like to thank Lisa Pankratz Yackel for her love and support, Robert Meyer for his invaluable guidance and encouragement, Eric Bach for his advice, and Shahram Ghandeharizadeh for the seed of this thesis.

Contents

Abstract **ii**

Acknowledgements **iv**

1 Introduction **1**

1.1 Perimeter minimization 4

1.2 Diversity minimization 6

1.2.1 Minimizing overhead and query response time 8

1.2.2 Minimizing data duplication in parallel hash-joins 8

1.3 Computational complexity 9

1.4 Unifying concepts 12

2 2-Dimensional Theory **16**

2.1 Lower bound on semi- and 2-perimeter 16

2.2 Minimum-perimeter and 2-perimeter groups 21

2.2.1 Optimal rectangles 26

2.2.2 Optimal Irregular Tiles 28

2.3 Optimal Tilings 33

2.3.1 Minimum-perimeter tilings 35

2.3.2 Minimum diversity toroidal tiling 37
2.4 Lower Bounds as Functions of Problem Size 45
2.5 Non-uniform Grids 46

3 Many-Dimensional Theory **48**
3.1 Lower bounds on d-perimeter 48
3.2 Lower bound on surface area 52

4 Heuristics **56**
4.1 A Diversity Minimization Algorithm 56
4.2 Data Structures 59

5 A genetic algorithm **67**
5.1 A generic GA 68
5.2 A minimum diversity GA 69
5.2.1 Encoding 70
5.2.2 Fitness Function 70
5.2.3 Initial Population 71
5.2.4 Genetic Operators and "Survival" Policies 71

6 Computational Results **74**
6.1 Interprocessor Communication 79
6.2 Effect of random seed 81
6.3 Effect of survival strategies 82
6.4 Effect of crossover and mutation rates 83

7 Conclusions and Future Research Directions **86**
7.1 Summary and Conclusions 86

7.2 Future Research Directions 88

APPENDICES

A Problem Variations **90**

A.1 Nonlinear objective functions 90

A.2 Minimizing maximum diversity 91

A.3 Relaxed load balancing 92

A.4 Weighted versions 92

A.5 Target diversities 93

A.6 More general variations 94

B Integer Programming Models for Diversity Minimization **95**

C A Diagonal-based Solution Technique **98**

D Elongated Grids **101**

Chapter 1

Introduction

As distributed and parallel computer systems are used to solve larger and larger problems, the issue of mapping the problem data and computational tasks onto multiple processors becomes more important because a poor mapping may cause a large amount of interprocessor communication or degrade the throughput of the system. This thesis focuses principally on two combinatorial optimization problems, both of which involve finding the least expensive way (as measured by some objective function) of partitioning the cells of a grid (which correspond to data and/or computation tasks) into a specified number of groups of specified size (usually the groups sizes are nearly equal, for load balancing reasons). These problems will be shown to model several problems that arise when partitioning data among the processors in parallel and distributed computer systems in order to minimize communication costs. Geometric concepts will be used to derive lower bounds on the costs of the partitions, and techniques for constructing optimal partitions for some grid classes will be presented.

The first problem we consider is the "geometric" problem of partitioning the grid cells of a 2-dimensional rectilinear domain into a specified number of groups

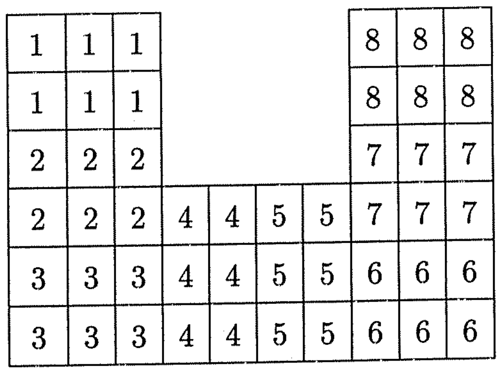


Figure 1: A minimum perimeter partition of a rectilinear domain

(subsets of cells) of specified size so as to minimize the sum of the lengths of the group perimeters. We refer to this problem hereafter as "perimeter minimization". The perimeter of a group is the length of the boundary enclosing the cells in the group. A set of cells is said to be *connected* if for every pair of cells c_i, c_j there is a *path* of cells in the set from c_i to c_j such that adjacent cells in the path share an edge. Any partition of a 2-dimensional grid into groups creates one or more maximal connected sets of cells per group, which we call "tiles". Given a group g, the perimeter of the group, denoted by $\mathcal{P}(g)$, is formally defined as the sum of the perimeters of these tiles. The objective function for the perimeter minimization problem is therefore $\sum_g \mathcal{P}(g)$. An optimal partition of a domain with 48 grid cells into 8 equal size groups is shown in figure 1 (each group is made up of a single tile with a perimeter of length 10 for a total perimeter of 80).

The other problem addressed here is a combinatorial problem whose objective function is determined by the $(d-1)$-dimensional *slices* of a d-dimensional grid. For ease of discussion, a d-dimensional grid can be be imagined to be aligned with a d-dimensional Cartesian coordinate system so that each grid cell can be identified by an integer d-dimensional coordinate vector. A slice is the subgrid formed by the cells that have a common coordinate value in a particular dimension; for example, the slices of a 2-dimensional grid are the rows and the columns. Again, we assume that some processor index is to be assigned to each cell. Given a partition of the grid cells into groups corresponding to processor indices, the *diversity* of a slice is defined to be the number of distinct indices occurring in that slice. The problem considered here is to partition the cells into a specified number of groups of specified size so as to minimize the sum of the slice diversities. This problem will be referred to as "diversity minimization". It can be formulated as an integer transportation problem with a nonconvex objective function, or as a linear integer program. The details of these formulations are contained in appendix B. The partition shown in figure 1 is optimal for both the minimum perimeter and the minimum diversity problem. Figure 2 shows a partition of a 5×5 grid into 5 equal size groups that optimizes total diversity but does not optimize total perimeter because too many tiles are "split" into more than one connected set. Because of its slicewise definition, diversity is less affected by such splitting of groups. Notice that in figure 2 there is a diversity of 3 for each row and 2 for each column, for a total diversity of 25, while the perimeters for groups 1 through 5 are 10, 10, 14, 14, and 12 respectively for a total perimeter of 60. The minimum possible perimeter for this problem is 52. For both problems, the notation $\mathcal{A}_g$ is used to denote the specified size or the "area" for group g.

Several variations on the problems stated above may be of interest depending on the underlying application. Alternate objective functions and/or constraints are the principal modifications which we consider. Some of these variations are discussed in appendix A.

1	4	4	5	5
1	1	2	5	5
1	1	2	2	3
3	4	2	2	3
3	4	4	5	3

Figure 2: A 5×5 grid partitioned for minimum slice diversity

1.1 Perimeter minimization

In many computations performed on parallel and distributed systems, the data and computational tasks that make up the problem are related by some kind of grid structure and the communication is decentralized, occurring only between neighbors in the grid structure. In fluid flow problems solved with domain decomposition and obstacle problems solved by successive overrelaxation, a 2-dimensional physical region is discretized by dividing it into grid cells[5, 13]. Values at each grid cell are iteratively computed based on the values at the north, south, east, and west neighbor cells. For cells on the boundary of the region, boundary conditions may be used in the computations. If the grid cells are assigned to processors (that is, the computation associated with each cell is performed by a particular processor), then sharing data between neighboring cells may involve interprocessor communication. With respect to communication corresponding to the domain boundary, there are at least two possible simplifying assumptions that mesh with the perimeter minimization model. One could assume that computing the boundary values is done locally by the processors (*e.g.* using boundary conditions) with no communication necessary. Alternatively one could assume some fixed amount of communication proportional to the number of boundary edges for each boundary cell. In either case the amount of communication corresponding to the grid boundary is a constant. Thus the communication due to inter-group perimeter is the only part of interprocessor communication that varies with the partitioning strategy.

Edge detection in computer vision systems is another application in which the data and the computational tasks have a grid structure. An image is represented as a 2-dimensional array of intensity values. To detect edges, a linear combination of each value and its north, south, east, and west neighbor values is computed[17]. If the image data is distributed across multiple processors, and the edge detection computation at each pixel is computed by the processor on which the pixel data reside, some computations require interprocessor communication. The communication associated with the image boundary is handled as described above.

For the computations described above, there exists the inherent problem of optimally distributing the data and the computational tasks among the processors. If the objectives are to minimize interprocessor communication and balance the computational load, the perimeter minimization problem is a good model under some assumptions about the parallel architecture. Each grid cell represents a data value and/or a computational task, and the number of groups into which to partition the grid is equal to the number of processors in the system. Load balancing is achieved by specifying $\mathcal{A}_g$ for each group. In most cases, the group sizes will be as equal as possible – modelling the case where the processors are identical. Different group sizes can be specified to model processors with unequal speeds or unequal amounts of memory. If we assume that there is an equal cost for communication between any pair of processors (as in a distributed system), then each unit of the perimeter corresponds to a unit of interprocessor communication or boundary value computation, and minimizing the interprocessor communication is equivalent to perimeter minimization. To explain this equivalence let σ denote the length of the inter-group boundary for a partition, *i.e.*, the number of edges between adjacent cells of different groups; and let β denote the length of the domain boundary. Notice that for any partition, the sum of the group perimeters $\sum_g \mathcal{P}(g)$ is equal to $2\sigma + \beta$ (each inter-group boundary edge is counted twice, once for each bordering group). The communication cost for a partition is $k_1\sigma + k_2\beta$, where k_1 and k_2 are chosen to relate the costs of communication across interior edges and domain boundary edges. Since β is a constant, minimizing $\sum_g \mathcal{P}(g)$ is equivalent to minimizing $k_1\sigma + k_2\beta$.

1.2 Diversity minimization

The physical design of distributed database systems presents several data partitioning problems. Many distributed systems are designed with a "shared-nothing" architecture in which each processor has its own disk drive on which it stores a portion of the database [6, 3, 18, 12]. Communication in such a system is centralized (messages travel only between the distributed processors and a coordinating processor), with a cost depending on the number of processors communicating. Partitioning the data on such a system in order to minimize the response time for various database operations may be modelled by the diversity minimization problem. The difference between the centralized communication model and the local communication model of the other applications necessitates the development of a different formal model, namely the diversity minimization problem.

In the relational database model, information is stored as a collection of "tuples". Each tuple is made up of a list of attribute values. For example, in a relational database of employees, there may be a tuple for each employee having attributes such as name, salary, age, social security number, etc. By partitioning the domain of each of d attributes into ranges, the domain of tuple values can be mapped onto a d-dimensional grid with a $(d-1)$-dimensional slice corresponding to each attribute range. For example consider the employee relation EMP in figure 3. The attributes are age and salary. Three partitions in each attribute are created, producing the 3×3 grid in figure 3. Each grid cell corresponds to the tuples of the relation that have attribute values in the corresponding ranges, *e.g.*, the upper left cell of the grid corresponds to tuples that have salary values less than \$20,000 and age values between 0 and 25. Each set of tuples in a grid cell is referred to as a "fragment" of the relation. The tuples of a relation may be

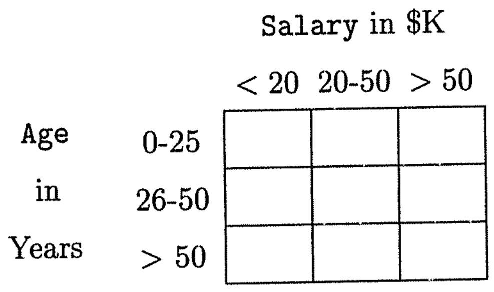


Figure 3: The EMP relation

partitioned among the processors by partitioning the set of fragments.

1.2.1 Minimizing overhead and query response time

Diversity minimization is a model for overhead minimization, and under certain conditions response time minimization, for selection queries on distributed databases. A selection query selects all tuples from the database that share a common value in a particular attribute. Such a query must access all the data corresponding to a slice of the grid, and utilizes all processors which own fragments in that slice. This allows each processor to execute a portion of a query in parallel with the other processors, resulting in a lower response time for the query. However, there is communication overhead associated with initiating and terminating a query on multiple processors, and this overhead increases as a function of the number of processors used to execute a query. This overhead is primarily in the form of additional messages to control the execution of the query on additional processors and, in the Gamma database machine [6], increases linearly with the number of employed processors. For this reason, it is desirable to minimize the number of processors used in a query. Each possible selection query corresponds to a grid slice, and the number of processors used to execute the query corresponds to the diversity of the slice. Minimizing the total overhead is therefore equivalent to diversity minimization.

The partitions that minimize overhead also, under certain assumptions, minimize the average query response time – a nonlinear function of the slice diversities. See appendix A for these results.

1.2.2 Minimizing data duplication in parallel hash-joins

The problem of minimizing data duplication for multiple parallel hash-joins can also be modeled by diversity minimization. If the parallel joins $R_1 \bowtie R_2, R_1 \bowtie$

$R_3, R_1 \bowtie R_4, \ldots$, with join attributes chosen from $\{A_1, A_2, \ldots, A_D\}$, are to be performed on a distributed system, one approach is to partition R_1 across the processors via a hash function on A_1, A_2, $A_3, \ldots A_D$. This divides the relation into fragments arranged in a D-dimensional grid (one dimension for each of the join attributes). In the join $R_1 \bowtie R_i$ on attribute A_j, each tuple t of R_i must be matched with the tuples of R_1 which share the value of t in attribute A_j. If a number of processors own tuples which might match t, then t must be duplicated and sent to each of these processors. The processors to which t needs to be sent are exactly those to which tuples matching t in attribute A_j would hash, *i.e.,* the processors in a $(D-1)$-dimensional grid slice in dimension j. Clearly the amount of duplication is proportional to the diversity of the corresponding slice. Assuming equal frequency of all joins, minimizing total data duplication is equivalent to diversity minimization. The load balancing constraints once again model the similarity of the processors with regard to storage space.

Various methods of partitioning are common in distributed database machines. Range partitioning divides the database into fragments by partitioning the values of a single attribute into ranges. This is equivalent to creating a 1-dimensional grid of fragments. Selection queries on the partitioning attribute execute on a single processor, but queries on any other attribute require all of the processors. Ghandeharizadeh generalized range partitioning to many dimensions to overcome this disadvantage [8].

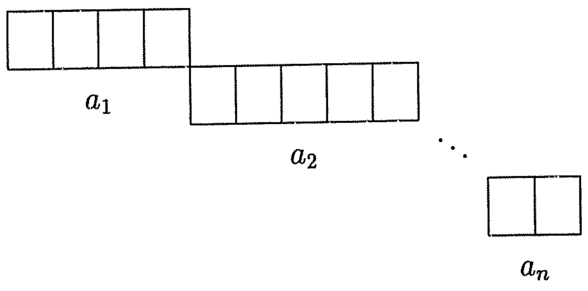
1.3 Computational complexity

Perimeter minimization is equivalent to a restricted form of the graph partitioning problem. In its full generality, the graph partitioning problem is to partition the nodes of a given graph into a specified number of disjoint subsets of specified sizes so as to minimize the number of "cut edges", *i.e.*, edges connecting nodes in distinct subsets. Given a domain composed of grid cells, construct a "grid graph" with a node for each grid cell and an edge between each pair of nodes corresponding to adjacent grid cells. Perimeter minimization is therefore equivalent to graph partitioning on grid graphs. Below, the problem is shown to be NP-hard, however the proof uses a non-rectangular subset of a rectangular grid, so the complexity of perimeter minimization restricted to rectangular graphs remain an open question.

Diversity minimization for general rectilinear domains is NP-hard. The NP-complete partition problem can be reduced to diversity minimization. The partition problem is: given a set of positive integers $a_1, a_2, \ldots, a_n$, answer the question "is there a partition of the a's into two disjoint subsets X and Y such that $\sum_{a_i \in X} a_i = \sum_{a_i \in Y} a_i = \frac{1}{2}\sum_{i=1}^{n} a_i$?"

Theorem 1 *The diversity minimization problem and the perimeter minimization problem are NP-hard.*

Proof: Given an instance of the partition problem $a_1, a_2, \ldots, a_n$, we construct an instance of diversity minimization which has minimum value $\sum_{i=1}^{n} a_i + n$ if and only if there exists a partition of $a_1, a_2, \ldots, a_n$. We divide a domain of $\sum_{i=1}^{n} a_i$ cells into two equal size groups, where the domain has the cells arranged in n rows and $\sum_{i=1}^{n} a_i$ columns as illustrated below.



For any feasible assignment of the cells, the diversity for each column will be 1, and the diversity for each row will be either 1 or 2. Therefore $\sum_{i=1}^{n} a_i + n$ is a lower bound on the total diversity. If there exits a partition X, Y of $a_1, a_2, \ldots, a_n$, then there is a feasible assignment of the cells of the domain which achieves a total diversity of $\sum_{i=1}^{n} a_i + n$ by assigning all cells in row i to group 1 if $a_i \in X$ and to group 2 if $a_i \in Y$. Conversely, if there is a feasible assignment of the cells which achieves a total diversity of $\sum_{i=1}^{n} a_i + n$, (*i.e.*, a diversity of 1 for each row), then there exists a partition X, Y of $a_1, a_2, \ldots, a_n$ which has $a_i \in X$ if the cells in row i are assigned to group 1 and $a_i \in Y$ if the cells in row i are assigned to group 2. Therefore there is a partition of $a_1, a_2, \ldots, a_n$ if and only if there is a minimum total diversity of $\sum_{i=1}^{n} a_i + n$.

Using essentially the same reduction, partition is reduced to a decision version of the minimum perimeter problem. Given an instance of the partition problem $a_1, a_2, \ldots, a_n$, construct an instance of perimeter minimization with the same domain as for the previous reduction split into two equal size groups. There is a partition of the numbers $a_1, a_2, \ldots, a_n$ if and only if the minimum value of the total perimeter is $2\sum_{i=1}^{n} a_i + 2n$. The perimeter of the domain must contribute to the total perimeter of any feasible solution, therefore $2\sum_{i=1}^{n} a_i + 2n$ is a lower bound on the total perimeter. If there is a partition of the numbers $a_1, a_2, \ldots, a_n$, then the same assignment of cells to groups as for the diversity problem yields a total perimeter of $2\sum_{i=1}^{n} a_i + 2n$. If there is a feasible assignment of cells which achieves a total perimeter of $2\sum_{i=1}^{n} a_i + 2n$, then each row of the domain must be assigned to a single group and the implicit partition of the rows corresponds to a partition of the a_i. $\blacksquare$

In most of the applications of diversity minimization, the domain is a rectangular grid. The complexity of diversity minimization on this restricted class of domain is not known. Diversity minimization is a restriction of a data aggregation problem of Helman[15]. The general data aggregation problem can be stated as follows: given a set of elements and a collection of subsets of these elements, partition the elements into a specified number of groups of specified size in order to minimize the sum of the diversities of the subsets, where again the diversity of a subset is the number of groups intersecting the subset. If the set of elements is the set of grid cells and the collection of subsets is the collection of all slices of the grid, then the grid problem can be seen to be a restricted class of the data aggregation problem.

1.4 Unifying concepts

This section introduces the central geometric concepts of "semi-perimeter" and "d-perimeter". The combinatorial diversity minimization problem is shown to be equivalent to the geometric problem of d-perimeter minimization. Semi-perimeter and d-perimeter are shown to be related, providing a single framework in which to address both perimeter and diversity minimization.

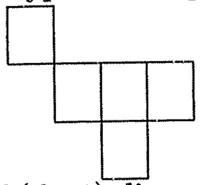
Given a 2-dimensional domain partitioned into groups, the semi-perimeter of a *connected* group of cells g, denoted by $\mathcal{S}(g)$, is defined as the width plus height of the smallest rectangle enclosing the group. If a group is made up of more than one maximal connected subgroup (tile), then its semi-perimeter is the sum of the semi-perimeters of the tiles.

We will now show that for a large class of cell groups, or "configurations", that play a key role in the theory below, the perimeter is exactly twice the semi-perimeter, and that for other configurations, perimeter is more than twice the semi-perimeter. A group of cells is "slice-convex" if for any two cells c_1, c_2 of the group in the same slice, the smallest rectangle containing c_1 and c_2 lies entirely in the group. (Note that a slice-convex configuration need not be convex.)

