SAND2003-3702
Unlimited Release
Printed October 2003

Algorithmic Support for Commodity-Based Parallel Computing Systems

Vitus J. Leung and Cynthia A. Phillips
Discrete Algorithms & Mathematics Department
Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM 87185-1110

Michael A. Bender
State University of New York
Stony Brook, NY 11794

David P. Bunde
University of Illinois
Urbana, IL 61801

Abstract Follows

Abstract

The Computational Plant or Cplant is a commodity-based distributed-memory supercomputer under development at Sandia National Laboratories. Distributed-memory supercomputers run many parallel programs simultaneously. Users submit their programs to a job queue. When a job is scheduled to run, it is assigned to a set of available processors. Job runtime depends not only on the number of processors but also on the particular set of processors assigned to it. Jobs should be allocated to localized clusters of processors to minimize communication costs and to avoid bandwidth contention caused by overlapping jobs.

This report introduces new allocation strategies and performance metrics based on space-filling curves and one dimensional allocation strategies. These algorithms are general and simple. Preliminary simulations and Cplant experiments indicate that both space-filling curves and one-dimensional packing improve processor locality compared to the sorted free list strategy previously used on Cplant. These new allocation strategies are implemented in Release 2.0 of the Cplant System Software that was phased into the Cplant systems at Sandia by May 2002.

Experimental results then demonstrated that the *average number of communication hops* between the processors allocated to a job strongly correlates with the job's completion time. This report also gives processor-allocation algorithms for minimizing the average number of communication hops between the assigned processors for grid architectures.

The associated clustering problem is as follows: Given n points in $\Re^d$, find k points that minimize their average pairwise L_1 distance. Exact and approximate algorithms are given for these optimization problems. One of these algorithms has been implemented on Cplant and will be included in Cplant System Software, Version 2.1, to be released. In more preliminary work, we suggest improvements to the scheduler separate from the allocator.

Acknowledgments

Michael Bender acknowledges support from HRL Laboratories, NSF Grant EIA-0112849 and CCR-0208670, and Sandia National Laboratories. David Bunde is supported in part by Sandia National Laboratories and NSF grant CCR-0093348.

Contents

List of Figures

List of Tables

1 Introduction

As part of the Advanced Simulation and Computing Initiative[1] [65], the Department of Energy Laboratories are purchasing a sequence of increasingly-powerful custom supercomputers. In a parallel effort to increase the scalability of commodity-based supercomputers, Sandia National Laboratories is developing the Computational Plant or Cplant [17, 96]. Although Sandia maintains a diverse set of computing resources, the tools for managing these resources commonly rely on scheduling/queuing software such as NQS [28] or PBS [82] to determine which of the available jobs should be run next. This decision is based on a variety of factors largely motivated by fairness and policy enforcement such as the job owner's past use of computing resouces, number of processors requested, running-time estimates, waiting time, and even day of week and time of day. When a job is selected to run, the allocator assigns it to a set of processors, which are exclusively dedicated to this job until it terminates. Until recently this decision was *not* based on the locations of the free processors: The allocator simply verified that a sufficient number of processors were free before dispatching a job.

To obtain maximum throughput, the processors allocated to a single job should be physically near each other to minimize communication costs and to avoid bandwidth contention caused by overlapping jobs. Processor locality is particularly important in commodity-based supercomputers, which typically have higher communication latencies and lower bandwidth than supercomputers with custom networks. For example, in Cplant supercomputers, where the switches form two or three-dimensional meshes with toroidal wraps in one or more dimensions[2], a good processor allocation forms a rough subcube of switches. Experiments on Cplant showed that poor allocation could increase the running time of a pair of high-communication jobs by as much as a factor of two.

This report describes resource-allocation algorithms to optimize processor locality in Cplant and other supercomputers. For the problem addressed in this report, the allocator has no control over the scheduler. Given a stream of jobs from the scheduler, we wish to allocate processors to maximize processor locality. More precisely, we address the following problem. Each parallel job j has an *arrival time* a_j (the time when it is dispatched from the scheduler for processor allocation), a requested *number of processors* p_j, and a *processing time*. The user submits an estimated processing time. The true processing time is known when the job completes. The user's job may get terminated if it does not complete by the estimated processing time.

The jobs arrive *online*, that is, job j is only known to the allocator after the time a_j when it is dispatched from the scheduler. *Preemption* and *migration* are not allowed, that is, once a job is begun it must be executed to completion on the same set of processors. The objective is to assign a set of processors to each job to optimize some global measure of locality. For example, if the machine is a mesh, we may choose

[1] This program was originally called the Accelerated Strategic Computing Initiative

[2] Some of the oldest systems have more highly-connected irregular topologies.

to optimize the average expansion of the bounding box, i.e. the ratio of the bounding box for the allocated processor set to the smallest possible bounding box.

The remainder of the report is organized as follows. Section 2 summarizes the importance of processor locality. Section 3 describes related work. Section 4 describes our allocation strategies given a processor ordering. Section 4 also defines a new locality measure motivated by this work. Experiments demonstrate that the *average number of communication hops* between the processors allocated to a job strongly correlates with the job's completion time. See for example, Figure 4(a), which is reproduced from [68].