Lemma 2 *For a group of cells g, $\mathcal{P}(g) \geq 2\mathcal{S}(g)$. Furthermore, $\mathcal{P}(g) = 2\mathcal{S}(g)$ if and only if each tile of g is slice-convex.*

Proof: Notice that the perimeter of a group is the sum of its tile perimeters. Consider a tile t_i of g. There are at least two cell edges forming part of the group boundary in each slice of t_i. Therefore each slice intersecting t_i contributes at least 2 to the perimeter of g, but exactly 1 to $\mathcal{S}(t_i)$. Since $\mathcal{S}(t_i)$ is the number of slices intersecting t_i, $\mathcal{P}(t_i) \geq 2\mathcal{S}(t_i)$. If t_i is slice-convex, each slice intersecting t_i contains exactly 2 group border edges, so that $\mathcal{P}(t_i) = 2\mathcal{S}(t_i)$. If t_i is not slice-convex, there is a slice of t_i with more than 2 group border edges, therefore $\mathcal{P}(t_i) > 2\mathcal{S}(t_i)$. The lemma follows since $\mathcal{S}(g) = \sum_i \mathcal{S}(t_i)$ and $\mathcal{P}(g) = \sum_i \mathcal{P}(t_i)$. ∎

Since lemma 2 links perimeter and semi-perimeter, a lower bound on perimeter may be obtained by developing a bound on semi-perimeter.

Consider the diversity minimization problem. Because reordering the slices of a grid does not affect the diversities, we may consider the cells in a particular group in their most "compact" configuration. For a group g of cells, define the compact configuration $\mathcal{C}(g)$ to be the group after the slices of the grid have been permuted so that in each dimension the slices containing cells in the group are contiguous. For example, Figure 4 shows a group g of 9 cells and $\mathcal{C}(g)$. For a group g of cells in a d-dimensional grid, we define the d-perimeter of the group, denoted $\mathcal{P}_d(g)$ to be the sum of the d dimensions of the smallest hyper-rectangle enclosing $\mathcal{C}(g)$. For example, the 2-perimeter of the configuration  is $4+3 = 7$.

Another way to interpret $\mathcal{P}_d(g)$ is as the total number of $(d-1)$-dimensional slices

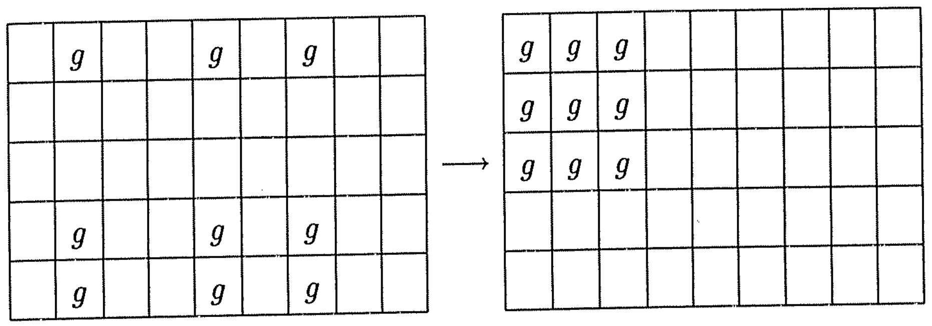


Figure 4: Permuting slices to produce a compact configuration.

intersecting g. To relate the slice diversities to the group d-perimeters, define the slice-group incidence vector χ:

$$\chi_s^g = \begin{cases} 1 & \text{if group } g \text{ intersects slice } s \\ 0 & \text{otherwise.} \end{cases}$$

The diversity of slice s is $\sum_g \chi_s^g$, and the sum of the slice diversities is $\sum_s \sum_g \chi_s^g$. The perimeters can also be expressed in terms of χ: $\mathcal{P}_d(g) = \sum_s \chi_s^g$. The sum of the group d-perimeters is $\sum_g \sum_s \chi_s^g$ which is equivalent, by reversing the order of summation, to the sum of the slice diversities. Therefore the problem of "d-perimeter minimization," *i.e.*, partitioning a d-dimensional grid into groups of specified sizes so as to minimize $\sum_g \mathcal{P}_d(g)$, is equivalent to diversity minimization.

Restricting d to 2, the 2-perimeter and semi-perimeter concepts are related by the following lemma:

Lemma 3 *Given a group of cells g,*

$$\mathcal{S}(g) \geq \mathcal{S}(\mathcal{C}(g)) \geq \mathcal{P}_2(\mathcal{C}(g)) = \mathcal{P}_2(g).$$

Furthermore

$$\mathcal{S}(\mathcal{C}(g)) = \mathcal{P}_2(g)$$

if g is connected or slice-convex.

Proof: The left inequality in the lemma follows because the compaction operation can coalesce two tiles, but cannot split a tile apart. The middle inequality follows because each slice intersecting $\mathcal{C}(g)$ contributes to the semi-perimeter of at least one tile of $\mathcal{C}(g)$. The right equality holds because compaction does not change the 2-perimter. The conditional equality follows because if g is connected or slice-convex, each slice through g intersects only one tile. ∎

This relationship allows us to derive a common lower bound on semi-perimeter and 2-perimeter as a function of the size of the group of cells.

Lemma 4 *If $l_{\mathcal{S}}(n)$ is a lower bound on the semi-perimeter of any group consisting of n cells, then $l_{\mathcal{S}}(n)$ is also a lower bound on the 2-perimeter of a group of n cells; and if $l_{\mathcal{P}}(n)$ is a lower bound on 2-perimeter, then $l_{\mathcal{P}}(n)$ is a lower bound on semi-perimeter.*

Proof: Since $\mathcal{P}_2(g)$ is a lower bound on $\mathcal{S}(g)$, any lower bound on $\mathcal{P}_2(g)$ must also be a lower bound on $\mathcal{S}(g)$. The other part of the proof is by contradiction. Assume that $l_{\mathcal{S}}(n)$ is a lower bound on semi-perimeter and that there exists a group of n cells, g, with 2-perimeter strictly less than $l_{\mathcal{S}}(n)$. Let h and w be the height and width respectively of the smallest rectangle enclosing $\mathcal{C}(g)$, *i.e.*, $\mathcal{P}_2(g) = h + w$. It is clearly possible to construct a connected configuration g' of n cells enclosed by an $h \times w$ rectangle since $hw \geq n$. Since g' is connected, $\mathcal{S}(g') = h + w < l_{\mathcal{S}}(n)$, a contradiction. ∎

Chapter 2

2-Dimensional Theory

This chapter deals with perimeter and diversity minimization on 2-dimensional domains and grids. It unifies the results from perimeter minimization [20] and diversity minimization [21]. A lower bound on group semi-perimeter and 2-perimeter is developed. The derivation of this lower bound yields a geometric characterization of tiles with minimum semi- and 2-perimeter. A technique for generating all optimal tiles, *i.e.*, tiles with minimum perimeter, of a fixed number of cells is described. We then show how optimal tiles may be used to tile several classes of grids, yielding provably optimal solutions to the diversity and perimeter minimization problems.

2.1 Lower bound on semi- and 2-perimeter

By lemma 4 a lower bound for semi-perimeter is a lower bound for 2-perimeter and vice versa. We derive a bound on semi-perimeter. Let the notation $|g|$ denote the number of grid cells in the group g. Let $\mathcal{A}^* : I\!N \longrightarrow I\!N$ be the function mapping a positive integer semi-perimeter to the maximum number of cells in a group with with that semi-perimeter, *i.e.*,

$$\mathcal{A}^*(n) = \quad \max_g \quad |g| \quad \text{such that} \quad \mathcal{S}(g) = n.$$

Theorem 5 *Given a semi-perimeter $\mathcal{S}$, the maximum size group with semi-perimeter $\mathcal{S}$ is a $\frac{\mathcal{S}}{2} \times \frac{\mathcal{S}}{2}$ square if $\mathcal{S}$ is even, and is a $\left(\frac{\mathcal{S}-1}{2}\right) \times \left(\frac{\mathcal{S}+1}{2}\right)$ rectangle is $\mathcal{S}$ is odd,* i.e.,

$$\mathcal{A}^*(\mathcal{S}) = \begin{cases} \left(\frac{\mathcal{S}}{2}\right)^2 & \text{if } \mathcal{S} \text{ is even} \\ \\ \left(\frac{\mathcal{S}-1}{2}\right)\left(\frac{\mathcal{S}+1}{2}\right) & \text{if } \mathcal{S} \text{ is odd} \end{cases}.$$

Proof: For a group g let $\mathcal{S}_x(g)$ and $\mathcal{S}_y(g)$ denote the width and height respectively of the smallest rectangle enclosing the group. Given any group g with semi-perimeter $\mathcal{S}$ and area $\mathcal{A}$, there is a rectangular tile with dimensions $\mathcal{S}_x(g) \times \mathcal{S}_y(g)$, with semi-peimeter $\mathcal{S}_x(g) + \mathcal{S}_y(g) = \mathcal{S}$ and area $\mathcal{S}_x(g)\mathcal{S}_y(g) \geq \mathcal{A}$. Therefore we need only consider rectangular tiles as candidates for maximum area groups. To find the rectangle of maximum area with semi-perimeter $\mathcal{S}$, we maximize $\mathcal{S}_x\mathcal{S}_y$ subject to $\mathcal{S}_x + \mathcal{S}_y = \mathcal{S}$. This result can be sharpened by constraining $\mathcal{S}_x$ and $\mathcal{S}_y$ to be integers. Of all pairs of *integers* with a certain sum, the pair with the greatest product is the one with the numbers closest together. If $\mathcal{S}$ is even, this is achieved by setting $\mathcal{S}_x = \mathcal{S}_y = \frac{\mathcal{S}}{2}$, and if $\mathcal{S}$ is odd, it is achieved when $\mathcal{S}_x$ and $\mathcal{S}_y$ differ by 1. ∎

By "inverting" the function $\mathcal{A}^*$, we obtain a function $\mathcal{S}^* : I\!N \longrightarrow I\!N$ which is defined as *the function mapping a group size to the minimum semi-perimeter of all groups of that size, i.e.*

$$\mathcal{S}^*(n) = \quad \min_g \quad \mathcal{S}(g) \quad \text{such that} \quad |g| = n.$$

Theorem 6 *Given a group size $\mathcal{A}$, the smallest achieveable semi-perimeter is*

$$\mathcal{S}^*(\mathcal{A}) = i\left\lceil \mathcal{A}^{1/2} \right\rceil + (2-i)\left\lfloor \mathcal{A}^{1/2} \right\rfloor$$

where i is the smallest positive integer such that

$$\left\lceil \mathcal{A}^{1/2} \right\rceil^{i} \left\lfloor \mathcal{A}^{1/2} \right\rfloor^{2-i} \geq \mathcal{A}.$$

Furthermore, the lower bound on $\mathcal{S}$ is tight, i.e., *for any $\mathcal{A}$, there is a group of $\mathcal{A}$ cells with a semi-perimeter of* $\left\lceil \mathcal{A}^{1/2} \right\rceil + (2-i)\left\lfloor \mathcal{A}^{1/2} \right\rfloor$.

Proof: We may bound the semi-perimeter of any group of $\mathcal{A}$ cells from below by finding the smallest semi-perimeter $\mathcal{S}$ that satisfies $\mathcal{A}^*(\mathcal{S}) \geq \mathcal{A}$, since this implies $\mathcal{A} > \mathcal{A}^*(\mathcal{S}-1)$, which means that a semi-perimeter of $\mathcal{S}-1$ is not achievable for a group size of $\mathcal{A}$.

Consider the following sequence of rectangles.

		Area	$\mathcal{S}()$
Q_0 :	0×0		
Q_1 :	1×0		
Q_2 :	1×1	1	2
Q_3 :	2×1	2	3
Q_4 :	2×2	4	4
Q_5 :	3×2	6	5
Q_6 :	3×3	9	6
	$\vdots$		

We call these rectangles "quasi-squares" since the dimensions of each rectangle differ by at most 1. Note that the areas of the quasi-squares in the sequence are strictly increasing after the second, the area of the ith quasi-square Q_i for $i \geq 2$

is $\mathcal{A}^*(i)$ (from theorem 5), and the semi-perimeter for the quasi-squares increases by 1 at each step. The areas of these quasi-squares are the points at which the lower bound on the semi-perimeter increases by 1.

For an arbitrary $\mathcal{A}$, there is a unique smallest quasi-square Q_j whose area is at least $\mathcal{A}$. Since the area of Q_j is at least $\mathcal{A}$, by selecting $\mathcal{A}$ cells from Q_j a semi-perimeter of at most $\mathcal{S}(Q_j)$ is achievable for $\mathcal{A}$ cells. Since the area of Q_{j-1} is smaller than $\mathcal{A}$, a semi-perimeter of $\mathcal{S}(Q_{j-1}) = \mathcal{S}(Q_j) - 1$ is not achievable for $\mathcal{A}$ cells. Therefore the smallest semi-perimeter achievable for any tile of $\mathcal{A}$ cells is $\mathcal{S}(Q_j)$. It is easy to see that each dimension of Q_j is either $\lfloor \mathcal{A}^{1/2} \rfloor$ or $\lceil \mathcal{A}^{1/2} \rceil$ and that $\mathcal{S}(Q_j)$ is exactly the semi-perimeter bound in the statement of the theorem. ∎

The functions $\mathcal{S}^*$ and $\mathcal{A}^*$ are partial inverses in the sense that $\mathcal{A}^*(\mathcal{S}^*(n)) \geq n$, $\mathcal{A}^*(\mathcal{S}^*(n) - 1) < n$, and $\mathcal{S}^*(\mathcal{A}^*(n)) = n$, relationships which can be derived from the function definitions. In the following lemma we present a perimeter optimality test for any group of cells.

Lemma 7

$$\mathcal{S}(g) = \mathcal{S}^*(|g|)$$

if and only if

$$\mathcal{A}^*(\mathcal{S}(g) - 1) < |g| \ . \qquad (1)$$

Proof: If (1) holds, it is not possible to achieve an area of $|g|$ via any group with smaller semi-perimeter, since the maximum achieveable area for any such group is less than $|g|$. On the other hand if (1) does not hold, then by theorem 6 there is a group $\hat{g}$ with $|\hat{g}| = |g|$ and $\mathcal{S}(\hat{g}) < \mathcal{S}(g)$. ∎

Lemma 7 can be used to show that $\mathcal{S}^*(\mathcal{A})$ can also be written in closed form.

Theorem 8 $\mathcal{S}^*(\mathcal{A}) = \lceil 2\,\mathcal{A}^{1/2} \rceil$.

Proof: To see that $\lceil 2\,\mathcal{A}^{1/2} \rceil$ is a lower bound on $\mathcal{S}^*(\mathcal{A})$, recall that semi-perimeter is the height plus the width of the smallest rectangle enclosing the group. Therefore a lower bound on semi-perimeter is $\min\{h+w | hw \geq \mathcal{A}\} = 2\mathcal{A}^{1/2}$. Since h and w must be integers, we can round up to the next integer to get $\lceil 2\,\mathcal{A}^{1/2} \rceil$.

Since $\mathcal{S}^*$ yields the best possible lower bound, we need only show $\mathcal{S}^*(\mathcal{A}) \leq \lceil 2\mathcal{A}^{1/2} \rceil$. Let $\mathcal{S}$ be the semi-perimeter of some group of $\mathcal{A}$ cells. We show that if $\mathcal{S} = \mathcal{S}^*(\mathcal{A})$ then $\mathcal{S} = \lceil 2\mathcal{A}^{1/2} \rceil$.

$$\begin{aligned} \mathcal{S} &= \lceil 2\mathcal{A}^{1/2} \rceil \\ \iff \mathcal{S} - 2\mathcal{A}^{1/2} &< 1 \\ \iff \mathcal{S} - 1 &< 2\mathcal{A}^{1/2}. \end{aligned}$$

From lemma 7 we have

$$\begin{aligned} \mathcal{S} &= \mathcal{S}^*(\mathcal{A}) \\ \iff \mathcal{A}^*(\mathcal{S}-1) &< \mathcal{A} \\ \iff \left\lceil \frac{\mathcal{S}-1}{2} \right\rceil^r \left\lfloor \frac{\mathcal{S}-1}{2} \right\rfloor^{2-r} &< L \end{aligned} \tag{2}$$

where $r = (\mathcal{S}-1) \mathit{mod}\, 2$.

If $r = 0$ then $\left\lfloor \frac{\mathcal{S}-1}{2} \right\rfloor = \frac{\mathcal{S}-1}{2}$ so (2) becomes

$$\begin{aligned} \left(\tfrac{\mathcal{S}-1}{2}\right)^2 &< \mathcal{A} \\ \iff \mathcal{S} - 1 &< 2\mathcal{A}^{1/2}. \end{aligned}$$

If $r = 1$ then $\left\lfloor \frac{\mathcal{S}-1}{2} \right\rfloor = \frac{\mathcal{S}}{2} - 1$ and $\left\lceil \frac{\mathcal{S}-1}{2} \right\rceil = \frac{\mathcal{S}}{2}$ so (2) becomes

$$\begin{aligned} \left(\tfrac{\mathcal{S}}{2} - 1\right) \tfrac{\mathcal{S}}{2} &< \mathcal{A} \\ \iff \mathcal{S}^2 - 2\mathcal{S} &< 4\mathcal{A}. \end{aligned}$$

Since both sides of the last inequality are even integers, we may add 1 to the LHS and maintain the inequality.

$$\begin{array}{rrcl} & \mathcal{S}^2 - 2\mathcal{S} + 1 & < & 4\mathcal{A}. \\ \iff & (\mathcal{S}-1)^2 & < & 4\mathcal{A} \\ \iff & \mathcal{S}-1 & < 2\mathcal{A}^{1/2}. & \end{array}$$

∎

Rosenberg [16] showed in the context of graph partitioning that $\lceil 2\mathcal{A}^{1/2} \rceil$ is a lower bound (as a corollary to a more general result), and formulated the tight lower bound as the closed form expression: $\lfloor \mathcal{A}^{1/2} \rfloor + \left\lceil \frac{\mathcal{A}}{\lfloor \mathcal{A}^{1/2} \rfloor} \right\rceil$, without realizing that the two actually co-incide.

Table 1 contains minimum semi-perimeter values for areas up to 56 constructed via theorem 6.

2.2 Minimum-perimeter and 2-perimeter groups

In this section we discuss the characteristics of groups which have minimum 2-perimeter and perimeter for their size. The notation $\mathcal{P}^*(\mathcal{A})$ denotes the minimum perimeter of all groups of $\mathcal{A}$ cells. In the following lemmas we present 2-perimeter and perimeter "optimality tests" for any group of cells.

Lemma 9 *A group of cells has a minimum 2-perimeter if and only if it has a minimum semi-perimeter.*

Proof: Follows directly from lemma 4. ∎

Lemma 10 *A group of cells has a minimum perimeter if and only if it is slice-convex and has a minimum semi-perimeter.*

min semi-perimeter ($\mathcal{S}^*(\mathcal{A})$)	area ($\mathcal{A}$)
2	1
3	2
4	3 – 4
5	5 – 6
6	7 – 9
7	10 – 12
8	13 – 16
9	17 – 20
10	21 – 25
11	26 – 30
12	31 – 36
13	37 – 42
14	43 – 49
15	50 – 56

Table 1: Minimum perimeters

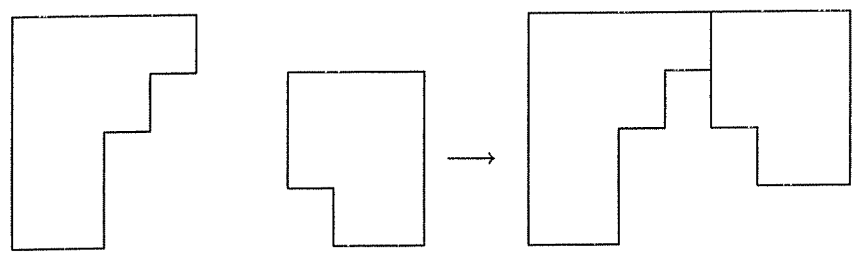


Figure 5: translating tiles to decrease perimeter

Proof: Slice-convexity and a minimum semi-perimeter are sufficient to guarantee a minimum perimeter by lemma 2. A minimum semi-perimeter is necessary because for any semi-perimeter $\mathcal{S}$ achievable with group size $\mathcal{A}$, there is a slice-convex group of size $\mathcal{A}$ with a perimeter of $2\mathcal{S}$. Slice-convexity is also necessary for a minimum-perimeter because the cells of a non-slice-convex group can be rearranged into a slice-convex group with a lower perimeter. ∎

It is also necessary for a group to consist of a single tile (*i.e.*, a single maximal connected set), in order for it to have a minimum perimeter. Consider a group of cells containing two disconnected tiles denoted by T_1 and T_2. By translating T_1 it is always possible to connect T_1 and T_2 (see figure 5), thereby decreasing the perimeter of the group of cells by at least 2.

We may classify tile shapes according to two independent characteristics. Tiles are either nearly square (dimensions differing by at most 1) or non-square. In addition, tiles are either regular (complete rectangles) or irregular. Figure 6 depicts examples of minimum-perimeter tiles in each of the four categories induced by these characteristics. The proof of theorem 6 implies a construction technique for "perimeter-optimal" tiles, *i.e.*, tiles with minimum perimeter. An optimal tile of any area $\mathcal{A}$ can be constructed by arranging $\mathcal{A}$ cells into a partial square as follows. Start with a complete square with sides of length $\lfloor \mathcal{A}^{1/2} \rfloor$. Add cells to

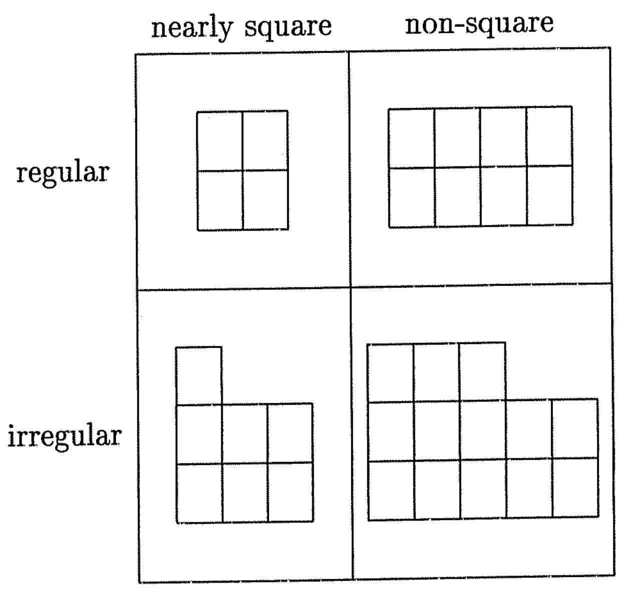


Figure 6: Categories of tiles

fill in new 1-dimensional faces (completing a face before starting on a new one) until the total number of cells is $\mathcal{A}$. The resulting partial square will have sides of length $\lfloor \mathcal{A}^{1/2} \rfloor$ and $\lceil \mathcal{A}^{1/2} \rceil$, and will measure $\lceil \mathcal{A}^{1/2} \rceil$ in as few dimensions as possible. By theorem 6, it will have minimum semi-perimeter. If the partial squares are constructed to be slice-convex then by lemma 10 they also have minimum perimeter. This construction technique is a special case of a technique described in §2.2.2 for constructing all minimum-perimeter tiles of a given area.

In figure 7, we show some perimeter-optimal partial squares constructed in this manner with areas ranging from one to sixteen.

Lemma 11 $\mathcal{P}^*(\mathcal{A}) = 2\mathcal{S}^*(\mathcal{A})$.

Proof: Follows from the fact that there is a minimum semi-perimeter slice-convex group of every size. ∎

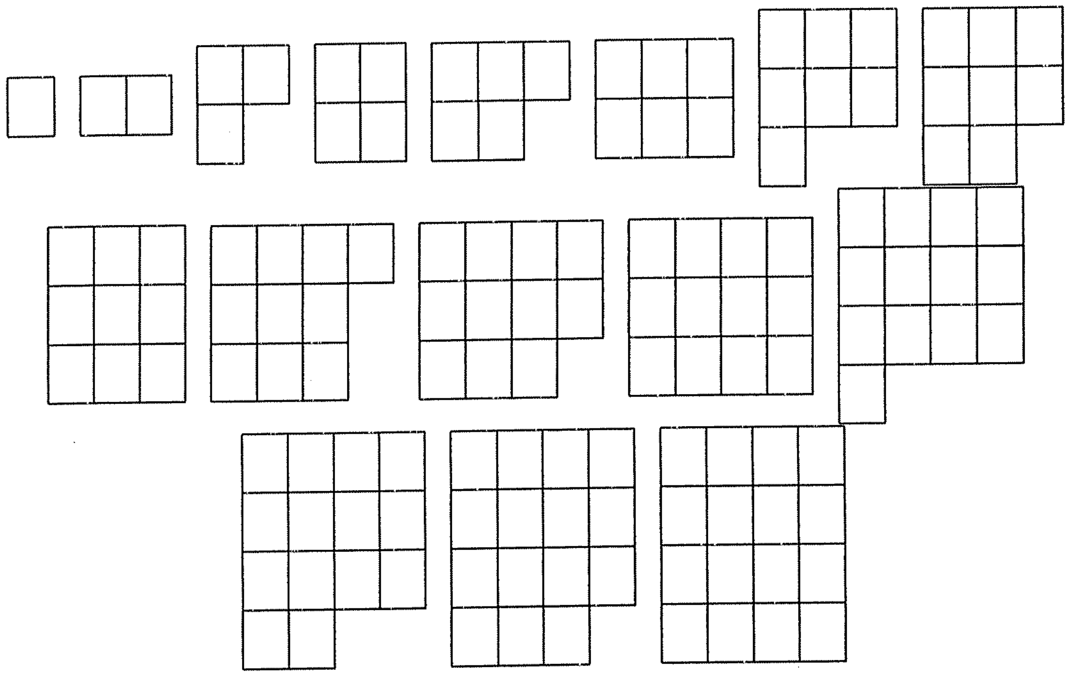


Figure 7: Partial squares with minimum perimeter

2.2.1 Optimal rectangles

Using the results from the previous section, we can characterize the rectangles which have minimum perimeter.

Theorem 12 *An $x \times (x+k)$ or an $(x+k) \times x$ rectangular block is perimeter-optimal iff*

$$k \text{ is even and } 1 + \tfrac{k}{2}(\tfrac{k}{2} - 1) \leq x$$
$$\text{or}$$
$$k \text{ is odd and } 1 + \left(\tfrac{k-1}{2}\right)^2 \leq x.$$

Conversely, an $x \times (x+k)$ or an $(x+k) \times x$ rectangle is perimeter-optimal iff the rectangularity increment k is at most

$$\max\left\{2\ round(x^{1/2}), 2\left\lfloor x^{1/2} - 1\right\rfloor + 1\right\}$$

where $round(x)$ rounds x to the nearest integer.

Proof: To prove the first part of the theorem, we simply apply the optimality test. By lemma 7, an $x \times (x+k)$ block is optimal iff

$$\left\lceil \frac{2x+k-1}{2} \right\rceil^r \left\lfloor \frac{2x+k-1}{2} \right\rfloor^{2-r} < x^2 + kx \qquad (3)$$

where $r = (2x + k - 1)\, mod\, 2$.

If k is even, (3) reduces to

$$\begin{array}{rrcl} & \left\lceil \frac{2x+k-1}{2} \right\rceil \left\lfloor \frac{2x+k-1}{2} \right\rfloor & < & x^2 + kx \\ \Longleftrightarrow & (x + \frac{k}{2})(x + \frac{k}{2} - 1) & < & x^2 + kx \\ \Longleftrightarrow & \frac{k}{2}(\frac{k}{2} - 1) & < & x. \end{array}$$

The integrality of both sides of the inequality allow us to derive the desired result.

If k is odd, (3) reduces to

$$\begin{array}{rrcl} & \left\lfloor \frac{2x+k-1}{2} \right\rfloor^2 & < & x^2+kx \\ \iff & \left(x+\frac{k-1}{2}\right)^2 & < & x^2+kx \\ \iff & \left(\frac{k-1}{2}\right)^2 & < & x. \end{array}$$

To prove the second part of the theorem, we show that $2\left\lfloor (x-1)^{1/2} \right\rfloor + 1$ and $2 \operatorname{round}\left(x^{1/2}\right)$ are the largest odd and even integers respectively satisfying (3).

To prove the result for the odd numbers, we start with the expression for x in terms of odd k.

$$\begin{array}{rrcl} & \left(\frac{k-1}{2}\right)^2+1 & \le & x \\ \iff & \frac{k-1}{2} & \le & (x-1)^{1/2}. \end{array}$$

Since the LHS of the last inequality is integer, we may take the floor of the RHS.

$$\begin{array}{rrcl} \iff & \frac{k-1}{2} & \le & \left\lfloor (x-1)^{1/2} \right\rfloor \\ \iff & k & \le & 2\left\lfloor (x-1)^{1/2} \right\rfloor + 1. \end{array}$$

Since the RHS of the last inequality is an odd integer, $k = 2\left\lfloor (x-1)^{1/2} \right\rfloor + 1$ is the largest odd integer satisfying (3).

To prove the result for the even numbers we write $x^{1/2}$ in the form $x^{1/2} = r + f$ where r is the integer part and $f \in [0,1)$ is the fractional part. If $f < \frac{1}{2}$ then $2 \operatorname{round}\left(x^{1/2}\right) = 2r$. If $f > \frac{1}{2}$ then $2 \operatorname{round}\left(x^{1/2}\right) = 2r+2$. (For integer x, f is never $\frac{1}{2}$ so $\operatorname{round}\left(x^{1/2}\right)$ is uniquely defined for integer x.)

If $f < \frac{1}{2}$ and $2 \operatorname{round}\left(x^{1/2}\right) = 2r$ then $k = 2r$ satisfies (3) because

$$\frac{k}{2}\left(\frac{k}{2}-1\right) = r(r-1) = r^2 - r < r^2 + 2fr + f^2 = x$$

and $k = 2r+2$ violates (3) because

$$\frac{k}{2}\left(\frac{k}{2}-1\right) + 1 = (r+1)r + 1 = r^2 + r + 1 > r^2 + 2fr + f^2 = x.$$

If $f > \frac{1}{2}$ and 2 round $\left(x^{1/2}\right) = 2r + 2$ then $k = 2r + 2$ satisfies (3) because

$$\frac{k}{2}\left(\frac{k}{2} - 1\right) = (r+1)r = r^2 + r < r^2 + 2fr + f^2 = x$$

and $k = 2r + 4$ violates (3) because

$$\frac{k}{2}\left(\frac{k}{2} - 1\right) = (r+2)(r+1) = r^2 + 3r + 2 > r^2 + 2fr + f^2 = x.$$

Therefore $k = 2$ round $\left(x^{1/2}\right)$ is the largest even integer satisfying (3). ∎

Note that the first part of the theorem shows that if a particular two-dimensional rectangle is optimal, then by increasing both dimensions by the same amount, the resulting larger rectangle is also optimal. Figure 8 shows the dimensions of all rectangles with $x \leq 30$ that have minimum perimeter. The integral points on the diagonal line in the figure represent the squares, and the outer boxes represent the most-skewed rectangles with optimal perimeter. All integer points between (and including) the boxed points correspond to rectangles with minimum perimeter. Table 2 lists dimensions of the most skewed optimal rectangles corresponding to the boxed points above the diagonal.

2.2.2 Optimal Irregular Tiles

In order to characterize irregular optimal groups of cells we prove a simple but powerful theorem giving another necessary and sufficient condition for the perimeter optimality of *any* group of cells.

Theorem 13 *A group of cells of $\mathcal{A}$ cells has minimum perimeter if and only if it is slice-convex and its minimum circumscribing rectangle has semi-perimeter $\mathcal{S}^*(\mathcal{A})$.*

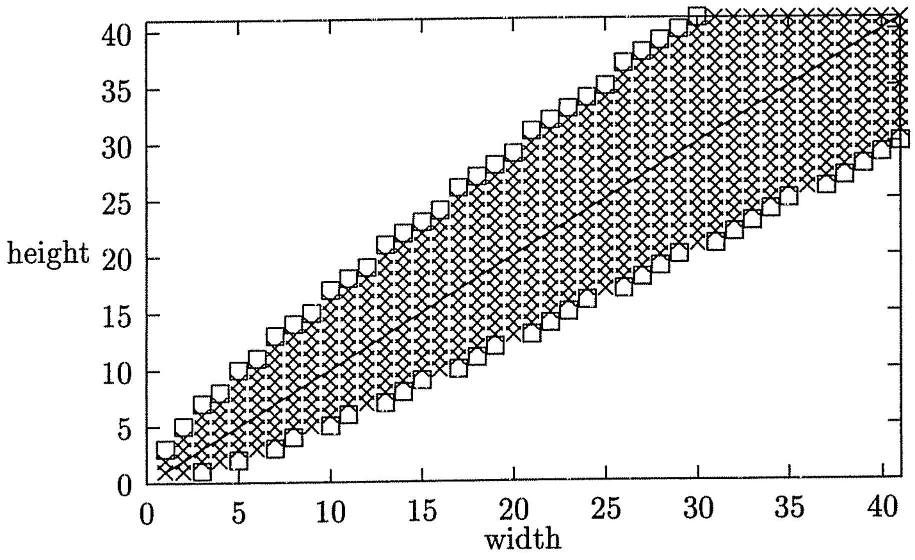


Figure 8: Dimensions of rectangles with optimal perimeter

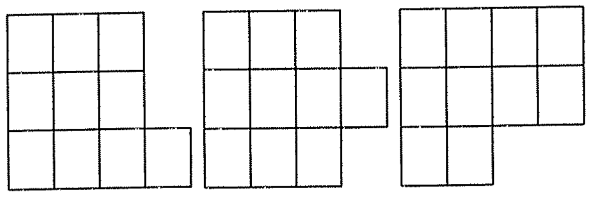
Proof: The theorem follows from the fact that a rectangle has the same perimeter as any slice-convex group of cells of cells it minimally circumscribes. ∎

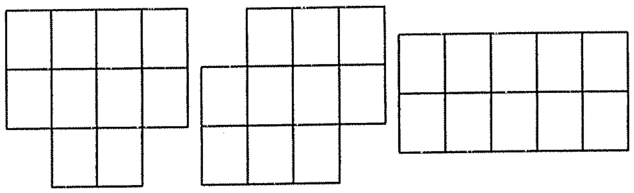
Theorem 13 suggests a way to find *all* the minimum 2-perimeter and minimum perimeter groups of cells for a given area $\mathcal{A}$. In this discussion we consider two groups of cells to be *equivalent* if one can be transformed into the other by rotation and/or reflection. Using the theorem, all the possible circumscribing rectangles for perimeter-optimal groups of cells of area $\mathcal{A}$ can be specified. For each of these rectangles, any subset of $\mathcal{A}$ cells forms a minimum 2-perimeter group, and any *slice-convex* subset of $\mathcal{A}$ cells forms a minimum-perimeter group of cells. We say a cell is a *corner cell* of a group if it is not between two group cells in any slice. By removing a corner cell from a slice-convex group, convexity is maintained. Therefore, starting from a rectangle, a minimum-perimeter group

k	Most-skewed perimeter-optimal rectangles $x \times (x+k)$				
2	1×3				
3	2×5				
4	3×7	4×8			
5	5×10	6×11			
6	7×13	8×14	9×15		
7	10×17	11×18	12×19		
8	13×21	14×22	15×23	16×24	
9	17×26	18×27	19×28	20×29	
10	21×31	22×32	23×33	24×34	25×35
11	26×37	27×38	28×39	29×40	30×41

Table 2: Some most-skewed perimeter-optimal rectangles

can be constructed by iteratively removing corner cells. For example, given an area of 10, the minimum-perimeter groups are generated as follows. $S^*(10) = 7$, so the rectangles of semi-perimeter 7 are considered. The possibilities are 1×6, 2×5, and 3×4. A 1×6 rectangle cannot enclose 10 cells, so all minimum-perimeter groups are circumscribed by either 2×5 or 3×4 rectangles. Therefore all minimum-perimeter groups of area 10 are equivalent to the following groups:





The first five groups above represent all the possibilities for groups enclosed by 3×4 rectangles: the first two are constructed by removing two corners from the same column, the next two by removing two corners from the same row, and the fifth by removing two corners from different rows and columns. There is only one possible group of area 10 contained in a 2×5 rectangle.

By examining the above technique, we are able to identify the cases in which there is a unique minimum-perimeter group of given size. Given an area $\mathcal{A}$, if there is a unique rectangle with semi-perimeter $\mathcal{S}^*(\mathcal{A})$ and area $\geq \mathcal{A}$, then all minimum-perimeter groups of size $\mathcal{A}$ are circumscribed by that rectangle. Furthermore, if the area of that unique enclosing rectangle is $\mathcal{A}$ or $\mathcal{A}+1$ then there is a unique optimal group of area $\mathcal{A}$, because all removals of either zero or one corner result in equivalent groups. We also show that all other cases lead to non-uniqueness.

Lemma 14 *There is a unique minimum-perimeter group of area $\mathcal{A}$ if and only if $\mathcal{A}$ can be expressed as $\mathcal{A} = k^2$, $k(k+1)$, or $k(k+1)-1$ for positive integer k.*

Proof: If $\mathcal{A} = k^2$ then $\mathcal{S}^*(\mathcal{A}) = 2k$. The unique enclosing rectangle with semi-perimeter $2k$ and area at least $\mathcal{A}$ has dimensions $k \times k$ (any other rectangle with semi-perimeter $2k$ has area less than k^2). Similarly, if $\mathcal{A} = k(k+1)$, or $k(k+1)-1$ and $k \geq 2$, then $\mathcal{S}^*(\mathcal{A}) = 2k+1$, and the unique rectangle with semi-perimeter $2k+1$ and area at least $\mathcal{A}$ has dimensions $k \times (k+1)$.

To show that these are the only classes of areas which have unique optimal groups, consider the fact that such an area $\mathcal{A}$ must have only one possible enclosing rectangle with perimeter $\mathcal{S}^*(\mathcal{A})$. Since $\mathcal{S}^*(\mathcal{A})$ corresponds to an enclosing square or quasi-square, it follows that the unique enclosing rectangle must be this square or quasi-square. This means that the area $\mathcal{A}$ is expressible as either $k^2 - j$ or $k(k+1) - j$ for positive integer k and non-negative integer j. For areas expressed as $\mathcal{A} = k^2 - j$ with $j \geq 1$, a $(k+1) \times (k-1)$ rectangle is a second enclosing rectangle with semi-perimeter $2k$ because $(k+1)(k-1) \geq k^2 - j$. For areas expressed as $k(k+1) - j$ with $j > 2$ there can not be a unique optimal group because there are at least two ways of removing j corners from a $k \times (k+1)$ rectangle. ∎

The preceding argument proves that squares, quasi-squares, and quasi-squares minus one corner are the unique optimal groups for their areas and that all other areas have alternate optimal groups. Similar reasoning can be used to show that for areas of the form $k^2 - 1$ with $k \geq 2$, there are exactly two optimal groups: a $k \times k$ square with one corner removed and a complete $(k-1) \times (k+1)$ rectangle.

Another interesting fact about the set of optimal groups of a given area is that there can be at most one rectangular group in the set. To prove this we make use of the following lemma.

Lemma 15 *For (unordered) pairs of positive numbers, two distinct pairs with identical pair sums have different pair products,* i.e., *for positive numbers x_1, y_1, x_2, y_2, if $x_1 + y_1 = x_2 + y_2$ and $\{x_1, y_1\} \neq \{x_2, y_2\}$ then $x_1 y_1 \neq x_2 y_2$.*

Proof: Write x_2 as $x_1 + k$. Then $y_2 = y_1 - k$. Now assume $x_1 y_1 = x_2 y_2$. Substituting into this last equation, we get

$$x_1 y_1 = (x_1 + k)(y_1 - k).$$

Simplifying, we get

$$k(y_1 - x_1 - k) = 0,$$

in other words $k = 0$ or $k = y_1 - x_1$. In the first case $x_1 = x_2$ and $y_1 = y_2$, and in the second case $x_1 = y_2$ and $y_1 = x_2$, a contradiction. ∎

The lemma tells us that all rectangles with a given semi-perimeter have different areas and implies that if a rectangular group of area $\mathcal{A}$ is optimal then it is the only optimal rectangular group with area $\mathcal{A}$ (any other optimal groups will be irregular). Note that this is not true in three or more dimensions.