Section 5 gives processor-allocation algorithms for minimizing the average number of communication hops between the processors assigned to a job on a grid architecture. In Section 6 we give the solution to the continuum version of this allocation problem which gives insight into the discrete problem and the properties of a good processor allocation. Section 7 summarizes our suggested improvements to the scheduler separate from the allocator. We conclude with Section 8.

2 Processor Locality

Shortest-path routes should give the best communication performance when Cplant nodes are allocated contiguously. In particular, they minimize interjob message contention because most of a job's messages are routed completely within the piece of the machine allocated to that job. However, shortest path routes were only guaranteed on Siberia between processors that are at most three switch hops apart. Thus only jobs using up to forty-eight contiguous nodes were guaranteed shortest-path routes. An analysis of job data provided by the PBS scheduler showed that only about forty percent of the jobs on Siberia are forty-eight nodes or fewer in size. After a thorough analysis of routing on Cplant, we developed general shortest-path routes for Antarctica which were incorporated in a preproduction release.

These new routes seem to improve Antarctica's performance independent of the allocation scheme, and make message-routing behavior more intuitive and predictable. The job-placement problem becomes easier to approximate, both theoretically (for automated allocators, see Sections 4 and 5) and in practice when users are allowed to choose processor allocations by hand.

We also experimented with job placement on Siberia, the Cplant machine for which the most documentation was available. We found that contiguous node allocation with low aspect ratio resulted in the best performance for communications-intensive jobs. The results seem to generalize to Antarctica, one of the current Cplant machines. Siberia was dismantled shortly after our experiments.

We now provide some details of these results with respect to the consequences of routing and processor allocation.

2.1 Routing

The router is a crucial factor in the difficulty of the resulting node allocation problem. The interprocessor routes generated for Siberia by the Myricom program simple_routes are not shortest paths. In the worst case, some processors that are only four switch hops apart have routes that are fourteen switch hops long. Instead of traversing the torus in the short directions, two hops along each axis, they traverse the torus in the long directions, eight and six hops. This is a result of the Siberia switch numbering and the Up*/Down* algorithm simple_routes uses to build routes. It would be interesting to investigate the general effect of node and switch numberings on the routes generated by the Up*/Down* routing software.

Longer paths result in increased message latencies from both additional physical delays and additional message traffic. Experiments on Siberia indicate that both the lengths of the routes and the volume of message traffic can affect the performance of message-intensive jobs. An increase of ten hops can result in an increase of twenty to eighty percent in the time to complete a message intensive job depending on the volume of other message traffic.

Since the length of the routes also affects the complexity of the node allocation problem, we have modified the Myricom software to bias it toward short routes. While studying simple_routes, we discovered a command line option for shortest path routes. This was undocumented and not known to the Cplant developers. Unfortunately, the resulting shortest path routes can deadlock.

Dally and Seitz[33] give an algorithm for deadlock-free shortest-path routing in k-ary n-cubes that we extended to Cplant. The Siberia switch topology is almost a ten-ary two-cube, and the Antarctica switch topology is almost an eight-ary three-cube. The paper's k-ary n-cubes have single unidirectional links between nodes. We must split the single links into two virtual links. Cplant has two bidirectional links between switches. A specific node (switch) and link numbering is required for the Dally-Seitz algorithm. We investigated the extension of this routing algorithm to bidirectional links. The correctness proofs for the algorithm hold for bidirectional links. We implemented the routing algorithm in the Dally and Seitz paper with the straightforward extension to bidirectional links. The Myricom program *deadlock* has declared that the resulting routes for Siberia "cannot deadlock." Thus we have computed shortest-path routes that cannot deadlock.

The algorithm required a node/switch numbering that was different from Siberia's. We generated the routes for Siberia with the required numbering and then mapped the new numbering back to the Siberia numbering. Based on this generalization of Dally and Seitz routing for Siberia, a three-dimensional extension of our work provided shortest-path routes for Antarctica [34]. Recent experiments on Antarctica indicate that these routes may decrease completion time for some application codes by at least fifty percent over the routes generated by Myricom's simple_routes program [104].

Based on this work in Cplant routing, we were also able to have an impact on the routing for Red Storm as documented in [67]. The Red Storm routing work was funded by CSRF.

2.2 Processor Allocation

We experimented with job placement on Siberia, the Cplant machine for which the most documentation was available. The results seem to generalize to Antarctica.

We used a Paragon communications test suite. The communications test suite performs one hundred iterations of ping-pong timing, exchange timing, ring timing, broadcast timing, and all-to-all timing. We measured communication overhead when multiple applications concurrently execute on Siberia. In particular, we compared the effect of low aspect ratio versus high aspect ratio for contiguous processor allocation and we compared contiguous versus non-contiguous processor allocation.

For contiguous processor allocation, we found that low-aspect-ratio jobs completed in less than ninety percent of the time of the high-aspect-ratio jobs, see Table 1. For thirty-two nodes, the low-aspect-ratio job was run on a ring of four switches, and the high-aspect-ratio job was run on a line of four switches. For the sixty-four node job,

Number of Nodes	"Square" Placement (sec.)	"Rectangular" Placement (sec.)
32	125	141
64	308	357

Table 1: Aspect Ratio Communications Test Completion Times

the low-aspect-ratio job was run on eight out of nine switches in a three-by-three grid of switches, and the high-aspect-ratio job was run on a two-by-four grid of switches.