2.3 Optimal Tilings

The results of the previous sections provide an implicit lower bound for both the minimum perimeter and minimum diversity problems.

Theorem 16 *Given either a perimeter minimization problem or a diversity minimization problem in two dimensions and a specified number of cells $\mathcal{A}_p$ for each of the processors, a lower bound on half the total perimeter or on the total slice diversity is*

$$\sum_p \mathcal{S}^*(\mathcal{A}_p).$$

Proof: Follows immediately from theorem 6 and lemma 11. ∎

It should be noted here that several methods for graph partitioning use spectral information about the graph's Laplacian matrix and the theory behind the methods yields lower bounds on the number of cut edges as a byproduct. The bounds, however, are based on continuous relaxations of integer problems and are therefore of poor quality. Our bounds, on the other hand, are tight for large classes of problems as will be demonstrated here.

In order to achieve the lower bound on total perimeter, each group must have a perimeter of exactly $\mathcal{P}^*(\mathcal{A}_p)$. Thus, we wish to interleave perimeter-optimal groups for all processors in order to fill the domain exactly. In order to achieve the lower bound on total slice diversity, each group must have a 2-perimeter of exactly $\mathcal{S}^*(\mathcal{A}_p)$. Thus, we wish to interleave minimum 2-perimeter groups for all processors in order to fill the domain exactly. In this section we exhibit classes of domains for which such partitions can be constructed.

If a grid is tilable with minimum-perimeter tiles, then both the total perimeter and total diversity are minimized by the tiling. Some grids may not be tilable with minimum perimeter tiles, *i.e.*, the lower bounds on perimeter are not achievable, but partitions exist where the groups each have a minimum 2-perimeter. In such cases, different partitions minimize total diversity and total perimeter respectively. For example, the 5×5 grid is partition-able into 5 equal size minimum 2-perimeter groups, but not into 5 equal size minimum perimeter groups. Figure 9 illustrates a minimum diversity partition and a minimum perimeter partition. In the partition on the left, each group has an optimal 2-perimeter of 5 for a total diversity of 25. However, the perimeters are not minimal – several groups consist of two tiles; the total perimeter is 60. In the partition on the right, groups 1,2,3, and 4 have a minimum perimeter of 10, and group 5 has a perimeter of 12, for a total of 52. The 2-perimeters are optimal except for group 5 which has a 2-perimeter of 6, for a total diversity of 26. To see why the lower bound of 50 for the total perimeter is not achievable, notice that an optimal tile of area five must be made up of a 2×2 block plus one more cell. Since it is impossible to fit five 2×2 blocks into a 5×5 grid, the lower bound cannot be achievable. Since we have provided a partition which achieves a total perimeter of 52, this must be optimal value. Thus, we have and example of a case where the minimum total perimeter is greater than the minimum total diversity.

1	4	4	5	5
1	1	2	5	5
1	1	2	2	3
3	4	2	2	3
3	4	4	5	3

1	1	4	4	4
1	1	5	4	4
1	5	5	5	3
2	2	5	3	3
2	2	2	3	3

Figure 9: Minimum diversity and minimum perimeter partitions

2.3.1 Minimum-perimeter tilings

Corollary 17 $\sum_p \mathcal{P}^*(\mathcal{A}_p) \leq \sum_g \mathcal{P}_g$.

Proof: Use theorem 6 and the fact that $\sum_g \mathcal{P}_g = \sum_p \mathcal{P}(T_p)$. ∎

Clearly, if the tile for each processor has minimum perimeter (*i.e.*, $\mathcal{P}(\mathcal{A}_p) = \mathcal{P}^*(\mathcal{A}_p)$ for all p), then the corresponding set of cell assignments achieves the lower bound on the communication measure $\sum_g \mathcal{P}_g$, and is therefore an optimal assignment.

In §2.3 we give classes of domains for which such tilings are possible.

Optimal Tilings with Rectangles

One class of problems that have easily obtainable optimal solutions are instances in which the domain is a $M_1 \times M_2$ rectangular grid that can be tiled with perimeter-optimal rectangles. In particular, if N can be factored as $f_1 f_2$ where f_1 divides M_1, f_2 divides M_2 and $\frac{M_1}{f_1} \times \frac{M_2}{f_2}$ rectangles are perimeter-optimal, then such a tiling is possible when all processors have equal loads. Below we demonstrate an optimal assignment for such an instance: a 6×18 grid with 6 processors, each of which has a load of 18.

1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3
1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3
1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3
4	4	4	4	4	4	5	5	5	5	5	5	6	6	6	6	6	6
4	4	4	4	4	4	5	5	5	5	5	5	6	6	6	6	6	6
4	4	4	4	4	4	5	5	5	5	5	5	6	6	6	6	6	6

However, it is not necessary that all the tiles be oriented in the same way. An alternate optimal assignment for the same problem is shown below.

1	1	1	2	2	2	2	2	2	3	3	3	4	4	4	4	4	4
1	1	1	2	2	2	2	2	2	3	3	3	4	4	4	4	4	4
1	1	1	2	2	2	2	2	2	3	3	3	4	4	4	4	4	4
1	1	1	5	5	5	5	5	5	3	3	3	6	6	6	6	6	6
1	1	1	5	5	5	5	5	5	3	3	3	6	6	6	6	6	6
1	1	1	5	5	5	5	5	5	3	3	3	6	6	6	6	6	6

Figure 1 in chapter 1 is an example of a non-rectangular domain optimally tiled with rectangles.

Optimal Tiling with Irregular Tiles

Irregular tiles can fit together to tile many grids. The example below shows how irregular optimal tiles of area 10 fit nicely together.

1	1	1	1	4	4	4	4	5	5	6	6	6
1	1	1	1	4	4	4	4	5	5	6	6	6
1	1	2	2	3	3	4	4	5	5	5	6	6
2	2	2	2	3	3	3	3	5	5	5	6	6
2	2	2	2	3	3	3	3	8	8	8	7	7
								8	8	8	7	7
								8	8	7	7	7
								8	8	7	7	7

This example also demonstrates the technique of optimally tiling by decomposing the domain into subdomains which can each be optimally tiled by a proportional subset of the processors. The domain above can be split into four 5×4 rectangles, each of which can be tiled optimally with two processors. Such a divide and conquer approach to tiling is a method of constructing optimal tilings for large domains with complex shapes.

2.3.2 Minimum diversity toroidal tiling

In this section we present an optimal solution technique for the diversity minimization problem on a certain class of grids. These results are useful because they provide a class of problems for which the optimal values are known, making this class a good source of test problems for heuristics. The technique described here was inspired by a method due to Schultz [10] which produces minimum perimeter solutions for $N \times N$ grids, *i.e.*, problems where the grid is square and the number of rows equals the number of processors. A description of Schultz's method is presented in appendix C. The new method is superior since it is proven to work for a larger class of problems and is more intuitive. It differs from the previous method by assigning all the cells for one processors at every step instead of assigning the cells along a diagonal to different processors. This aspect may allow it to be modified to produce a heuristic for the minimum perimeter problem.

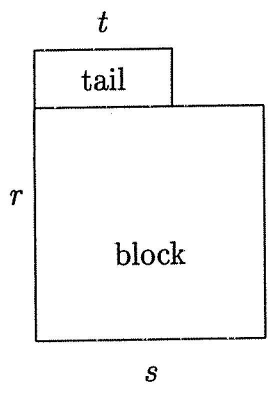
Since the groups of cells do not have to be connected in order to have minimum 2-perimeter, one technique for creating optimal partitions is to allow the tiles to "wrap around" the top or side of the grid. We can therefore think of the domain as lying on the surface of a torus, *i.e.*, the top row of the grid is adjacent to the bottom row, and the left-most column is adjacent to the right-most. The perimeter of group g as measured on a torus, denoted by $\mathcal{P}_t(g)$, is in between $\mathcal{S}(g)$ and $\mathcal{P}_2(g)$. The lower bound $\sum_g \mathcal{S}^*(\mathcal{A}_g)$ is a common lower bound on total perimeter, total toroidal perimeter, and total 2-perimeter. For any partition of grid cells into groups, the following inequalities hold:

$$\sum_g \mathcal{S}^*(\mathcal{A}_g) \leq \sum_g \mathcal{P}_2(g) \leq \sum_g \mathcal{P}_t(g) \leq \frac{1}{2} \sum_g \mathcal{P}(g)$$

If $\frac{1}{2}\sum_g \mathcal{P}(g)$ is equal to the lower bound, then $\sum_g \mathcal{S}^*(\mathcal{A}_g) = \sum_g \mathcal{P}_2(g) = \frac{1}{2}\sum_g \mathcal{P}(g)$ and the partition is optimal with respect to all three measures.

So we are interested in tiling toroidal domains with contiguous (in a toroidal sense) minimum semi-perimeter tiles. Tilings of this form provide optimal solutions to the diversity minimization problem.

Consider a $N \times N$ toroidal domain, with $\mathcal{A}_p = N$, $(p = 1, 2, \ldots, N)$. Under these constraints it is always possible to tile the domain with minimum-perimeter tiles. Let $r = \lfloor \sqrt{N} \rfloor$, $s = \max\{k | rk \leq N\}$ and $N = rs + t$, $0 \leq t < s$. Note that $s - r$ is either 0 or 1. Consider the partial square tile of area N (having minimum perimeter), made up of an $r \times s$ block and a "tail" of length t.

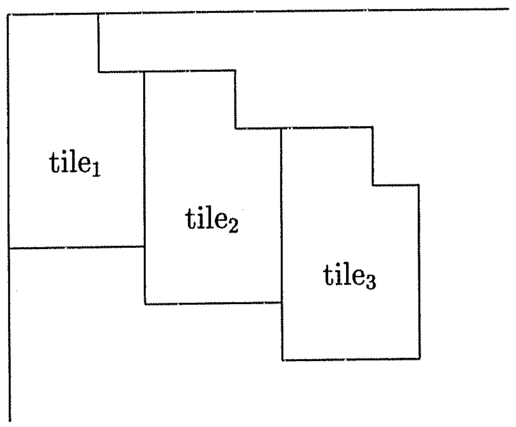


We describe how to tile the grid with these partial squares. If the tail is of length 0, then the grid can be tiled in the regular way by $r \times s$ rectangular tiles. (This is true because $N = rs + 0$.)

If the tail length is not 0, then diagonal tiling (with wrap-around) partitions the grid into tiles as follows. The first tile in the grid's upper left corner is made up of the set of cells:

$$\begin{aligned} \text{tile}_1 = \{ \; & (1,1),(1,2),\ldots,(1,t), \\ & (2,1),(2,2),\ldots,(2,s), \\ & \qquad\quad \vdots \\ & (r+1,1),(r+1,2),\ldots,(r+1,s)\} \; . \end{aligned}$$

The coordinates of the cells in the $(i+1)$st tile are obtained by adding i to all the row coordinates and is to all the column coordinates of the cells in tile_1 modulo N, as shown below.



Theorem 18 *The $\mathcal{A}$ tiles tile$_1$, tile$_2$, ..., tile$_{\mathcal{A}}$ cover the grid completely with no overlap.*

Proof: Since each tile contains $\mathcal{A}$ cells and the grid has $\mathcal{A}^2$ cells, it suffices to prove that each each grid cell belongs to at least one of tile$_1$, tile$_2$,..., tile$_{\mathcal{A}}$, *i.e.*, for any grid cell (x, y)

$$x \equiv x_1 + i \ \ (mod\,\mathcal{A})$$

and

$$y \equiv y_1 + is \ \ (mod\,\mathcal{A})$$

for some $0 \leq i \leq \mathcal{A}-1$, and some $(x_1, y_1) \in$ tile$_1$. Assume by way of contradiction that there exists some grid cell (x, y) such that for any $(x_1, y_1) \in$ tile$_1$

$$x \not\equiv x_1 + i \ \ (mod\,\mathcal{A})$$

or

$$y \not\equiv y_1 + is \ \ (mod\,\mathcal{A})$$

for $i = 0, 1, \ldots, \mathcal{A} - 1$. For each $(x_1, y_1) \in$ tile$_1$, $x \equiv x_1 + (x - x_1)$ $(\mathcal{A})$ so our assumption forces the incongruence

$$y \not\equiv y_1 + (x - x_1)s \ \ (mod\,\mathcal{A}).$$

Therefore the following set of $\mathcal{A}$ incongruences (one for each cell in tile$_1$) must hold:

$$\begin{aligned}
y &\not\equiv t + (x-1)s \ (mod\, \mathcal{A}) \\
y &\not\equiv t - 1 + (x-1)s \ (mod\, \mathcal{A}) \\
&\vdots \\
y &\not\equiv 1 + (x-1)s \ (mod\, \mathcal{A}) \\
\\
y &\not\equiv s + (x-2)s \ (mod\, \mathcal{A}) \\
y &\not\equiv s - 1 + (x-2)s \ (mod\, \mathcal{A}) \\
&\vdots \\
y &\not\equiv 1 + (x-2)s \ (mod\, \mathcal{A}) \\
\\
&\vdots \\
\\
y &\not\equiv s + (x-r-1)s \ (mod\, \mathcal{A}) \\
y &\not\equiv s - 1 + (x-r-1)s \ (mod\, \mathcal{A}) \\
&\vdots \\
y &\not\equiv 1 + (x-r-1)s \ (mod\, \mathcal{A}).
\end{aligned}$$

The right hand sides of the above incongruences form a sequence of $\mathcal{A}$ consecutive integers.

Thus, we have $y \not\equiv i \ (mod\, \mathcal{A})$, $i = 1, 2, \ldots, \mathcal{A}$, a contradiction. ∎

Figure 2.3.2 illustrates diagonal tiling on a 7×7 grid with 7 processors.

The result of theorem 18 can be generalized to rectangular grids where the group size divides both dimensions.

Theorem 19 *A toroidal domain can be tiled with partial square tiles of size $\mathcal{A}$ if $\mathcal{A}$ divides both the number of rows and columns evenly.*

1	6	6	6	7	7	7
1	1	1	2	7	7	7
1	1	1	2	2	2	3
3	3	4	2	2	2	3
3	3	4	4	4	5	3
5	6	4	4	4	5	5
5	6	6	6	7	5	5

Figure 10: Diagonal tiling of a square toroidal domain

Proof: Let the dimensions of the domain be $I\mathcal{A} \times J\mathcal{A}$. Let $r = \lfloor\sqrt{\mathcal{A}}\rfloor$, and $s = \max\{k|rk \leq \mathcal{A}\}$, so that $\mathcal{A} = rs + t$, $0 \leq t < s$.

Consider the partial square tile of area $\mathcal{A}$ (having minimum perimeter), made up of an $r \times s$ block and a "tail" of length t. Diagonal tiling (with wrap-around) partitions the grid into tiles by creating J diagonal strips of $I\mathcal{A}$ tiles each as follows. Tile$_{i,j}$ denotes the ith tile in the jth diagonal strip. The first tile in the $j+1$st diagonal strip abuts the top row of the grid and consists of the set of cells

$$\begin{aligned}\text{tile}_{1,j} = \{ \quad & (1, j\mathcal{A}+1), (1, j\mathcal{A}+2), \ldots, (1, j\mathcal{A}+t), \\ & (2, j\mathcal{A}+1), (2, j\mathcal{A}+2), \ldots, (2, j\mathcal{A}+s), \\ & \vdots \\ & (r+1, j\mathcal{A}+1), (r+1, j\mathcal{A}+2), \ldots, (r+1, j\mathcal{A}+s)\} \ .\end{aligned}$$

The $i+1$st tile in a strip is obtained by adding $i \bmod I\mathcal{A}$ to the row coordinate and adding $is \bmod J\mathcal{A}$ to the column coordinate of each cell in the first tile of the strip.

To prove that the procedure described above is a tiling of the domain, it suffices to show that each cell of the domain is contained in one of the $IJ\mathcal{A}$ tiles, *i.e.*, for any grid cell (x, y)

$$x \equiv x_{1,j} + i \pmod{I\mathcal{A}}$$

and

$$y \equiv y_{1,j} + is \pmod{J\mathcal{A}}$$

for some $0 \leq i < I\mathcal{A}$, some $0 \leq j < J$, and some $(x_{1,j}, y_{1,j})$ in tile$_{1,j}$.

Assume, by way of contradiction, that there exists some cell (x, y) that is not contained in any tile. For each $(x_{1,j}, y_{1,j})$ in tile$_{1,j}$, a cell in row x can be reached by moving ahead $d(x, x_1)$ tiles in the strip where

$$d(x, x_1) = \begin{cases} x - x_{1,j} \text{ if } x - x_{1,j} \geq 0 \\ I\mathcal{A} + x - x_{1,j} \text{ otherwise,} \end{cases}$$

i.e., $x \equiv x_{1,j} + d(x, x_{1,j}) \pmod{I\mathcal{A}}$. Therefore our assumption forces the incongruence

$$y \not\equiv y_{1,j} + d(x, x_{1,j})s \pmod{J\mathcal{A}}.$$

Letting $(x_{1,j}, y_{1,j})$ take on the coordinate values of cells in tile$_{1,j}$ for $0 \leq j < J$, our assumption forces the following set of incongruences:

$$\text{tail}\begin{cases} y \not\equiv & j\mathcal{A} + t + d(x,\,1)s \qquad (mod\, J\mathcal{A}) \\ y \not\equiv & j\mathcal{A} + t - 1 + d(x,\,1)s \quad (mod\, J\mathcal{A}) \\ & \vdots \\ y \not\equiv & j\mathcal{A} + 1 + d(x,\,1)s \qquad (mod\, J\mathcal{A}) \end{cases}$$

$$\text{row 1}\begin{cases} y \not\equiv & j\mathcal{A} + s + d(x,\,2)s \qquad (mod\, J\mathcal{A}) \\ y \not\equiv & j\mathcal{A} + s - 1 + d(x,\,2)s \quad (mod\, J\mathcal{A}) \\ & \vdots \\ y \not\equiv & j\mathcal{A} + 1 + d(x,\,2)s \qquad (mod\, J\mathcal{A}) \end{cases}$$

$$\vdots$$

$$\text{row r}\begin{cases} y \not\equiv & j\mathcal{A} + s + d(x,\,r+1)s \qquad (mod\, J\mathcal{A}) \\ y \not\equiv & j\mathcal{A} + s - 1 + d(x,\,r+1)s \quad (mod\, J\mathcal{A}) \\ & \vdots \\ y \not\equiv & j\mathcal{A} + 1 + d(x,\,r+1)s \qquad (mod\, J\mathcal{A}) \end{cases}$$

For a fixed value of j, the $\mathcal{A}$ right hand side values of the above incongruences are unique modulo $\mathcal{A}$, therefore when j varies, all the $J\mathcal{A}$ right hand side values are unique modulo $J\mathcal{A}$. One of the incongruences must therefore not hold, contradicting our assumption and finishing the proof. ∎

Figure 11 illustrates the diagonal tiling technique on a non-square grid. The grid cells are partitioned into 15 groups of size 5.

1	9	9	10	10	11	4	4	5	5	6	14	14	15	15
1	1	2	10	10	11	11	12	5	5	6	6	7	15	15
1	1	2	2	3	11	11	12	12	13	6	6	7	7	8
8	9	2	2	3	3	4	12	12	13	13	14	7	7	8
8	9	9	10	3	3	4	4	5	13	13	14	14	15	8

Figure 11: Diagonal tiling of a rectangular toroidal domain

2.4 Lower Bounds as Functions of Problem Size

The lower bounds for the diversity minimization problem have some unusual properties as functions of the problem size. Consider the class of diversity minimization problems where all the specified group sizes are either the floor or the ceiling of the number of grid cells divided by N. Given a fixed grid, as N increases, the lower bound on diversity generally increases. This general trend can be observed by considering the extreme cases where $N = 1$ in which case each slice has a diversity of one, and $N =$ the number of grid cells in which case each slice has a diversity equal to the number of cells in it. Contrary to what might be expected, however, the lower bound (and in fact the minimal diversity) is not monotonic as a function of N. Consider the example of a 9×9 grid with $N = 8$ and $N = 9$ illustrated in figure 12. For the case of $N = 8$, 7 of the groups will have a size of 10 and one group will have a size of 11, yielding a lower bound on diversity of $7 \cdot 7 + 7 = 56$. For the case of $N = 9$, all groups will have a size of 9 and therefore the lower bound will be $9 \cdot 6 = 54$. In the figure, optimal solutions for the two problems which meet the lower bound are given. Therefore, adding another processor to the problem can actually decrease the total diversity.

1	1	1	2	2	2	1	7	7
1	1	1	2	2	2	2	7	7
1	1	1	2	2	2	8	8	8
3	3	3	4	4	4	3	7	7
3	3	3	4	4	4	4	7	7
3	3	3	4	4	4	8	8	8
5	5	5	6	6	6	5	7	7
5	5	5	6	6	6	6	8	8
5	5	5	6	6	6	8	8	8

1	1	1	2	2	2	3	3	3
1	1	1	2	2	2	3	3	3
1	1	1	2	2	2	3	3	3
4	4	4	5	5	5	6	6	6
4	4	4	5	5	5	6	6	6
4	4	4	5	5	5	6	6	6
7	7	7	8	8	8	9	9	9
7	7	7	8	8	8	9	9	9
7	7	7	8	8	8	9	9	9

Figure 12: Optimal solutions for $N = 8$ and $N = 9$

2.5 Non-uniform Grids

Under certain assumptions, the tiling approaches discussed above can also be used to deal with non-uniform grids which are important in many applications (see *e.g.*, Gropp and Keyes [13]). Considering the case shown in figure 13, suppose that that N processors are available, that the amount of computation per cell (regardless of cell size) is uniform, that total computing loads are to be nearly balanced among processors, and, finally, to ensure relative simplicity and uniformity of the computational procedure implemented on each processor, we impose the constraint that the cells assigned to each processor be of uniform size (i.e., that the tile associated with a processor lie entirely in subdomain D_1 or D_2). In order to first set up load balancing constraints, we would then partition the given number of processors between the two domains so that the number of cells per processor (rather than the area spanned by the processor) was approximately equal. In the second stage of this process, communication would be independently minimized

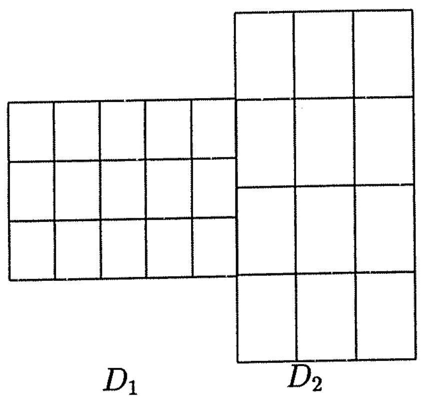


Figure 13: A non-uniform grid

for each subdomain by tiling (using its pre-determined appropriate number of processors) according to the procedures described above for minimum-perimeter tilings. The shared boundaries of the subdomains, across which communication would be required, would be automatically included in the objective function via the boundaries of the individual subdomains. Of course, this two-stage approach is easily generalized to provide a decomposition of arbitrary unions of subdomains into single subdomain problems of the type previously considered.

Chapter 3

Many-Dimensional Theory

The bounds on semi-perimeter generalize to many dimensions yielding bounds on multi-dimensional diversity minimization and on surface area minimization, the three-dimensional analog of perimeter minimization.

3.1 Lower bounds on d-perimeter

Theorems 5 and 6 and lemma 7 from chapter 2 are generalized here for multiple dimensions.

Let $\mathcal{A}_d^*(\mathcal{P})$ be the function mapping d-perimeter $\mathcal{P}$ to the maximum size of a configuration with d-perimeter $\mathcal{P}$.

Theorem 20 *Given d-perimeter $\mathcal{P}$, the compacted form $C(g)$ of any maximum size configuration with d-perimeter $\mathcal{P}$ is a d-dimensional hypercube with sides of length $\mathcal{P}/d$ if $d|\mathcal{P}$, and is a hyper-rectangle with sides of length $\lceil d/\mathcal{P}\rceil$ in r dimensions and length $\lfloor d/\mathcal{P}\rfloor$ in $d-r$ dimensions (where $r = \mathcal{P} \bmod d$) otherwise,* i.e.,

$$\mathcal{A}_d^*(\mathcal{P}) = \left\lceil \frac{\mathcal{P}}{d} \right\rceil^r \left\lfloor \frac{\mathcal{P}}{d} \right\rfloor^{d-r}.$$

Proof: Let $\mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^d$ denote the dimensions of a compacted configuration. The maximum load for which d-perimeter $\mathcal{P}$ is achievable is the optimal value of the following problem.

$$\begin{aligned} \mathcal{A}_d^*(\mathcal{P}) = \quad & \max \quad \textstyle\prod_d \mathcal{P}^d \\ & \text{s.t.} \quad \textstyle\sum_d \mathcal{P}^d = \mathcal{P} \\ & \qquad\quad \mathcal{P}^d \text{ a positive integer } \forall d. \end{aligned} \tag{4}$$

A necessary optimality condition for (4) is that no two $\mathcal{P}^d$'s differ by more than 1. Furthermore, only one (unordered) set of $\mathcal{P}^d$'s satisfy this condition:

$$\left\{ \underbrace{\left\lceil \frac{\mathcal{P}}{d} \right\rceil, \left\lceil \frac{\mathcal{P}}{d} \right\rceil, \ldots, \left\lceil \frac{\mathcal{P}}{d} \right\rceil}_{r}, \ldots, \underbrace{\left\lfloor \frac{\mathcal{P}}{d} \right\rfloor, \left\lfloor \frac{\mathcal{P}}{d} \right\rfloor, \ldots, \left\lfloor \frac{\mathcal{P}}{d} \right\rfloor}_{d-r} \right\}.$$

∎

By "inverting" the function $\mathcal{A}_d^*(\mathcal{P})$, we obtain a function $\mathcal{P}_d^*(\mathcal{A})$ which maps load $\mathcal{A}$ to the smallest d-perimeter achievable by a configuration of $\mathcal{A}$ cells.

Theorem 21 *The minimum d-perimeter of all configurations of $\mathcal{A}$ cells is*

$$\mathcal{P}_d^*(\mathcal{A}) := s \left\lceil \mathcal{A}^{1/d} \right\rceil + (d-s) \left\lfloor \mathcal{A}^{1/d} \right\rfloor$$

where s is the smallest positive integer such that

$$\left\lceil \mathcal{A}^{1/d} \right\rceil^s \left\lfloor \mathcal{A}^{1/d} \right\rfloor^{d-s} \geq \mathcal{A}.$$

Furthermore, the lower bound on $\mathcal{P}$ is tight.

Proof: We may bound the d-perimeter of any configuration of $\mathcal{A}$ cells from below by finding the smallest d-perimeter $\mathcal{P}$ that satisfies $\mathcal{A}^*(\mathcal{P}) \geq \mathcal{A}$, since this implies $\mathcal{A} > \mathcal{A}^*(\mathcal{P}-1)$, which means that a d-perimeter of $\mathcal{P}-1$ is not achievable for a load of $\mathcal{A}$.

Consider the following sequence of d-dimensional rectangular blocks.

$$\begin{array}{ll}
Q_0 : & 0 \times 0 \times \ldots \times 0 \\
Q_1 : & 1 \times 0 \times \ldots \times 0 \\
 & \vdots \\
Q_d : & 1 \times 1 \times \ldots \times 1 \\
Q_{d+1} : & 2 \times 1 \times \ldots \times 1 \\
Q_{d+2} : & 2 \times 2 \times \ldots \times 1 \\
 & \vdots \\
Q_{2d} : & 2 \times 2 \times \ldots \times 2 \\
Q_{2d+1} : & 3 \times 2 \times \ldots \times 2 \\
 & \vdots
\end{array}$$

We call these blocks "quasi-hypercubes" since the lengths of any two sides of a block differ by at most 1. Note that the volumes of the quasi-hypercubes in the sequence are strictly increasing after the dth, the volume of the ith quasi-hypercube Q_i for $i \geq d$ is $\mathcal{A}^*(i)$, and the d-perimeter for the quasi-hypercubes increases by 1 at each step. The volumes of these quasi-hypercubes are the points at which the lower bound on the d-perimeter increases by 1.

For an arbitrary $\mathcal{A}$, there is a unique smallest quasi-hypercube Q_j whose volume is at least $\mathcal{A}$. Since the volume of Q_j is at least $\mathcal{A}$, by selecting $\mathcal{A}$ cells from Q_j a d-perimeter of at most $\mathcal{P}(Q_j)$ is achievable for $\mathcal{A}$. Since the volume of Q_{j-1} is smaller than $\mathcal{A}$, a d-perimeter of $\mathcal{P}(Q_{j-1}) = \mathcal{P}(Q_j) - 1$ is not achievable for $\mathcal{A}$. Therefore the smallest d-perimeter achievable for any configuration of $\mathcal{A}$ cells is

$\mathcal{P}(Q_j)$. Each dimension of Q_j is either $\lfloor \mathcal{A}^{1/d} \rfloor$ or $\lceil \mathcal{A}^{1/d} \rceil$ so $\mathcal{P}(Q_j)$ is exactly the d-perimeter bound in the statement of the theorem. ∎

Lemma 22 *A d-dimensional configuration of $\mathcal{A}$ cells with d-perimeter $\mathcal{P}$ has minimum d-perimeter iff*

$$\mathcal{A}^*(\mathcal{P} - 1) < \mathcal{A} . \tag{5}$$

Proof: Same as the proof of lemma 7. ∎

It is interesting to note that the closed form version of the lower bound in theorem 8 does not generalize to multiple dimensions as the next theorem points out.

Theorem 23 $\lceil d\mathcal{A}^{1/d} \rceil \leq \mathcal{P}_d^*(\mathcal{A})$ i.e., $\lceil d\mathcal{A}^{1/d} \rceil$ *is a lower bound on d-perimeter but for $d > 2$ is not tight.*

Proof: To see that $\lceil d\mathcal{A}^{1/d} \rceil$ is a lower bound on the d-perimeter of a configuration of $\mathcal{A}$ cells, consider any configuration g of $\mathcal{A}$ cells intersecting $\mathcal{P}_i$ cells in dimension i, so that the d-perimeter of g is $\sum_{i=1}^{d} \mathcal{P}_i$. The compacted form $\mathcal{C}(g)$ is bounded by a d-dimensional box of volume $\prod_{i=1}^{d} \mathcal{P}_i$. Therefore

$$\mathcal{A} \leq \prod_{i=1}^{d} \mathcal{P}_i$$

Taking dth roots gives

$$\mathcal{A}^{1/d} \leq \left(\prod_{i=1}^{d} \mathcal{P}_i \right)^{1/d}$$

Because the arithmetic mean dominates the geometric mean (see Hardy *et al*[14])

$$\mathcal{A}^{1/d} \leq \left(\prod_{i=1}^{d} \mathcal{P}_i \right)^{1/d} \leq \frac{1}{d} \sum_{i=1}^{d} \mathcal{P}_i$$

whence

$$d\mathcal{A}^{1/d} \leq \sum_{i=1}^{d} \mathcal{P}_i.$$

Since the right-hand-side of the last inequality is an integer, we may take the ceiling of the left-hand-side to obtain the bound.

To see that $\lceil d\mathcal{A}^{1/d} \rceil$ is not a *tight* lower bound, consider $d = 3$ and $\mathcal{A} = 37$ where $\lceil 3 \cdot 37^{1/3} \rceil = 10$ but from theorem 21 $\mathcal{P}_3^*(37) = 1 \cdot 3 + 2 \cdot 4 = 11$ (this is the smallest d and smallest $\mathcal{A}$ for which the bound is not tight). ∎

Notice that the bound $\lceil d\mathcal{A}^{1/d} \rceil$ is tight infinitely often: whenever $\mathcal{A} = k^d$ for some positive integer k, the two lower bounds coincide.

3.2 Lower bound on surface area

Rather than minimizing perimeter in three dimensions, a related but different problem is obtained by minimizing surface area. Specifically, consider the three-dimensional problem of partitioning a three-dimensional rectilinear domain into a specified number of configurations of *specified volume* so as to minimize the total *surface area* of the configurations. This models minimization of interprocessor communication subject to load balancing constraints for three-dimensional domain decomposition on parallel systems. The bound on semi-perimeter from chapter 2 is useful in obtaining a surface-area bound.

Theorem 24 *The smallest possible surface area for a configuration of $\mathcal{A}$ cells is*

$$\begin{array}{rl} \min_s & 2\left(\max_i \{s_i\} + \sum_i \mathcal{S}^*(s_i)\right) \\ \textit{such that} & \textit{the } s_i \textit{ form a partition of } \mathcal{A} \\ & \text{i.e., } \sum_i s_i = \mathcal{A}, \textit{ and} \\ & s_i \textit{ are positive integers} \end{array}$$

Proof: First notice that the surface area of any configuration will be an even number since each cell face contributing to the surface area has a corresponding parallel face on the the other side of the configuration which also contributes. We will therefore bound the number of these pairs or the "semi-surface-area". As one way of calculating the semi-surface-area of a configuration of cells, consider the two-dimensional slices of the configuration along one dimension. Each slice contributes an amount at least as big as its semi-perimeter to the semi-surface-area. This takes care of the semi-surface-area in two of the dimensions. In the third dimension, the contribution to the semi-surface-area is at least as big as the area of the projection of the entire configuration onto the plane in that direction, which is at least as big as the maximum slice volume. If the ith slice is of size s_i, then its minimum possible contribution to semi-surface-area in two of the dimensions is $\mathcal{S}^*(s_i)$. Therefore $\max_i\{s_i\} + \sum_i \mathcal{S}^*(s_i)$ is a lower bound on the semi-surface-area of the configuration. Clearly any configuration of $\mathcal{A}$ cells may be partitioned into slices, inducing a partition $\{s_i\}$ of the integer $\mathcal{A}$. Therefore the minimum $\max_i\{s_i\} + \sum_i \mathcal{S}^*(s_i)$ over all partitions $\{s_i\}$ of $\mathcal{A}$ must be a lower bound on semi-surface-area.

To show that the bound is tight, we show that given *any* partition $\{s_i\}$ of $\mathcal{A}$ it is possible to construct a corresponding configuration of $\mathcal{A}$ cells with semi-surface-area exactly $\max_i\{s_i\} + \sum_i \mathcal{S}^*(s_i)$. The configuration is constructed so that each slice is a partial square of area s_i, and the slices are ordered from smallest to largest with maximum overlap between each slice and its neighbor (the amount of overlap will be the equal to the area of the smaller slice). The slice-convexity of the individual slices and the overlap property guarantee that the semi-surface-area is exactly equal to $\max_i \{s_i\} + \sum_i \mathcal{S}^*(s_i)$. ∎

The d-dimensional analog of surface area can be defined recursively using the reasoning from the proof of theorem 24. Define the d-surface-area of a configuration of grid cells in d dimensions to be the number of $(d-1)$-dimensional cell faces on the exterior of the configuration. For $d = 2$ this is the perimeter of the configuration and for $d = 3$ it is the surface area. Let $\mathcal{S}^*_d(\mathcal{A})$ denote the minimum d-dimensional semi-surface area achievable by a configuration of size $\mathcal{A}$.

Theorem 25 *The d-surface-area for a configuration of $\mathcal{A}$ cells must be at least*

$$\begin{array}{rl} 2 \cdot \min_s & \left(\max_i \{s_i\} + \sum_i \mathcal{S}^*_{d-1}(s_i)\right) \\ \textit{such that} & \textit{the } s_i \textit{ form a partition of } \mathcal{A} \\ & \text{i.e., } \sum_i s_i = \mathcal{A} \\ & s_i \textit{ are positive integers} \end{array}$$

Proof: For any d-dimensional configuration, each 1-dimensional slice through it intersects at least two $(d-1)$-dimensional cell faces on the exterior of the configuration. (The coordinates of cells in a 1-dimensional slice are fixed in $d-1$ of their dimensions while the remaining coordinate varies.) Therefore, the d-surface-area is bounded by twice the number of 1-dimensional slices intersecting the configuration. In order to count these 1-dimensional slices, partition the configuration into $(d-1)$-dimensional slices (select any one of the d dimensions to partition along). Let s_i denote the volume of the ith $(d-1)$-dimensional slice. The number of 1-dimensional slices intersecting the ith slice in the $d-1$ unconstrained dimensions is at least as big as the minimum possible $(d-1)$-semi-surface-area for a slice of volume s_i. Therefore the total number of 1-dimensional slices intersecting the configuration in the dimensions orthogonal to the slicing is $\geq \sum_i \mathcal{S}^*_{d-1}(s_i)$. The number of 1-dimensional slices intersecting each slice in the constrained dimension is s_i, therefore the number of 1-dimensional slices intersecting the entire configuration in that dimension is at least $\max_i \{s_i\}$. In total, the number of 1-dimensional slices intersecting the configuration is at least $\max_i \{s_i\} + \sum_i \mathcal{S}^*_{d-1}(s_i)$ ∎

In order to prove that the bound of theorem 25 is tight, it is necessary to show that minimum $(d-1)$-surface-area slices can be "stacked" together with minimum overlap. This requires some geometric information about such slices, which is lacking for $d > 3$. Unfortunately the bounds of theorems 24 and 25 are not efficiently computable, since it requires the generation of all partitions of $\mathcal{A}$. The number of partitions of the integer n is asymptotic to $e^{\pi\sqrt{2n/3}}$ [7].

Rosenberg [16] proved that the d-semi-surface-area of a configuration of size $\mathcal{A}$ is at least $\lceil d\mathcal{A}^{1-1/d} \rceil$ (but did not show that this expression is in fact tight for $d = 2$). This bound is not tight however for $d \geq 3$, as demonstrated by the example in which $d = 3$ and $\mathcal{A} = 5$ where $\lceil 3 \cdot 5^{2/3} \rceil = 9$ and $\min_{s_i} \max_i \{s_i\} + \sum_i \mathcal{S}^*(s_i) = 10$ (this is the smallest such example when $d = 3$). For $d = 3$ the minimum surface-area appears to always be attained by a "partial cube", *i.e.*, a configuration of cells which is as close to a cube as possible for its size. The conjecture has been computationally verified for $\mathcal{A}$ up to 300 by a program which computes the bound by enumerating the partitions for a given $\mathcal{A}$. For each $\mathcal{A}$, one of the partitions minimizing $\max_i \{s_i\} + \sum_i \mathcal{S}^*(s_i)$ is a set of slice sizes for a partial cube. If the conjecture is true, it would make the bound of theorem 25 tight for $d = 4$.

Chapter 4

Heuristics

In this chapter we use the geometric information about good solutions of both diversity and perimeter minimization to develop heuristics for both problems. We then combine the problem-specific techniques with genetic algorithms to create a robust method.

4.1 A Diversity Minimization Algorithm

The diversity minimization problem can be solved in a rough sense by placing optimal or near-optimal tiles for each processor in the given grid, allowing splitting of tiles along horizontal or vertical lines. This forms the basis for the heuristic solution technique presented below. Algorithm TILE is presented in figure 4.1. Each of the major steps in the algorithm will be described in detail below.

The first step of the algorithm is to generate the set of optimal blocks. As described in chapter 2.2.2, each optimal tile can be inscribed in a rectangle with optimal semi-perimeter. Given the area of the optimal shapes, it is easy to generate the dimensions of the possible circumscribing rectangles. Among the tiles

```
generate all optimal blocks
pick an optimal shape for each processor
for p = 1 to N            /* blocks placement */
    place the block for processor p
for p = 1 to N            /* fringe placement */
    place the fringe for processor p
for s = 1 to swap_limit   /* swapping */
    pick a pair of cells (c1,c2)
if swapping assignment of c1 and c2 improves diversity then
        swap c1 and c2
```

Figure 14: Algorithm TILE

which can be inscribed in these rectangles the algorithm restricts itself to those which are made up of a rectangular "block" of cells plus a single row or column of "fringe" cells. Rather than explicitly representing each possible placement of fringe cells, a block implicitly represents all tiles constructed by adding on the appropriate number of fringe cells to that block. For example, the tiles of area ten considered by the algorithm would be described by the categories: a 3×3 block plus one fringe cell, a 2×5 block (plus no fringe cells), and a 2×4 block plus 2 fringe cells. If the number of grid cells is not evenly divisible by N, then two sets of optimal shapes must be generated for the two different tile areas.