Number of Nodes	Contiguous Job	Single Non-Contiguous Job	Two Non-Contiguous Jobs
4	$\frac{1}{2}$ switch	$\frac{1}{4}$ of 2 switches 9 hops apart	$\frac{1}{4}$ each of 2 switches 9 hops apart
8	1 switch	$\frac{1}{2}$ of 2 switches 11 hops apart	$\frac{1}{2}$ each of 2 switches 11 hops apart
16	2 adjacent switches	$\frac{1}{2}$ of 2 pairs of adjacent switches 9–12 hops apart	$\frac{1}{2}$ each of 2 pairs of adjacent switches 9–12 hops apart
32	ring of 4 switches	$\frac{1}{2}$ of 2 rings of 4 switches 4–12 hops apart	$\frac{1}{2}$ each of 2 rings of 4 switches 4–12 hops apart
64	8 of 9 switches in 3x3 grid of switches	$\frac{1}{2}$ of 16 of 20 switches in 4x5 grid of switches	$\frac{1}{2}$ each of 16 of 20 switches in 4x5 grid of switches

Table 2: Locality Communications Test Node Allocations

In the case of contiguous versus non-contiguous processor allocation, we found that the contiguous jobs were completed in approximately sixty to ninety percent of the time of the non-contiguous jobs, depending on the number of nodes used and other jobs on the machine. (See Tables 2 and 3.) The one anomaly occurs with sixty-four nodes and a single non-contiguous job, but in that case even though the processors were not contiguous, the switches were contiguous. Furthermore, none of the jobs interfered with each other. Perhaps the communications traffic for the job was high enough that spreading it out over more switches resulted in faster completion. Moore and Ni observed the same effect in their simmulations [80]. Note that with two non-contiguous sixty-four node jobs interfering with each other, the anomaly disappeared. This anamoly is not expected to occur frequently.

Traffic load could also explain the similar results for the single and double non-contiguous thirty-two node jobs. The single job was run during user time with other jobs, but the double jobs were run during system time without other jobs, because

Number of Nodes	Contiguous Job (sec.)	Single Non-Contiguous Job (sec.)	Two Non-Contiguous Jobs (sec.)
4	25	29	44
8	50	58	67
16	73	90	108
32	125	154	156
64	308	282	415

Table 3: Locality Communications Test Completion Times

the interactive partition of Siberia was usually only thirty-two nodes. Note that the routes on Siberia most likely had an effect on these results also.

As mentioned above, the routing is a crucial factor in the difficulty of the resulting scheduling and node allocation problem. If the routes are shortest paths, we can use bin packing. If they are not, we must find minimum-weight cliques of a specific size. While both problems are NP-complete and therefore equally difficult to solve exactly, their approximability is quite different. Bin packing can be approximated to a constant factor from optimal in polynomial time [31]. The best known polynomial time approximation algorithms for the general unweighted clique problem achieve an approximation ratio of $n^{1-\epsilon}$ [16], and unless NP = coR, clique is hard to approximate within a factor $n^{\frac{1}{2}-\epsilon}$ for any $\epsilon > 0$ in polynomial time [52]. Even constructing the graph for the clique problem would be time-consuming and difficult since it would require knowledge of the exact Cplant interconnection.

3 Related Work

Initial processor-allocation algorithms allocated only convex sets of processors to a job [15, 24, 70, 106]. Doing so reduces interjob communication contention compared to random assignment, since each job's communication is routed entirely within the set of processors assigned to that job. Other researchers have shown in a variety of studies that poor interprocessor communication can reduce throughput [9, 73, 74, 81]. Unfortunately, requiring that jobs be allocated to convex sets of processors reduces the achievable system utilization to levels unacceptable for any government-audited system [63, 101].

The simulation-based investigations of Subramani et al. [101] show that fragmentation is necessary for high performance. Their work is directly motivated by the Cplant system, though some of it can be applied to more general systems. They investigated the effect on system throughput of a policy forbidding fragmentation, using trace files from the Cornell Supercomputing Center. In their simulation, they queued jobs until a set of contiguous processors were available, scaling the running times down to indicate the benefit of contiguous allocation. They determined that fragmentation must cause at least a factor-of-two slowdown in order for the benefit of completely contiguous allocation to compensate for the loss of processor utilization. Thus, any real system must allow for fragmentation.

More recent work [22, 68, 71, 75, 101] allows discontiguous allocations, attempting to cluster processors and minimize contention with previously-allocated jobs. Mache and Lo [73, 74] proposed various metrics to measure the quality of a discontiguous allocation, including average number of communication hops between processors assigned to the same job. Subramani et al. [101] investigated a strategy that allows fragmentation, motivated by the buddy strategy for memory allocation. They considered two and three dimensional meshes. The machine is subdivided geometrically. For example, two halves of the machine are a buddy pair, two quarters within the half, etc. Jobs are allocated to these predefined subblocks. Their system holds some jobs back rather than fragmenting them. This buddy approach does not directly apply to our problem because the allocator cannot ever delay jobs.

A problem closely related to Cplant processor allocation is *memory allocation*. In this problem there is an array of memory, and contiguous sub-arrays are allocated and deallocated online [72, 90, 91]. One objective is to minimize the highest memory address used and consequently the required memory size. Memory allocation differs from processor allocation because memory allocators leave empty space to guarantee contiguity and are allowed to refuse requests that do not fit contiguously.