The next step in the algorithm is to pick one of the block shapes for each processor in the problem instance. In effect, this causes the algorithm to focus a particular category of optimal tiles for each processor. The algorithm places the block and the fringe portion of each tile separately into the grid. This approach is reasonable since the semi-perimeter of a tile is not affected if the fringe is placed far away from the block. If the complete block for a processor does not fit into the empty space in the grid the algorithm tries to fit it in by splitting it and fitting in the resulting smaller pieces without increasing the semi-perimeter. If only part of a block will fit, the remaining part is dealt with as fringe cells.

Once the blocks are placed, the algorithm places the fringe cells. They are placed in rows of the vertical strip containing the block and/or in columns of the horizontal strip containing the block. The fringe for a tile is placed optimally if and only if it fits into one such row or column. Any fringe cells which can not be placed as described above are placed arbitrarily after the fringes for all other tiles are placed.

By the end of the fringe placement step, a feasible assignment of grid cells to processors has been created. The assignment is not necessarily optimal however, so the algorithm has a final step which looks for pairs of cell assignments to swap. A limit on the number of swaps which the algorithm evaluates is given as input to the algorithm. Swaps which either decrease the diversity or do not increase the diversity are actually performed. Experiments show that performing non-increasing swaps is necessary in order to find decreasing swaps.

In figures 15, 16, and 17 the results after each step of algorithm TILE are illustrated for the example of a 17×17 grid with $N = 17$, and using a 4×4 block plus a fringe of one as the optimal shape for each processor. The blocking step places sixteen 4×4 blocks into the grid. The seventeenth block will not entirely fit, therefore TILE places as much of the block as will fit into the remaining free space. The fringe placement step then places the fringes of size one for the first sixteen processors and the fringe of size 13 for the last processor. The small fringes are all placed optimally, *i.e.*, they increase the 2-perimeter of each processor by only one. The large fringe for the last processor can not be placed optimally because of the shape of the grid, and the last processor has a 2-perimeter of 18 (9 more than the minimum value) at the end of this step. The swapping step then examines possible swaps and performs those which do not increase total diversity. For this example, if the limit on the number of swaps examined is set to 1.7 million, the result is a solution with optimal diversity.

4.2 Data Structures

The block and fringe placement steps of the algorithm work with rectangular blocks of grid cells. The basic computation in these steps is to assign a rectangular group of grid cells to a processor. In these steps, the grid does not have to be represented as a two-dimensional array of grid cells. Instead, the (partial)

1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	X
1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	X
1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	X
1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	X
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	X
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	X
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	X
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	X
9	9	9	9	10	10	10	10	11	11	11	11	12	12	12	12	X
9	9	9	9	10	10	10	10	11	11	11	11	12	12	12	12	X
9	9	9	9	10	10	10	10	11	11	11	11	12	12	12	12	X
9	9	9	9	10	10	10	10	11	11	11	11	12	12	12	12	X
13	13	13	13	14	14	14	14	15	15	15	15	16	16	16	16	X
13	13	13	13	14	14	14	14	15	15	15	15	16	16	16	16	X
13	13	13	13	14	14	14	14	15	15	15	15	16	16	16	16	X
13	13	13	13	14	14	14	14	15	15	15	15	16	16	16	16	X
17	17	17	17	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 15: After block placement step of TILE

1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	1
1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	2
1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	3
1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	4
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	5
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	6
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	7
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	8
9	9	9	9	10	10	10	10	11	11	11	11	12	12	12	12	9
9	9	9	9	10	10	10	10	11	11	11	11	12	12	12	12	10
9	9	9	9	10	10	10	10	11	11	11	11	12	12	12	12	11
9	9	9	9	10	10	10	10	11	11	11	11	12	12	12	12	12
13	13	13	13	14	14	14	14	15	15	15	15	16	16	16	16	13
13	13	13	13	14	14	14	14	15	15	15	15	16	16	16	16	14
13	13	13	13	14	14	14	14	15	15	15	15	16	16	16	16	15
13	13	13	13	14	14	14	14	15	15	15	15	16	16	16	16	16
17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17

Figure 16: After fringe placement step of TILE

1	2	1	1	2	1	2	2	17	17	4	17	4	4	17	4	17
1	3	1	1	2	3	2	2	17	17	4	17	4	4	17	4	3
1	3	1	1	2	3	2	2	3	17	4	17	4	4	17	4	3
1	3	1	1	2	3	2	2	3	11	11	11	8	8	8	4	3
5	5	5	5	6	6	7	6	7	7	7	6	6	8	8	8	7
5	5	5	5	6	6	7	6	7	5	7	6	8	8	8	8	7
5	5	5	5	6	6	7	6	7	7	7	6	8	8	8	8	7
5	3	5	5	6	3	7	6	3	5	7	6	6	8	8	8	7
10	9	12	9	12	12	10	10	11	11	11	11	9	12	10	12	9
10	9	12	9	12	12	10	10	11	11	11	11	9	12	10	12	9
10	9	12	9	12	10	10	10	11	11	15	11	9	12	10	12	9
10	3	12	9	12	3	10	10	3	11	11	11	9	12	10	16	9
13	13	14	13	14	14	14	14	15	15	15	15	16	16	16	16	13
13	13	14	9	14	14	14	13	15	15	15	15	16	16	16	16	9
13	13	14	13	14	14	14	13	15	15	15	15	16	16	16	16	13
13	13	14	13	14	14	14	13	15	15	15	15	16	16	16	16	13
1	2	1	1	2	1	2	2	17	17	4	17	4	4	17	4	17

Figure 17: An optimal solution resulting from the swapping step of TILE

assignment of grid cells is represented as a list of assigned rectangles and free rectangles. This data structure allows an efficient implementation of the the first two steps of the algorithm.

Consider the problem of finding free space in a partially filled grid in which to place a fixed-size block. With a cell-based representation, the time required to determine if a particular block position is valid is proportional to the size of the block (the status of each grid cell in the candidate area must be checked). Furthermore, the number of candidate positions for the block which must be checked is proportional to the number of free grid cells. In contrast, with a rectangle-based representation, the location of a valid block position requires only finding a free rectangle which is at least as big as the block. Therefore a block can be placed by comparing its height and width to the heights and widths of each of the free rectangles. To guarantee that a block will be placed whenever possible, it suffices to represent the free space as a list of the *maximal free rectangles* in the grid.

In order for such a list to be an efficient data structure for block and fringe placement, its size must be reasonable. Clearly each rectangle (assigned or free) can be represented in constant space: four integers (height, width, and coordinates of the upper left corner) suffice to represent a rectangle regardless of its size. Each tile in the assignment is represented by one rectangle for the block and one or more rectangles for the fringe, therefore the length of the list of assigned space is roughly linear in the number of assigned blocks.

Initially, the free space is represented by a single rectangle comprising the entire grid. After n blocks have been placed in the grid, a theoretical upper bound on the length of the list is $O(n^4)$, but in practice the lists are sublinear in the number of processors in the problem. Figure 4.2 illustrates one instance of

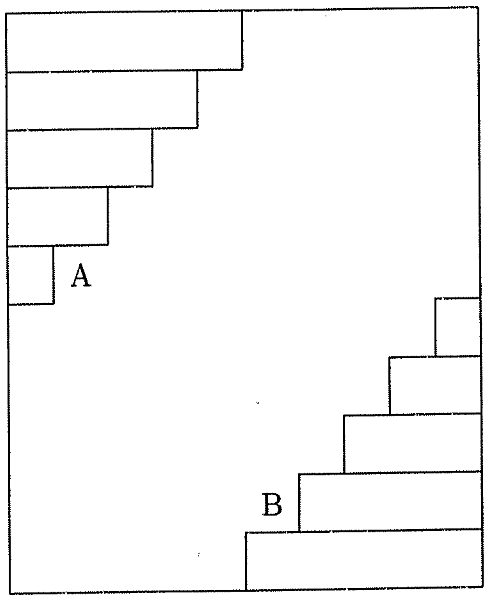
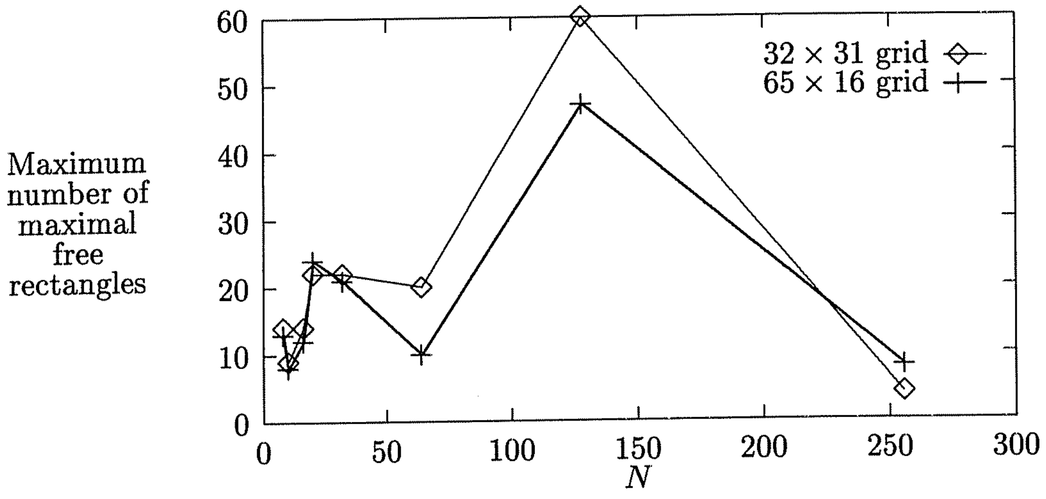


Figure 18: An example with $O(n^2)$ maximal free rectangles

a class of examples which create $O(n^2)$ maximal free rectangles by placing $O(n)$ rectangles. In the figure, 10 rectangles have been placed and there are 36 maximal free rectangles (one for each pair of opposing inner corners). Similar examples may be constructed of any size with the number of inner corners linear in the number of rectangles placed and hence a quadratic number of maximal free rectangles.

To bound the number of maximal free rectangles by $O(n^4)$, observe that after placing n rectangles, both the horizontal and vertical dimensions of the grid are partitioned into $\leq 2n + 1$ intervals determined by the coordinates of the corners of the n rectangles. This partition effectively defines a coarser grid of dimension at most $(2n + 1) \times (2n + 1)$. Every non-maximal free rectangle in the grid can be extended in each dimensions until it hits the edge of some partition, therefore every maximal free rectangle is comprised of cells of the coarse grid. The coarse



grid contains at most $\binom{2n+2}{2}^2 = 4n^4 + 12n^3 + 13n^2 + 6n + 1$ rectangles (choose 2 horizontal and two vertical boundaries to define a rectangle), therefore the number of maximal free rectangles is bounded by $O(n^4)$.

In actual runs of algorithm TILE, the length of the free list is small and appears to be independent of the number of processors in the problem instance. Figure 4.2 shows the maximum list length as a function of the number of processors for a number of problem instances. The maximum size of the list does not seem to grow as n^4 or even n^2 in any of the experiments. These empirical results show that the free rectangle list is a very efficient data structure compared to the explicit representation of each grid cell.

The independence of N and the maximum list length is interesting because it shows that worst case behavior does not occur in practice. This is due to the fact that the program always places blocks and fringes in the upper left corner of a maximal free rectangle, keeping the shape of the free space relatively simple.

Chapter 5

A genetic algorithm

Algorithm tile described in the last chapter uses knowledge about optimal tilings extracted from the theory of minimum semi-perimeter groups. However it leaves unanswered the question of which combinations of optimal block shapes are promising for creating good assignments. Algorithm tile expects as input a collection of optimal block shapes which determines the resulting assignment. If an instance of the two-dimensional minimum diversity problem has N processors and a library of optimal block shapes with s entries then there are s^N possible inputs to algorithm tile for that problem instance. We may consider the problem of finding the best input to algorithm tile as a combinatorial optimization problem. As a technique for obtaining good solutions to this problem, *i.e.*, inputs to algorithm tile which produce good assignments, we use the genetic algorithm (GA) paradigm.

The genetic algorithm is a search method for a large discrete search space which is motivated by concepts from biology and genetics. The basic idea is to simulate the effects of natural selection on an evolving population of points in the search space, hopefully producing over time the "best" points in the search space. The basic components of a genetic algorithm are a "fitness function" which provides a way of evaluating the desirability of the points in the search space, and methods for combining points to produce new ones. Usually the points in the search space are represented in the GA as strings of integers called "individuals", and several genetic operators called "selection", "crossover", and "mutation" are defined to combine and alter individuals to create new ones. The individuals correspond to chromosomes and the genetic operators correspond to natural selection, mating, and mutation in biological populations. In the next section we we present a high level view of the GA and describe the genetic operators in detail,

5.1 A generic GA

Given a representation of points in the search space as individuals, a fitness function, and genetic operators, the generic genetic algorithm has the following form:

```
create initial generation
while not done
    evaluate fitness of each individual in current gen
    select a mating pool
    perform crossover among mating pool members to get new gen
    perform mutation on each individual in new gen
```

Each of the components of the above algorithm comes in several flavors. The initial generation can be created randomly or information from the underlying application can be used to make an intelligent guess at good individuals for the initial generation. Evaluation of fitness for the individuals requires that a subroutine which computes the fitness function be available. A fitness computation may be very fast or may require large amounts of time and memory depending on the application. The selection step in the algorithm involves using the fitness of the individuals as a criterion for participation in crossover. From among the current generation, a number of individuals are chosen for the "mating pool". Individuals with good fitness values have a high probability of being selected and those with bad fitness have a low probability. The new generation is formed by combining the mating pool members, so selection determines which individuals pass on their genes. The crossover operator combines two individuals to produce two offspring. The operation may be simple or complicated depending on the application. In some cases, simple crossover produces offspring which do not correspond to feasible points in the search space, necessitating some post-processing to maintain feasibility. Once the new generation has been created, mutation occurs in each offspring, maintaining diversity in the population and adding randomness to the algorithm. The creation of new generations continues until some stopping criteria are met, such as an optimal or near-optimal individual is found, or computing time is exhausted.

5.2 A minimum diversity GA

Our GA is a search method over the space of inputs to algorithm tile and our fitness function uses algorithm tile to compute feasible assignments. Several factors motivate this two-level approach to solving the minimum diversity problem. The space of possible inputs to algorithm tile is significantly smaller than the space of feasible partitions of a grid, the input to algorithm tile encodes high-level information about a partition, there is a simple encoding of the inputs which allows the standard genetic operators to work without any modification, and a method for picking good inputs is necessary for algorithm tile to be practical. In this section we lay out the details of our GA.

5.2.1 Encoding

The obvious way to represent the input to algorithm tile as a string of integers is to assign an index to each optimal block shape and represent the input as a list of these indices. This encoding works well because the length of the individuals and the range of values for each string position are both reasonably small. The length of individuals in the GA under this encoding is equal to N, the number of groups in the problem instance. The allowable range of values in each position depends on the specified group size corresponding to that position, and is bounded by $\lceil 2\mathcal{A}^{1/2} \rceil /2$ where $\mathcal{A}$ is the group size. On all of our test problems we choose the group sizes to be as similar as possible so the range of values in each position is similar to the others.

5.2.2 Fitness Function

The fitness function is a map from the space of individuals to the non-negative reals. The computation of the fitness an individual has two parts. First algorithm tile is used to create a feasible partition derived from the individual. The fitness value F is then calculated as

$$F = (U - (d - \mathcal{B})) + aR + b$$

where d is the diversity of the partition generated by the heuristic, $\mathcal{B}$ is the lower bound on the minimum diversity, U is the largest $d - \mathcal{B}$ value in the current generation, R is the range of diversities among the population, and a and b are scalar parameters. Similar scaling of fitness values is advocated by Ferris *et al.* to control convergence [1, 4]. We found our GA to perform well with parameter settings $a = 0.2$ and $b = 1$.

5.2.3 Initial Population

We experimented with two techniques for creating the initial generation for our GA: individuals made up of either uniform or of random values. By uniform we mean that if the range of allowed values is $[1, r]$, approximately $1/r$th of the individuals in the initial population have all alleles equal to 1, $1/r$th have all alleles equal to 2, *etc.* By random, we mean that each allele in the initial population is a random integer in the range $[1, r]$. In our experiments, better results were obtained when starting with uniform individuals. This is explained by the observation that many optimal or near-optimal solutions use the same block shapes for all the groups. It is also a good idea to evaluate the fitness of all of the possible uniform individuals because this will determine whether there is an optimal solution made up entirely of rectangles.

5.2.4 Genetic Operators and "Survival" Policies

Three primary genetic operators (selection, crossover and mutation [11]) are presented, followed by the various survival policies that we considered.

Selection

The individuals in the mating pool are selected by a roulette wheel strategy. That is, the probability that each individual is chosen is proportional to its fitness. An individual with high fitness may have several copies in the mating pool.

Crossover

Crossover provides a powerful exploration capability by exchanging portions of the internal representations of two parents. We use three types of crossover: one-point crossover, two-point crossover and random crossover. Two-point crossover treats the string as a ring. Two points are selected at random to break the ring into two portions. The parents exchange either portion to produce two children. If we look at the front end of the string as a break point and select the other break point, then we have one-point crossover. In uniform crossover, each gene of the first child is inherited from either parent with equal probability, while the second child inherits those genes not inherited by the first child. The crossover rate, the probability that each mating pair actually performs crossover, was set between 0.7 and 0.9 in our tests.

Mutation

Mutation creates new individuals by modifying several gene values of an existing individual to maintain the diversity of the population. The process of mutation comes after selection and crossover. Instead of mutating each gene with some fixed probability, which requires the generation of many random numbers, we achieve a similar effect by picking a random number of genes to mutate. The expected number of mutated genes when using the standard mutation scheme is the mutation rate p_m times the string length N. We therefore randomly choose m, the number of genes to mutate, in the range $[0, \lceil 2Np_m \rceil]$. We then randomly choose m positions in the child string and randomly change their values. We tried various mutation rates and settled on $p_m = 0.01$ for our experiments.

Survival policies

A survival policy is a non-standard step in the GA in which each offspring from the crossover step is compared to the previous generation to determine if it survives to participate in the selection step. If a particular offspring is judged too poor to participate in selection, it is replaced by a member of the previous generation in order to maintain a fixed population size. Five survival policies are described below. Two of the policies use the notion of the "incumbent" diversity value, defined to be the smallest value obtained so far by the algorithm.

1. pure: both children are placed in the new generation.

2. keep-incumbent: if the best diversity value of all the children does not improve on the incumbent value, then the children of parents with the incumbent value do not survive and each is replaced in the new generation by its better parent. This policy maintains the condition that the best values in the generations are monotone.

3. no-worse: if a child is not as good as the worst of all the individuals in the previous generation then that child is replaced probabilistically by one of its parents. (If the probability that a "worse" child survives is p_b, then $p_b \approx 1$ is essentially the same as pure survival.) In our experiments, $p_b = 0$, which maintains the condition that the worst values in the generations are monotone.

4. keep-incumbent and no-worse: a combination of the above two policies.

5. best-two: from each parent-child 4-tuple, choose the best two. This policy maintains the condition that both the best and worst values in the generations are monotone.

Chapter 6

Computational Results

We implemented our GA on a Thinking Machines Corporation Connection Machine CM-5 in the Computer Sciences Department at the UW-Madison. This machine consists of 64 SPARC processors each with 32 megabytes of local memory connected by a "fat tree" network [19]. In MIMD mode, which we used, the processors run asynchronously and communicate via calls to the routines in the TMC message passing library CMMD (CMMD release 2.0 Beta 2, September 16, 1992).

The high-level structure of the parallel implementation of our genetic algorithm is presented in figure 19. We use the host-node programming paradigm in which there is a host processor which coordinates the node processors working in parallel. In the GA, the genetic operators crossover, mutation, and survival as well as fitness function evaluation can run for all individuals in parallel since they require no global information. A processor performing selection must have available all fitness values for the current

generation, hence the host processor does selection. For each generation, the selection operation picks out the best individuals, and the crossover step combines

```
in parallel on node processors:
     create initial generation and evaluate fitness of individuals
     send fitness values of initial generation to host

while convergence not reached do
     on host processor:
          receive fitness values from nodes
          *select* a mating pool
          coordinate distribution of mating pool between node processors

     in parallel on node processors:
          receive two mating pool individuals (possibly from other nodes)
          perform *crossover*
          perform *mutation* on crossover results
          run algorithm TILE on each offspring
          scale diversity values to get fitness values
          apply *survival* strategy to get two new individuals per node
          send fitness values of new generation to host
```

Figure 19: The Parallel Genetic Algorithm

pairs of selected individuals producing a new generation of offspring. Each offspring is mutated, and the resulting individual is evaluated using algorithm TILE. Finally, a survival strategy determines the makeup of the new generation using the selected individuals and the new offspring as candidates. The final result is a new generation with the same number of individuals as the previous generation. The processors of the CM-5 are split into partitions, with a separate control processor coordinating the jobs on each partition. The partition size available on our CM-5 is 32 nodes, so we designed our genetic algorithm with a population size of 64 (two individuals per processor).

We report below tests of our GA on fifteen different problems. Table 3 shows the dimensions, number of 0-1 variables (rows $\times$ columns $\times$ processors) and number of constraints (rows $\times$ columns + processors) in a fixed-charge transportation formulation, and lower bounds for the test problems. The computing time per generation varies from 0.4 seconds to 600 seconds on the test set. Problems B1-B11 come from the database application of diversity minimization and are the result of the MAGIC partitioning strategy on the Wisconsin Benchmark relations [2]. B1-B8 are problems on a 32×31 grid and B9-B11 are problems on a 65×16 grid. These benchmark problems were used as test problems for Ghandeharizadeh's diversity minimization heuristic [9], which is roughly comparable to the block placement step of algorithm TILE. Our GA produces solutions which are at least as good as Ghandeharizadeh's heuristic on his benchmark problems B1-B11. The new test problems K1-K4 are interesting because of their size and because their optimal values are known to be equal to the lower bounds.

Our computational experiments were designed to test how well the GA can do on the test problems, how the quality of its solutions are affected by variations in the program parameters, and how efficiently the program runs on the

	grid	procs(N)	# variables	# constraints	lower bound
B1	32 × 31	8	7,936	1,000	184
B2	32 × 31	10	9,920	1,002	200
B3	32 × 31	16	15,872	1,008	256
B4	32 × 31	20	19,840	1,012	292
B5	32 × 31	32	31,744	1,024	384
B6	32 × 31	64	63,488	1,056	512
B7	32 × 31	128	126,976	1,120	768
B8	32 × 31	256	253,952	1,248	1,024
B9	65 × 16	8	8,320	1,056	184
B10	65 × 16	32	33,280	1,072	384
B11	65 × 16	128	133,120	1,105	768
K1	101 × 101	101	1,030,301	10,302	2,121
K2	142 × 71	142	1,431,644	10,153	2,414
K3	128 × 128	128	2,097,152	16,512	2,944
K4	1,000 × 1,000	1,000	1,000,000,000	1,001,000	64,000

Table 3: Test problems and their lower bounds

CM-5. To establish baseline values for the parameters in the GA, we performed many trial-and-error runs on the million-variable problem K1. Using $p_m = 0.01$, $p_c = 0.9$, an initial generation of uniform individuals, one-point crossover, and the best-two survival strategy, the GA solved K1 to optimality in 14 generations. We therefore define the above parameter settings to be the baseline. In the experiments reported here, we fix all but one parameter to the baseline values and vary the remaining parameter. To asses the quality of the solutions computed by the GA, we measured the relative distance from the solutions to the theoretical lower bounds. For all of the test problems the GA obtained solutions within 9% of the bound (the solutions, of course, may be closer than this to the optimal value, which is unknown for these problems except for B5 and B8). For several problems, including the million-variable problem K1, the solutions were provably optimal, *i.e.*, the relative distance was 0.

Figure 20 plots the progress of the GA on a run which is cut short by the appearance of an optimal individual. In the graph, the minimum, maximum and mean distance from the lower bound over the entire population is plotted for each generation. The steep drop in the first few generations is typical of our GA. The utility of the lower bound is evident here – the program was able to terminate after only fourteen generations.

Figure 21 shows an optimal partition obtained by our genetic algorithm for the million-variable problem K1. Each processor is represented by a different shade of gray (color visualization in which a different color is associated with each processor index is even more useful in this context). In the figure, the blocks and fringes are visible, as are several scattered cell assignments resulting from swapping.

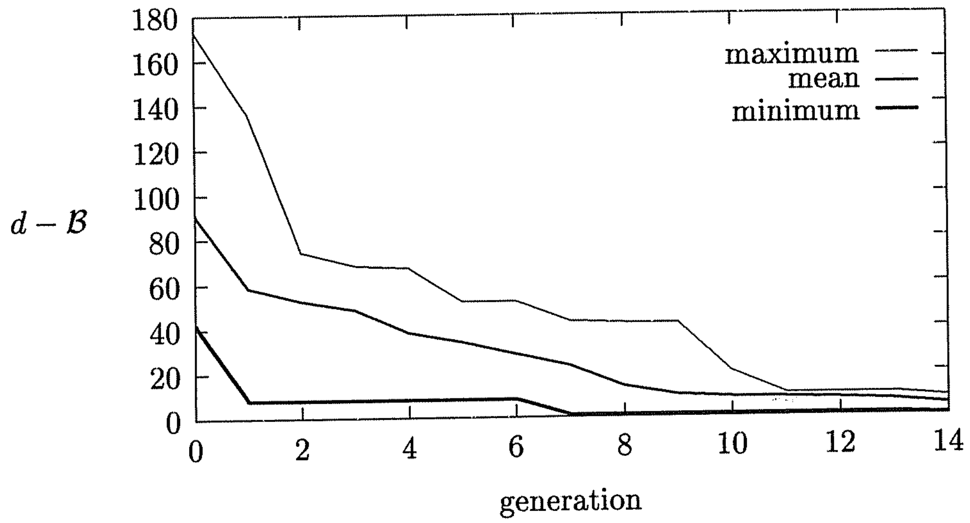


Figure 20: The GA on K1 with baseline parameters

6.1 Interprocessor Communication

In figure 22, the percentage of total running time spent on communication for the problems B1-B11 is plotted as a function of N. The problems on a 32×31 grid are grouped on one curve and the problems on a 65×16 grid are grouped on the other curve. In order to ensure that algorithm TILE was performing the same amount of work for each problem size, we set a limit on the number of cell swaps that are examined in the swapping step of the algorithm. This limit was set to 0.2% of the number of possible swaps, so that the same proportion of possible swaps was considered for each problem. In the GA, the individuals (whose length is N) are exchanged between processors via message passing, therefore the length of the inter-node messages is proportional to N. This causes the amount of time spent on interprocessor communication to increase with N. On the other hand,

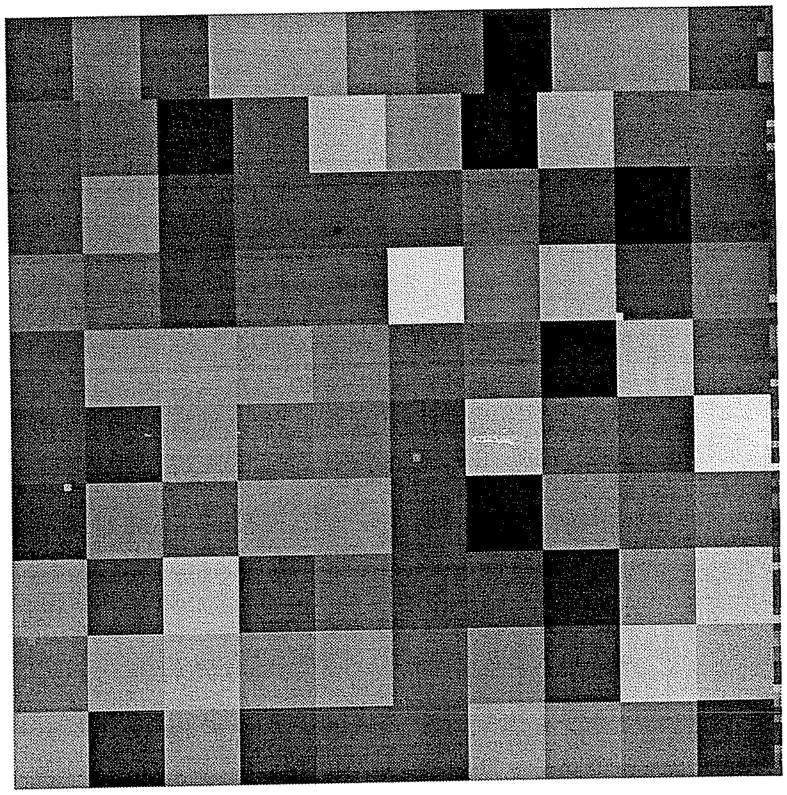


Figure 21: An optimal partition of a 101×101 grid among 101 processors

the running time of algorithm TILE increases with N. The graph indicates that the increase in computation time dominates the increase in communication time, since the fraction of time spent on communication decreases with N. For problems K1, K2, K3, and K4, the fractions of time spent on communication are less than 0.0015, 0.0013, 0.0006, and 0.0012 respectively. For the K4 experiment, it would take an unreasonable amount of computing time to examine 0.2% of the total possible swaps, so we set the swap limit to $300,000$ (a larger limit than for any of the other problems). These results indicate that as the problem size grows, the communication ratio tends to zero, making the GA an efficient method for large-scale problems.

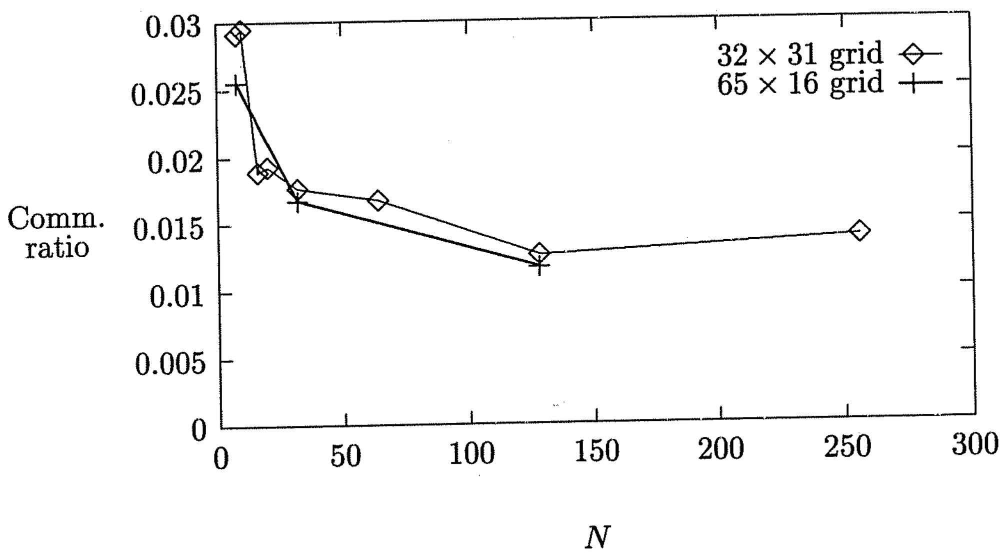


Figure 22: Fraction of time in communication for problems B1-B11

6.2 Effect of random seed

Selection, crossover, mutation, and survival all require random number generators to simulate random variables. We found that the seeds used to initialize the generator can considerably affect the performance as figures 23 and 24 show. The results in the figures were obtained by making 10 runs for each test problem with 10 different random seeds using the baseline parameter settings (there are not 10 distinct stars in each column because some random seeds produce identical solutions). The range of solutions obtained with different seeds suggests that a good way to use the GA is to make several runs with different random seeds and take the best one.

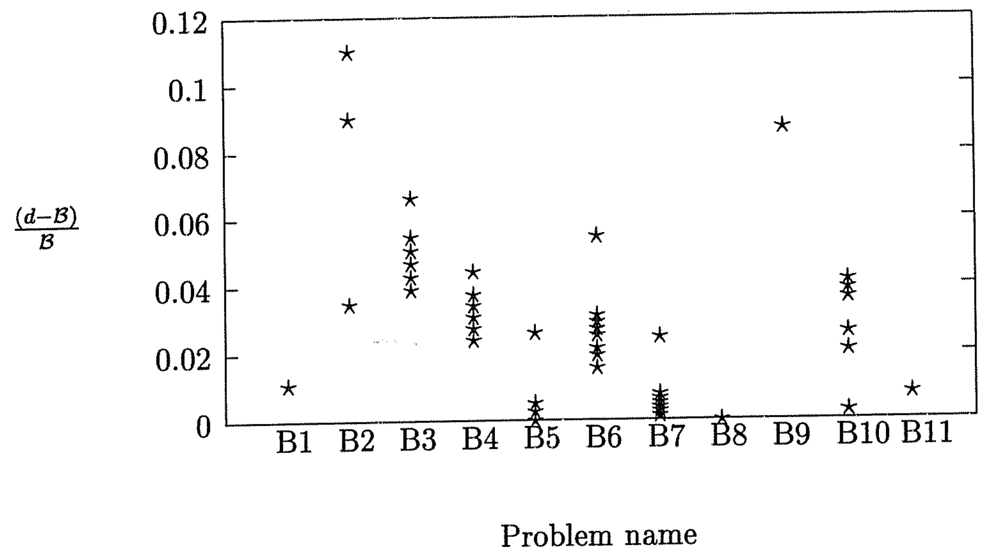


Figure 23: Effect of random seed for problems B1-B11

6.3 Effect of survival strategies

In figure 25, we compare the performance of different survival policies. The horizontal axis indicates the eight different test problems and the vertical axis indicates the diversity relative to the lower bound: $(d - \mathcal{B})/\mathcal{B}$ where d is the best diversity obtained by the genetic algorithm after 100 generations, and $\mathcal{B}$ is the lower bound for the problem. Note that for problems B1 through B11, the optimal diversity may not be equal to the lower bound, hence the solutions we obtained may be optimal; for K1 through K4 the optimal diversity and the lower bound are known to be equal. The values shown are the best values in ten runs with different random seeds. In these tests, no particular survival policy is clearly superior. The no-worse, keep-incumbent and no-worse, and best-two policies all were best for at least one problem. On the other hand, the pure and keep-incumbent policies

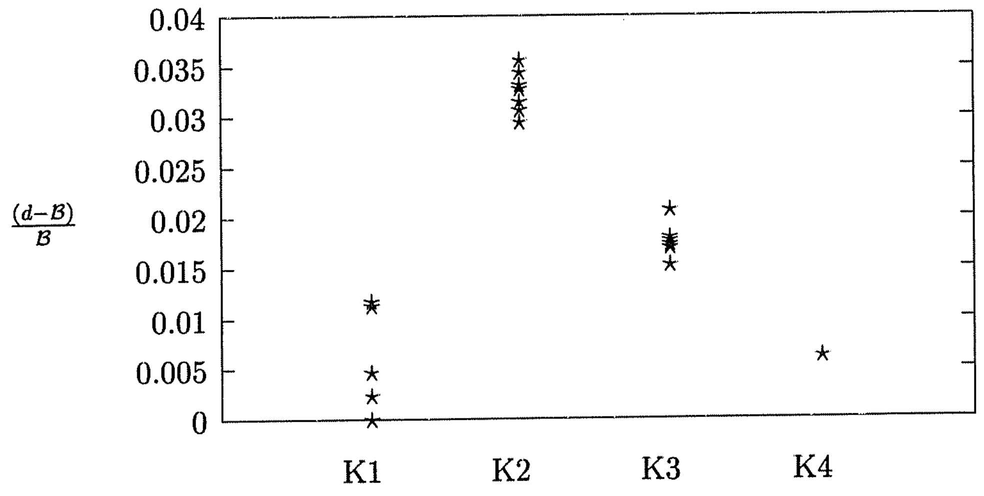
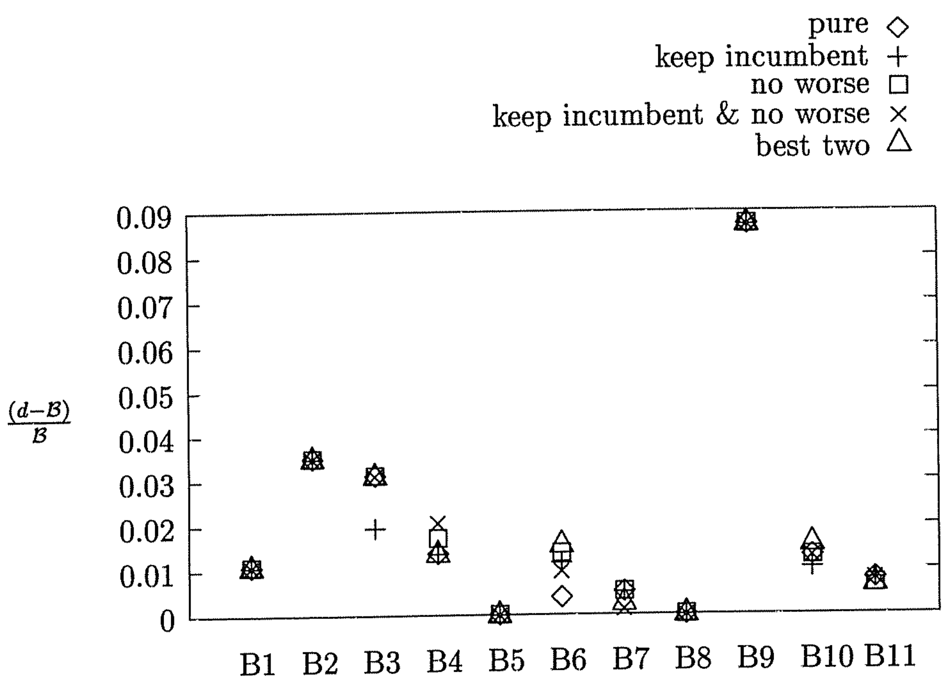


Figure 24: Effect of random seed for problems K1-K4

did not outperform the others on any of the problems. These results suggest that these simpler survival strategies should not be used in our GA.

6.4 Effect of crossover and mutation rates

The effects of varying both the crossover rate and the mutation rate were not as pronounced as was expected. The quality of solutions obtained in 100 generations of the GA was relatively constant for crossover rates at least 0.5 (see figures 23 and 24 for solution qualities when $p_c = 0.9$), and slightly poorer for $p_c < 0.5$. In another set of trials, the mutation rate was varied between 0.001 and 0.10. The results showed good solutions for values under 0.02 and slightly poorer solutions for the higher values of p_m (the results for $p_m = 0.01$ are shown in figures 23



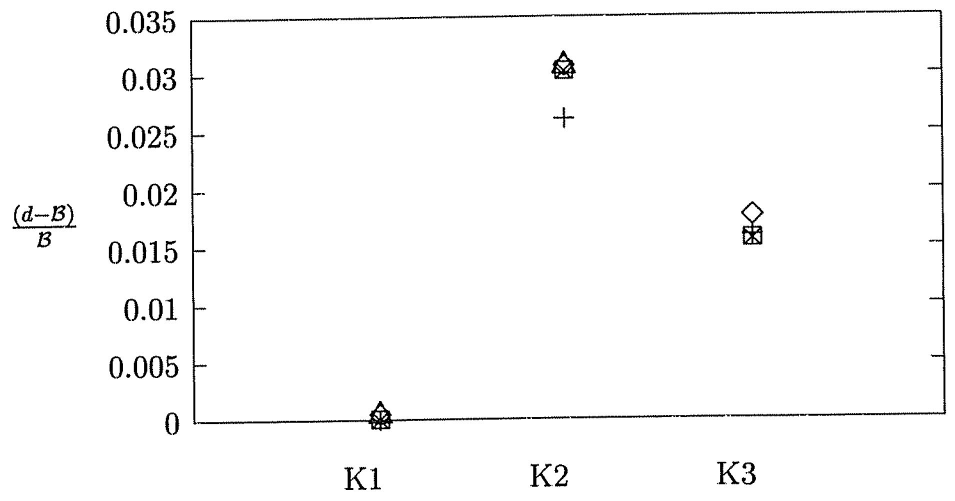


Figure 25: Effect of survival policies

and 24). These findings combined with the results of our trial-and-error runs on problem K1 indicate that high crossover rates combined with low mutation rates lead to the best solutions.