Another related problem is online *bin packing.* In bin packing, the objective is to pack a set of items with given sizes into bins. Each bin has a fixed capacity and cannot be assigned items whose total size exceeds this capacity. The goal is to minimize the number of bins used. The off-line version is NP-hard [45] and bin packing was one of the first problems to be studied in terms of both online and

offline approximability [55, 56, 57]. Multi-dimensional bin packing, where the items and bins are hyperrectangles, has also been studied. The seminal offline and online results appear in [25, 27], while the latest results are in [97]. For a more detailed review of bin packing, see the surveys [26, 32]. Bin packing results cannot be directly applied to our problem since we have only a single "bin". Also objects can leave the system, creating multiple holes within this bin because jobs cannot migrate.

There is a large body of work on scheduling and online scheduling, in particular. We do not attempt to review all this work here, but refer the reader to the survey of Sgall [98].

3.1 One-Dimensional Reduction

Other researchers have used space-filling curves for a variety of problems. Originally, space-filling curves were introduced by Hilbert [53] and Peano [85]. Recent presentations appear in [39] and [94]. Hilbert curves have been shown to preserve several measures of "locality" [46, 79]. An alternative with better performance in two dimensions is given in [83]. Generalizations of Hilbert curves to higher dimensions are given in [1]. Specific applications include matrix multiplication [23, 43], domain decomposition [3, 49, 86], and image processing [2, 4, 59, 60, 77, 102]. They are also a standard tool in the creation of *cache-oblivious algorithms* [13, 14, 18, 44, 87, 88], which have asymptotically optimal memory performance on multilevel memory hierarchies while avoiding memory-specific parameterization.

Our work adapts several of the algorithms for one-dimensional online bin packing. A common feature of these algorithms is they keep a list of partially-filled bins. Arriving objects may be placed in one of these bins (assuming they fit) or they may be placed in a new bin, which is then added to the list. The First Fit algorithm [55] places a new object in the first bin in which it fits. Best Fit [55] places a new object in the bin whose remaining space will be smallest. When the bins and objects have integral sizes, the more complicated Sum of Squares algorithm [30] is also available. This algorithm bases its decisions on a vector N, where $N(i)$ is the number of bins with remaining size i. It places a new item in the bin which minimizes the resulting value of $\sum N(i)^2$. This allocation policy encourages a variety of sizes of unallocated regions. When the input comes from a discrete distribution, this algorithm has near-optimal behavior [11, 29].

3.2 Minimizing the Number of Communication Hops

3.2.1 Processor Allocation

The processor allocation algorithm most closely related to our work was proposed by Mache, Lo, and Windisch [75]. This algorithm, called MC, assumes that jobs request processors in a particular shape, such as a 4×6 submesh. Each free processor evaluates the quality of an allocation centered on itself. It does so by counting the

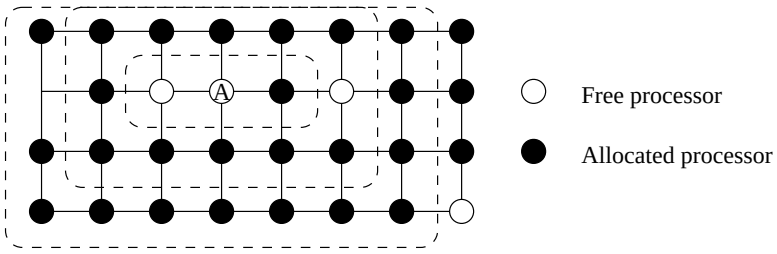

Figure 1: Illustration of MC: Shells around processor A for a 3×1 request.

number of free processors within a submesh of the requested size centered on itself and within "shells" of processors surrounding this submesh. The processors are weighted by the shell containing them; 0 for the initial submesh, 1 for the first shell out, 2 for the second, and so on. The sum of the weights gives the cost of the allocation. The algorithm chooses the allocation with lowest cost. This is illustrated in Figure 1, which is reproduced from Mache et al. [75]. The work of Krumke et al. [64] implies that MC is a 4-approximation for average pairwise distance when shell 0 is a 1×1 submesh (a reasonable choice when a desired shape is not given).

Krumke et al. consider the discrete problem of selecting a subset of k points from a set of n points to minimize their average pairwise distance. They prove a 2-approximation for the case when the interpoint distances obey the triangle inequality and prove hardness of approximation for arbitrary distances.

3.2.2 Related Problems

The maximization version of our problem, where the objective is to pick k points from a set of size n so that the pairwise distance is maximized, is called *maximum dispersion* and has been studied with the motivation of minimizing interference between the selected points. When the edge weights do not necessarily obey the triangle inequality, Kortsarz and Peleg [62] give an $O(n^{0.3885})$-approximation. Asahiro et al. [7] improve this to a constant factor in the special case when $k = \Omega(n)$ and Arora et al. [5] give a polynomial-time approximation scheme (PTAS) when both $|E| = \Omega(n^2)$ and $k = \Omega(n)$. When the edge weights obey the triangle inequality, Ravi et al. [89] give a 4 approximation that runs in $O(n^2)$ time and Hassin et al. [51] give a 2 approximation that runs in $O(n^2 + k^2 \log k)$ time. For points in the plane and Euclidean distances, Ravi et al. [89] give an approximation with performance bound arbitrarily close to $\pi/2 \approx 1.57$. When L_1 distances are used, Fekete and Meijer [40] give an optimal algorithm for fixed k and a PTAS when k is part of the input. Their PTAS is similar to one of ours given in Section 5.