Chapter 7

Conclusions and Future Research Directions

7.1 Summary and Conclusions

Minimizing communication and achieving a good load balance are two competing goals that designers of parallel algorithms and systems weigh against each other in the quest for efficient implementations. Assigning all the computational tasks to a single processor will minimize the interprocessor communication at the expense of the load balance (and hence the efficiency and computing time), while naively splitting up the computation over all the processors may necessitate a large volume of interprocessor communication which also decreases efficiency. In this dissertation, the problem of minimizing communication given a grid-oriented set of tasks and communication requirements and a fixed set of processor loads is discussed. This work has provided both an elegant theory and an efficient method for computing optimal and near-optimal solutions.

The two NP-hard problems of diversity minimization and perimeter minimization (arising from various parallel computing applications) have been investigated and a common theory has been built which characterizes good solutions to both problems. The theory provides lower bounds on the objective functions of both diversity and perimeter minimization in two dimensions, and also extends to diversity minimization and surface area minimization in many dimensions. The lower bounds yield geometric sufficient conditions for optimal configurations of computational tasks and/or data for individual processors in the system. By showing that these configuration fit together to tile grids, these results are used to derive actual closed-form optimal solutions for certain classes of problems.

A heuristic for the diversity minimization problem has been developed based on the theory. The set of possible inputs to the heuristic forms a large discrete space which is searched via a high-level genetic algorithm. A combination of the genetic algorithm and the heuristic produces good solutions for various diversity minimization test problems. A parallel implementation of the method on a Thinking Machines CM-5 produces solutions within 7% of the optimal value on all test problems and achieves better solutions than another heuristic by Ghandeharizadeh. The implementation has been shown to be efficient with respect to its ratio of communication to computation. Several experiments have been performed to investigate the effect of various algorithm parameters on the quality of solutions produced. The results of these experiments will help users to tune the parameters, and provide insight into which components of the algorithm are critical to the quality of solutions.

7.2 Future Research Directions

The work in this thesis can be extended in several directions, both theoretical and computational. The computational complexity of both diversity minimization and perimeter minimization when restricted to the class of rectangular grids remains unknown. The proof of NP-hardness or the discovery of a polynomial-time algorithm for perimeter minimization on rectangular grids would be an interesting result in graph theory.

The lower bounds on d-perimeter are tight, but they do not take into account the shape of the grid. Ad hoc methods have been successful in showing that the diversity and perimeter bounds are not tight in certain instances, but more general results are needed. A lower bound on diversity based on the shape of the grid for the class of "elongated" grids was developed by Meyer and Schultz [10] and is summarized in appendix D. The lower bounds also assume that fixed load-balancing constraints are provided. An interesting variation of the problem which needs to be explored is minimizing diversity or perimeter subject to a constraint on the maximum load-imbalance.

The two-dimensional optimal configurations are well studied, but a complete study of optimal configurations for $d > 2$ has yet to be done. This could include characterizing the dimensions of optimal d-dimensional hyper-rectangles, which is complicated by the fact that unlike the two-dimensional case there exist multiple hyper-rectangles with the same volume *and* the same d-perimeter. For example, the three-dimensional rectangles with dimensions $5 \times 8 \times 9$ and $6 \times 6 \times 10$ both have a volume of 360 and an optimal 3-perimeter of 22 (these are the smallest such pair of hyper-rectangles in the smallest number of dimensions).

The lower bounds on surface area can be improved on several fronts. By proving the conjecture that all "partial cubes" have minimum surface area or by finding some other minimum surface area class of shapes, an efficient method of computing the optimal surface area for a given volume would be obtained (evaluation of the current bound requires exponential time). The lower bound on d-surface-area for general d is not known to be tight for $d > 3$, and a constructive proof might also provide an efficient method for computing the bound.

Algorithm TILE minimizes diversity. The same basic approach could be applied to perimeter minimization, perhaps merging the block and fringe placement steps in order to keep the cells for each processor close together in the grid. TILE could also be modified to create heuristics for the three-dimensional diversity and surface area minimization problems. Various modifications to the genetic algorithm and TILE can be evaluated, including alternate representations of genetic individuals and inclusion of backtracking in the TILE algorithm. The genetic algorithm should scale to parallel machines with larger numbers of processors by simply increasing the population size. Another possible modification is to have each pair of parents produce k offspring (instead of two) and use k processors to evaluate the fitness of the offspring in parallel. Finally, a parallel asynchronous genetic algorithm is possible in which the selection-crossover-mutation-survival cycle, guided by global best and worst fitness values, runs independently on the processors.

Appendix A

Problem Variations

The main body of the thesis is devoted to the investigation of the diversity minimization and perimeter minimization problems. For each of these problems there are several variations on objective functions and constraints which are also of interest. Each section of this appendix explains one or more variations and summarizes what is known about solving the modified problem(s).

A.1 Nonlinear objective functions

For the problem of assigning database fragments to processors, the partitions that minimize overhead (*i.e.* diversity) also, under certain assumptions, minimize the sum of the response times for the queries. DeWitt *et al.* [6] found that a reasonable model for the response time r as a function of n, the number of processors used, is $r(n) = \mathcal{O}n + \mathcal{Q}/n$ where $\mathcal{O}$ is the overhead incurred by using each additional processor, and $\mathcal{Q}$ is the time needed to process the query on a single processor. In the absence of any constraints, the number of processors per query that minimizes response time is $(\mathcal{Q}/\mathcal{O})^{1/2}$. The results developed in chapter 2 show that under load balancing constraints, the minimum number of processors achievable per query is about $N^{(d-1)/d}$ in the case of a d-dimensional grid in which each dimension is partitioned into N intervals and N is the number of processors in the system. If $\mathcal{Q}/\mathcal{O} < N^{2(d-1)/d}$, then the partition with the lowest possible overhead also attains the lowest response time. This condition holds exactly when the time to process the query on a single processor is dominated by the overhead incurred by using $N^{2(d-1)/d}$ processors. For example, if $d = 2$, the condition holds when $\mathcal{Q} < N\mathcal{O}$ or when the system is "broadcast limited" – it costs more to start up all the processors than to process the query on a single processor. For larger d the condition holds under a weaker assumption that a larger hypothetical system of $N^{2(d-1)/d}$ processors would be broadcast limited. These results suggest that in many real world applications, a minimum response time may be achieved by minimizing the overhead.

A.2 Minimizing maximum diversity

An alternative goal in the database application is to minimize the maximum diversity value (over all slices) so that the worst case overhead is as small as possible. In this case, the objective function measuring total diversity is replaced with one measuring the maximum diversity. Clearly the maximum diversity must be at least as large as the average, hence the lower bounds on total diversity provide bounds on the maximum. In many cases a solution that minimizes total diversity also minimizes maximum diversity. For example, the minimum diversity solution illustrated in figure 2.3.2 has a diversity of 3 for each slice. Since the total diversity is minimized and each slice has the same diversity, the maximum diversity must also be minimized. In [10], we derived a lower bound of $\lceil N^{(d-1)/d} \rceil$

on the maximum diversity under equal load balance assumptions. Notice that this bound is independent of the grid size and shape.

A.3 Relaxed load balancing

In the statements of the diversity and perimeter minimization problems, the processor loads are strictly specified. In some applications it may be satisfactory to minimize the objective subject to the constraint that the loads differ by no more than some specified tolerance. If the tolerance is zero or one then the loads for the processors are fixed and the lower bounds presented above apply, but when the tolerance is greater than one, many possible sets of loads will satisfy the constraint. Notice that each processor's load must be at least the average load minus the tolerance. The average load is simply the number of grid cells divided by N. Using this underestimate for each processor load, loose lower bounds may be obtained (see [10]).

A.4 Weighted versions

In the distributed database application of diversity minimization, the assumption is made that all slices of the grid are accessed with equal frequency via database queries. This may not always hold: for example the records of the employees in the EMP relation who earn less than $10K$ might be accessed more frequently than the high level managers who earn more then $40K$ because of more frequent interdepartmental changes, salary changes, *etc.* This property can be incorporated into the formal problem statement by assigning a weight to each slice according to its frequency of access and then minimizing the *weighted* sum of the diversities.

Another simplifying assumption that was made in the problem formulation is that each grid cell (database fragment) contains about the same number of tuples, and therefore assigning the same number of cells to each processor distributes the tuples evenly among the processors. This assumption may not hold: for example in the EMP relation, the older employees may tend to make more than the younger ones, causing the tuples to cluster along the diagonal of the grid. To take this type of situation into account, a weight can be associated with each grid cell and the load balancing constraints of the problem can be modified so that the *weighted* sums of the grid cells assigned to each processor are specified. This last variation, of course, makes even the the computation of a feasible solution an NP-complete problem.

A.5 Target diversities

In some cases the database designer does not wish to minimize diversity, but rather seeks to achieve a target diversity for each grid slice in order to minimize response time for the corresponding query. The techniques for minimizing diversity may be used to achieve this goal if the target diversities are larger than the minimum possible. The heuristic described in this thesis may be used to construct a nearly minimum diversity solution. By appropriate modifications to this solution, the slice diversities may be increased to approximate their targets. In the extreme case where each target diversity is N, *i.e.*, the total diversity is to be *maximized*, the problem is relatively easy to solve. For example, in any d-dimensional grid where N divides each of the grid dimensions, it is possible to achieve a diversity of N in every slice of the grid by placing each processor along a grid diagonal.

A.6 More general variations

The database application involves striking a compromise between the two conflicting goals of diversity minimization and load balancing. Up to this point the problem has been stated in terms of minimizing the diversity subject to (approximately) balancing the load. Another way to approach the problem is to minimize the difference between the loads subject to the constraint that the diversity is below a specified limit – in effect switch the roles of objective function and constraint. Two more variations on this theme are:

- find a solution where the loads differ by at most some specified tolerance and the diversity is at most some specified limit (*i.e.*, two constraints and no objective)
- minimize some function of the load differences and the diversities (*i.e.*, an unconstrained problem)

In this last type of problem, the objective function should be chosen so that its minimizers provide good compromises between the conflicting goals of the application.

Appendix B

Integer Programming Models for Diversity Minimization

The diversity minimization problem may be formulated as a fully dense transportation problem with integrality constraints and an objective function which is the sum of a large number of fixed-charge functions, or as a mixed integer linear program. Let d denote the number of dimensions of the given grid, C denote the number of grid cells, s denote the number of $(d-1)$-dimensional slices in the grid, and N denote the number of processors in the problem instance.

The transportation network corresponding to an instance of diversity minimization has N sources each with supply equal to the floor or ceiling of the number of cells divided by the number of processors to be assigned, and C sinks each with a demand of one, and arcs from each source to each sink. Let $x_{i,j}$ be a zero-one variable denoting the flow on the arc from source i to sink j, let $y_{i,s} := \sum_{j \in \text{slice } s} x_{i,j}$ denote the total flow from processor i into the cells of slice s (the y's are not variables in the formulation, rather they appear as convenient shorthand), and let

$$f_{i,s}(x) := \begin{cases} y_{i,s} & \text{if } y_{i,s} \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

be a fixed-charge function which has value 0 if processor i does not appear in slice s and value 1 if it does. Then diversity minimization may be formulated as follows:

$$\begin{array}{rl} \min & \sum_{i,s} f_{i,s}(x) \\ \textit{such that} & \sum_j x_{i,j} = \text{supply}_i \, \forall i \\ & \sum_i x_{i,j} = 1 \, \forall j \\ & x_{i,j} \in \{0,1\} \, \forall i \, \forall j \end{array}$$

The fixed-charge functions may be replaced by a linear objective function by introducing extra variables and constraints in the formulation. Define a new continuous variable $z_{i,s}$ for each pair of processor i and slice s and add the constraints

$$z_{i,s} \geq x_{i,j} \forall j \in \text{ slice } s.$$

The diversity for slice s is then bounded above by $\sum_i z_{i,s}$ and the total diversity is at most $\sum_s \sum_i z_{i,s}$. Minimizing $\sum_s \sum_i z_{i,s}$ will then minimize total diversity.

The number of variables in the nonconvex transportation formulation is CN (there is a decision variable for each pair of processor and grid cell), and the number of constraints (not counting integrality constraints) is $N+C$. In the mixed integer LP, there are SN additional continuous variables, and an additional CdN constraints.

Thus, in the two-dimensional "square" case of an $N \times N$ grid of cells to be assigned to N processors, there are N^3 0-1 variables, N^2+N network constraints, and $2N^2$ fixed-charge terms in the objective function. Note that the number of feasible solutions to this problem is

$$\binom{N^2}{N}\binom{N^2-N}{N}\binom{N^2-2N}{N}\cdots\binom{N^2-(N-2)N}{N} = \frac{(N^2)!}{(N!)^N}.$$

The number of optimal solutions is at least $N!^3$, since any optimal solution may be transformed into an alternate optimal solution by a row permutation followed by a column permutation followed by a permutation of the processor indices.

Specifically, in the case of $N = 5$, there are more than 623×10^{12} feasible solutions and at least 1,728,000 optimal solutions. The difficulty of treating even a small problem like this by conventional optimization techniques is illustrated by an attempt to solve small problems via the GAMS ZOOM module: the largest $N \times N$ problem that was solvable in under an hour on a workstation was the 5×5 problem and the solution time grew exponentially with N. In contrast, the genetic algorithm described below was able to solve to optimality a problem with $N = 101$ in about 70 seconds on a 32-node partition of a CM-5.

Appendix C

A Diagonal-based Solution Technique

This technique was developed by Schultz and produces minimum diversity solutions for $N \times N$ grids by successively assigning cells on the grid diagonals. Given an integer $\rho \in \{1, \ldots, P\}$, the technique produces a solution so that there are exactly ρ processors in each row and $\lceil N/\rho \rceil$ processors in each column. We then show that the resulting solution is optimal for this problem if $\rho = \left\lceil \sqrt{N} \right\rceil$,

A Diagonal Solution Technique

Define the jth diagonal of the grid to be the set of cells (i, k) that satisfy

$$k - i \equiv j \, mod \, N. \tag{6}$$

Note that there are exactly N cells in each diagonal. An algorithm for computing an assignment is given in figure 26. This algorithm is given inputs N and $\rho \in \{1, \ldots, N\}$ and assigns one diagonal at a time, using the procedure assign.single.diag. The procedure assign.single.diag is given an "initial processor

```
Procedure assign.single.diag($p, j$)
    $q \leftarrow p$        ( initial processor index is $p$ )
    $k \leftarrow j$        ( initial column index is $j$ )
    For $i = 0, \ldots, N-1$
        cell($i, k$) $\leftarrow q$
        $k \leftarrow (k+1) \, mod \, N$
        $q \leftarrow (q+1) \, mod \, N$


Algorithm assign.diags($N, \rho$)
    For $j \leftarrow 0, \ldots, N-1$
        assign.single.diag($j \, mod \, \rho, j$)
```

Figure 26: The diagonal-based algorithm

index" p, and a column coordinate j, and then assigns each cell in the jth diagonal to a different processor. Note that the algorithm assign.diags takes $O(N^2)$ operations, which is optimal since it makes each cell assignment in constant time. Figure 27 shows the portion of the assignment created by assign.diags(7, 3) after the first two calls to assign.single.diag. Figure 28 shows the complete assignment.

0						
	1					
		2				
			3			
				4		
					5	
						6

(a)

0	1					
	1	2				
		2	3			
			3	4		
				4	5	
					5	6
0						6

(b)

Figure 27: (a) After 1 diagonal assigned; (b) After 2 diagonals assigned

0	1	2	0	1	2	0
1	1	2	3	1	2	3
4	2	2	3	4	2	3
4	5	3	3	4	5	3
4	5	6	4	4	5	6
0	5	6	0	5	5	6
0	1	6	0	1	6	6

Figure 28: An optimal assignment of $\mathbb{Z}^2_{(7,7)}$ with 7 processors

Appendix D

Elongated Grids

We say that a d-dimensional grid is *elongated* if the size of one dimension dominates the sizes of the others by at least a factor of N. In citeGMSY we showed that there is a closed form optimal solution for such problems when all the $\mathcal{A}_p$ are equal. This result follows from the fact that the total diversity may be written as a linear function of the average diversities in each dimension, and that the product of these average diversities is at least N^{d-1}.

Let $\overline{\mathcal{D}}_d$ denote the average diversity over all the slices in dimension d and let the dimensions of the grid be $M_1 \times M_2 \times \cdots \times M_d$. Then the total diversity is given by

$$\sum_d M_d \overline{\mathcal{D}}_d$$

Since each $\overline{\mathcal{D}}_d$ must be between 1 and N, and (from the previously mentioned result) $\prod_d \overline{\mathcal{D}}_d \geq N^{d-1}$, a lower bound on diversity is given by the solution to the nonlinear optimization problem (with variables $\overline{\mathcal{D}}_d$):

$$\min \sum_d M_d \overline{\mathcal{D}}_d \mathit{such\ that}\ 1 \leq \overline{\mathcal{D}}_d \leq N \text{ and } \prod_d \overline{\mathcal{D}}_d \geq N^{d-1}.$$

When one dimension dominates the others by a factor of at least N, then the optimal solution of the above problem has $\overline{\mathcal{D}}_d = 1$ for the dominating dimension and $\overline{\mathcal{D}}_d = N$ for the others. If N divides the dominating dimension, this bound can be achieved by dividing the grid into N blocks along the dominating dimension and assigning one block to each processor. The resulting optimal solution has diversities of 1 for the slices in the dominating dimension and diversities of N for all other slices.

Bibliography

[1] K. Bennett, M. C. Ferris, and Y. E. Ioannidis. A genetic algorithm for database query optimization. In *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufman, 1991.

[2] D. Bitton, D. DeWitt, and C. Turbyfill. Benchmarking database systems: A systematic approach. In *Proceedings of the 1983 VLDB Conference*, October 1983.

[3] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez. Prototyping Bubba, a highly parallel database system. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.

[4] M. Cao and M. C. Ferris. Genetic algorithms in optimization. *The Journal of Undergraduate Mathematics and its Applications*, 12(1):81–90, 1991.

[5] R. DeLeone and M. A. Tork-Roth. Massively parallel solution of quadratic programs via successive overrelaxation. Computer Sciences Technical Report 1041, University of Wisconsin - Madison, Madison, WI, August 1991.

[6] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.

[7] L.E. Dickson. *History of the Theory of Numbers.* Chelsea, 1971.

[8] S. Ghandeharizadeh. *Physical Database Design in Multiprocessor Database Machines.* PhD thesis, University of Wisconsin - Madison, 1990.

[9] S. Ghandeharizadeh. *Physical Database Design in Multiprocessor Systems.* PhD thesis, University of Wisconsin - Madison, 1990. Computer Sciences technical report #964.

[10] S. Ghandeharizadeh, G. L. Schultz, R. R. Meyer, and J. Yackel. Optimal balanced assignments and a parallel database application. Computer Sciences Technical Report 986, University of Wisconsin - Madison, Madison, WI, December 1990.

[11] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[12] G. Graefe. Volcano: An extensible and parallel dataflow query processing system. Computer science technical report, Oregon Graduate Center, Beaverton, OR, June 1989.

[13] W. D. Gropp and D. E. Keyes. Domain decomposition methods in computational fluid dynamics. Technical Report 91-20, ICASE, February 1991.

[14] G.H. Hardy, J.E. Littlewood, and G. Polya. *Inequalities.* Cambridge, 1959.

[15] P. Helman. A family of NP-complete data aggregation problems. *Acta Informatica*, 26:485–499, 1989.

[16] A. L. Rosenberg. Encoding data structures in trees. *JACM*, 26(4):668–689, October 1979.

[17] R.J. Schalkoff. *Digital Image Processing and Computer Vision.* John Wiley & Sons, Inc., 1989.

[18] Tandem Performance Group. A benchmark non-stop SQL on the debit credit transaction. In *Proceedings of the 1988 SIGMOD Conference*, Chicago, IL, June 1988.

[19] Thinking Machines Corporation. *The Connection Machine CM-5 Technical Summary*, October 1991.

[20] J. Yackel and R. R. Meyer. Minimum-perimeter domain decomposition. Computer Sciences Technical Report 1078, University of Wisconsin - Madison, Madison, WI, February 1992.

[21] J. Yackel and R. R. Meyer. Optimal tilings for parallel database design. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 293–309. North-Holland, 1992.