Another problem related to processor allocation is MIN-SUM k-CLUSTERING, also called MINIMUM k-CLUSTERING SUM. This problem involves separating a graph into k clusters to minimize the sum of pairwise distances between nodes in the same

cluster. It differs from our problem in that we are asked to find a single cluster of a given size and our performance does not depend on the pairwise distance between processors not selected. For general graphs, Sahni and Gonzalez [95] showed that the MIN-SUM k-CLUSTERING problem is NP-hard to approximate within any constant factor for $k \geq 3$. In a metric space, the problem is easier to approximate. Guttmann-Beck and Hassin [50] give a 2-approximation using ideas somewhat similar to those presented in this paper. Indyk [54] gives a PTAS for $k = 2$ and Bartel et al. [8] give an $O(1/\epsilon \log^{1+\epsilon} n)$-approximation for general k.

4 One-Dimensional Reduction

The thesis of this section is that processor locality can be achieved in massively parallel supercomputers using simple, one-dimensional allocation strategies. This approach is applicable even when the processors are connected by irregular, higher dimensional networks. We accomplish this reduction using a space-filling curve which imposes an ordering on the network of processors such that locations near each other on the curve are also near each other in the network of processors.

In Cplant supercomputers, the switches are usually connected by two- or three-dimensional meshs with toroidal wraps in one or more dimensions, but some of the oldest systems have more highly-connected irregular topologies. We used *Hilbert curves* (also called *fractal curves*) in two dimensions and found an integer program for general networks. We present preliminary experimental results and motivating simulations for two and three dimensional meshes.

The remainder of the section is organized as follows. Subsection 4.1 gives the baseline Cplant allocation strategy which is a one-dimensional reduction. Subsection 4.2 describes our processor ordering. Subsection 4.3 describes our allocation strategies given a processor ordering. Subsection 4.4 summarizes our simulations. Subsection 4.5 describes our experimental results. Subsection 4.6 offers some concluding remarks. Some of these results were previously published in Leung et al. [68].

4.1 Baseline Cplant Allocation

Our test Cplant system, Zermatt, is a two dimensional toroidally-wrapped mesh. The Cplant Release 1.9 default allocator uses a sorted free list based on a left-to-right, top-to-bottom linear processor order. When a job j requiring p_j processors is dispatched by the scheduler, the allocator queries the system to determine which processors are free and gathers these processors into a sorted list. Job j is allocated to the first p_j processors in the list. These processors may be far apart with respect to the linear order (and the real machine), even if there is a contiguous piece of sufficient size available later in the list.

We use the Release 1.9 default Cplant system as our baseline against which to measure improvement.

4.2 Transforming to One-Dimensional Allocation

The first part of our one-dimensional allocation algorithms is to transform the allocation problem into a one-dimensional setting. As with the current Cplant node-allocation algorithms, we impose a linear ordering on the processors. We use a Hilbert curve, rather than an arbitrary order or sorting by row and column. We then allocate to obtain locality within this linear ordering.

The Hilbert curve only applies to grid topologies. We considered the problem of

finding good one-dimensional orderings for general parallel interconnection topologies and formulated this problem as an integer program.

Given a processor graph $G = (V, E)$,

$$\min \sum_{k=1}^{n} wt(k) \sum_{|j-i|=k} dist(v, w) * z_{ivjw}$$

subject to:

$$\begin{aligned}
&\sum_{v \in V} x_{iv} = 1; i = 1 \ldots n \\
&\sum_{i=1}^{n} x_{iv} = 1; \forall v \in V \\
&\sum_{v \in V} z_{ivjw} = x_{jw}; i, j \in 1 \ldots n; w \in V \\
&\sum_{i=1}^{n} z_{ivjw} = x_{jw}; j \in 1 \ldots n; v, w \in V \\
&z_{ivjw} = 0; dist(v, w) > |j - i| \\
&z_{ivjw}, x_{ij} \in \{0, 1\}.
\end{aligned}$$

Variables $x_{iv} = 1$ if vertex v is the ith vertex in the processor order and $x_{iv} = 0$ otherwise. Variables $z_{ivjw} = x_{iv} \wedge x_{jw}$ logically. If two processors' ranks in the one-dimensional ordering differ by k, then their contribution to the objective function (which we minimize) is a parameter $w(k)$ times their distance in the graph. The parameter $w(k)$ decreases rapidly (e.g., inverse exponentially) with k, so that close pairs in the linear order are coerced to be close physically. We can also use this objective function to compare different curves for a given topology.

The above integer-programming problem for computing a good one-dimensional ordering is NP-complete since it is a generalization of the Hamiltonian path (HP) problem. This problem is HP if we set $w(k) = 0$ for all $k > 1$, and $w(1) = 1$. The graph has a Hamiltonian path if and only if the integer program has a solution with an objective function value of $n-1$ where n is the number of nodes in the graph. Though the problem is NP-complete we may be able to solve particular instances optimally or to within a provable instance-specific error tolerance using PICO (Parallel Integer and Combinatorial Optimizer), a massively-parallel branch-and-bound code developed at Sandia National Laboratories and Rutgers University. PICO includes a (branch-and-cut) mixed-integer program solver. Though this computation may be time-consuming, it is performed only once for any given physical machine and choice of $w(k)$.

4.3 One-Dimensional Allocation Strategies

The second part of our algorithm is how processors from the linear order are selected. We modify existing memory-allocation and bin-packing algorithms for the

Cplant processor-allocation problem. The modification is not a straightforward generalization because it is not required (although desirable) that processors be allocated contiguously. We use analogs to bin-packing algorithms when processors can be allocated contiguously. The intervals of contiguous free processors are analogous to free space in unfilled bins. However, we must determine a different allocation strategy when there is no contiguous interval of sufficient size.

Span Metrics Our one-dimensional processor-locality metric is motivated by a linear or ring topology. Let r_p be the *rank* of processor p in the linear ordering. This will be an integer in the range $1, \ldots, |P|$, where P is the set of processors. Let M_j be the set of processors assigned to job j. The *linear span*, s_j^ℓ of these processors is the number of processors potentially involved in message propagation/delivery for job j if the processors are connected only in a line. That is, s_j^ℓ is the maximum difference in rank between any pair of processors assigned to job j (plus one): $s_j^\ell = \max_{p \in M_j} r_p - \min_{p \in M_j} r_p + 1$. All processors with ranks between this minimum and maximum rank (including the endpoints) are involved in routing a message between these two processors. These are the processors "owned" by job j plus those "trapped" between pieces of job j. The *ring span* s_j^w is a measure of locality if the processors are connected in a ring, again corresponding to the processors owned by job j and those trapped by these segments. Computationally, it is easier to determine the size of the largest "free" set of processors, accounting for the ring wraparound, and subtract it from the number of processors. Let $r_{j,i}$ be the ith-smallest rank of a processor in M_j for $i = 0 \ldots p_j - 1$. Then we define $s_j^w = |P| - \max(\max_{i=0}^{p_j-2} r_{j,i+1} - r_{j,i} - 1, |P| - 1 - r_{j,p_j-1} + r_{j,0})$. In this section, we use ring span which we call *span* and denote s_j for brevity. Span s_j is a measure of the processor locality of job j for more general topologies provided the space-filling curve closely reflects processor locality. The integer program described in Subsection 4.2 computes a processor ranking for ring span provided difference in rank is computed as the minimum distance around the ring.

In this section we test heuristic methods for span minimization. Minimizing metrics based on span is computationally difficult. Examples of such metrics include the sum of the spans of jobs ($\sum_{i=1}^n s_i$), the max of the spans of jobs ($\max_{i=1}^n s_i$), the sum (resp. max) of the spans divided by the requested number of processors ($\sum_{i=1}^n s_i/p_i$), the sum (resp. max) of the spans weighted by the processing times ($\sum_{i=1}^n s_i t_i$), etc.

Strategies When job j is dispatched, we determine if there is a contiguous interval of free processors large enough to run job j. When a job *cannot* be allocated contiguously, it is allocated across multiple intervals. We choose the allocation that *minimizes the span* of the job. In a tie we start the job at the smallest rank possible. When a job *can* be allocated contiguously, we choose which interval to use based on adaptations of one-dimensional bin-packing algorithms. We consider three strategies:

- *First-Fit Allocation* – Allocate j to the first interval that is large enough.

- *Best-Fit Allocation* – Allocate j to the interval that minimizes the number of unallocated processors remaining in the interval.
- *Sum-of-Squares Allocation* – For each interval to which j could be allocated, determine the number of intervals of each size that would remain. Allocate j to the interval that minimizes the sum of squares of these numbers of intervals.

Heuristic (total allocation and load time in seconds)	32 nodes x 4 MB	32 nodes x 6 MB	64 nodes x 4 MB	64 nodes x 6 MB
Free List (no curve)	6.58701	7.26286	7.31679	8.26224
Free List (s curve)	6.73076	7.87602	7.66293	8.92789
First Fit (no curve)	6.53511	8.25920	8.10882	8.54911
First Fit (s curve)	6.66584	7.67763	7.47365	8.76033
Best Fit (no curve)	7.01775	7.49869	7.22588	8.47155
Best Fit (s curve)	8.35122	8.83040	9.05512	9.61046
Sum of Squares (no curve)	6.68588	8.04053	7.26856	8.36683
Sum of Squares (s curve)	6.51863	7.88177	7.78887	9.18457

Table 4: Total allocation and load time for scheduled jobs.

All of these strategies are easy to implement and run quickly [92]. See Table 4 which is reproduced from the appendix of [92]. The gains in system throughput (described in Subsection 4.6) far outweigh the additional computation time of the allocator.

4.4 Simulations

To test our one-dimensional allocation algorithm, we built an event-driven Cplant simulator, which tests the allocation strategies from Subsection 4.3 on space-filling curves. The objective of the simulator is to exhibit tendencies rather than to predict running times precisely. Our simulations suggested that one-dimensional allocation strategies coupled with space-filling curves yield processor locality in higher dimensions. A variety of performance metrics gauge the processor locality.

With this simulator, we were also able to study the throughput of a fixed number of processors partitioned into various numbers of clusters [20]. The study was used to determine in part the configuration of the Sandia Institutional Cluster due to go online at the end of 2003.

Trace Files The Cplant simulator was run on traces from October, November, and December 2000. These trace files contain data about all jobs submitted to Alaska, a Cplant machine configured as a heavily augmented two dimensional mesh with 592

compute processors. The trace file includes the times that the jobs were dispatched from the scheduler, the number of processors requested, and the actual running times. These traces did not contain the processors on which the job was actually run so we cannot compute the fragmentation/runtime environment of these jobs.

From a trace it is hard to predict how the running time of the jobs would change if the allocation were different because the running times depend on factors that are hard or impossible to model. These factors include the processor allocation, the communication patterns of the jobs, the overlaps of the jobs, and the properties of the communication network.

Rather than make potentially spurious estimates about the change in the running time of the job with different allocations, our simulations hold the running times constant and use metrics based on processor locality. The assumption is that increased locality improves performance, but that the actual speed-ups should be determined through experimentation.

We transformed the traces into many inputs that model different workloads. We developed one parameterized set of inputs by increasing or decreasing the running times of the jobs by a factor that we call the *work multiple*. All the jobs were increased by this work multiple. Increasing running times makes processors busier since jobs are in the system for a longer amount of time. Note that we do not change release times so that the interaction between jobs are different. We developed a second set of parameterized inputs by duplicating jobs and perturbing the arrival times; the number of times that a job is duplicated is called the *replication factor*. The results for both types of inputs were similar, so we report only the work-multiple results.

Metrics One-dimensional metrics include the *average span* and the *average span divided by the number of processors* (*stretch-span*). Three-dimensional metrics include the *average size of a bounding box* (size of the region defined by the maximum difference between the x, y, and z dimensions of the job allocation), the *average sum of the dimensions of the bounding box*, the *average size of the bounding cube*, the *average number of connected components* per job, as well as metrics based on the maximum and sum-of-squares of these parameters as well as metrics weighted by the running times or divided by the number of processors.

Simulator The simulator assumes a single 8x8x5 grid with one processor per vertex, for a total of 320 processors. This topology is a simplification of the production Cplant architecture at the time the traces were obtained.

Our simulator models the Cplant job queue and scheduler so that the workloads are similar to those on Cplant. When a job arrives it is placed in a job queue. The job queue is sorted first by number of requested processors and then by requested processing time. (Thus, fairness between users and different night and day priorities are not modeled.) Periodically, the scheduler polls to determine which processors are free to execute jobs, and jobs are removed from the front of the queue.

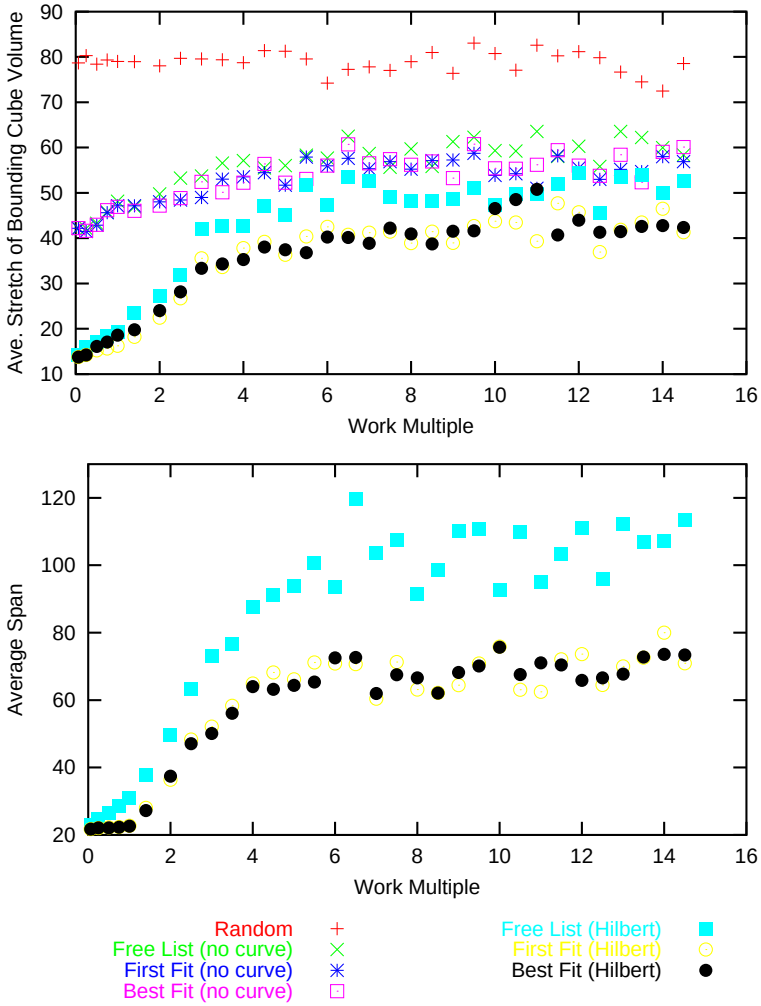

Figure 2: Comparison of one-dimensional allocation heuristics under different workloads.

Results Our results suggest that one-dimensional allocation strategies coupled with space-filling curves yield processor locality in higher dimensions. We tested a variety of performance metrics; for the sake of brevity, only a few representative results appear in Figure 2. The x-axes plot the work multiple, where we simulate the strategies on a range of workloads. The top plot gives the resulting average bounding cube divided by the smallest bounding cube. The bottom plot gives the resulting average span of jobs.

We do not know how much the increased locality speeds up the running time. However, the work-multiple parameterization demonstrates that as workloads increase, it becomes harder to obtain processor locality and as workloads decrease it becomes easier. Thus, as the locality of the jobs improves, the running time decreases which further decreases the load, thus further decreasing the running time.

The overall trend is that the processor locality improves through our approach. The simulation results were sufficiently promising to justify implementing the allocation algorithms on Cplant. The gains in system throughput (described in Section 4.5) are consistent with these simulation results.

4.5 Experiments

We have performed a limited number of experiments on Zermatt, a 128-processor Cplant machine configured as a two dimensional mesh. This development machine has an 8×8 mesh of switches with toroidal wraps in both dimensions. Four of the rows have four processors per switch. The other rows contain no compute processors; they contain service and I/O nodes, but fewer than four per switch on average. This pilot study serves as a proof of the concept: careful one-dimensional ordering and allocation to preserve locality within this ordering both improve system throughput.

All runs use identical job streams containing replicas of various-sized instances of a single communication test suite. The communication test suite contains all-to-all broadcast, all-pairs ping-pong (message sent in each direction), and ring communication. Each communication test is repeated a hundred times in each suite. The suite computes a variety of statistics, whose computation consumes a small fraction of the total running time. Because locality is most important for jobs with high communication demand, this test suite represents a best-case scenario for the benefits of allocation improvements.

Our test job stream had 91 jobs of size 2, 33 jobs of size 5, 31 jobs of size 25, and 33 jobs of size 30. This gives a small range of "large" (approximately 1/4 or 1/5 of the machine) and small jobs. The stream starts with some large jobs to fill up the machine. Small jobs are interspersed among the large ones to cause fragmentation. The last job submitted is small, but it always finishes in front of the last large job. The machine is busy through the release of the last job.

Running times on the Cplant system are nondeterministic. If we run the same job stream twice with the same allocation algorithm, same job ordering, same release times, starting from an empty machine, and having dedicated processors, the running times are not the same. Cplant has inherent nondeterminism in the network. There is variability in time to load executables, in message delivery times, and so on. If the completion time of a single job changes, the options available for the allocation of subsequent jobs also changes. This effect propagates so that later jobs can be allocated significantly better or worse than in a previous run. We even see different job execution orderings, when a job that is held up for insufficient free processors in one run finds enough free processors in a different run. We found that this nondeterminism did not significantly affect the makespan of the job stream,[3] but the running times of individual job types did vary by 4-16%.

[3]The makespan of a set of jobs is the time between the start of the first job and the completion of the last job.

We ran the job stream two to five times (an average of four) for each of the following strategies: First Fit and Sum of Squares with the Hilbert curve, and Free List and Best Fit with and without the curve.

Allocation Strategy	Average Makespan	Standard Deviation
Free List (no curve)	5:46:31	0:10:10
Best Fit (no curve)	5:27:58	0:05:48
Free List (Hilbert)	4:58:52	0:07:37
Sum of Squares (Hilbert)	4:32:09	0:03:16
First Fit (Hilbert)	4:30:22	0:06:09
Best Fit (Hilbert)	4:25:23	0:03:00

Table 5: Effect of allocation policy on the makespan of the test stream

Table 5 shows the effect of the allocation algorithm on the makespan of the job stream. For this particular job stream, it is better to use a space-filling curve than the row-based ordering. It is also better to pack a job into a consecutive interval if possible. However, the performance of the various bin-packing-based allocation strategies were largely indistinguishable.

Figure 3 shows the waiting times of the 30 node jobs as a function of their order in the job stream. The x-axis shows order of job release. The y-axis shows waiting time. The baseline points use the default processor ordering. All other runs are for the indicated algorithm with the Hilbert curve. Jobs of size 2, 5, and 25 are not represented since these would all be near the line $\boldsymbol{y} = \mathbf{0}$.

Recall the job stream is identical for all runs, so job order is identical across runs. Wait time measures the amount of time a job sits in a queue waiting for a sufficient number of free processors. This plot does not include the 2-node, 5-node, and 25-node jobs. Their wait time was so insignificant compared to that of the 30-node jobs that they all sit near the x axis. This figure shows that waiting time is yet another metric that orders the methods the same way with substantial separation.

Figure 4 examines job completion time as a function of two job-fragmentation metrics, one inherent to the topology of the job placement and one used by the algorithms. A natural geometric fragmentation metric is the average of the number of communication hops between processors allocated to a job. Figure 4(a) plots job completion time as a function of this average for the 30-node jobs. Figure 4(b) is a similar plot for span with the Hilbert curve. We do not include the 2-node, 5-node, and 25-node jobs in these plots. The 2-node, 5-node, and 25-node jobs differ enough from the 30-node jobs to add noise to the plots. When the 2-node, 5-node, and 25-node jobs are plotted by themselves, they show the same weak correlation on a different scale and with a different slope. Figure 4(a) includes all processor orderings. Figure 4(b) is for Hilbert curve only. These plots include all 30-node jobs placed with

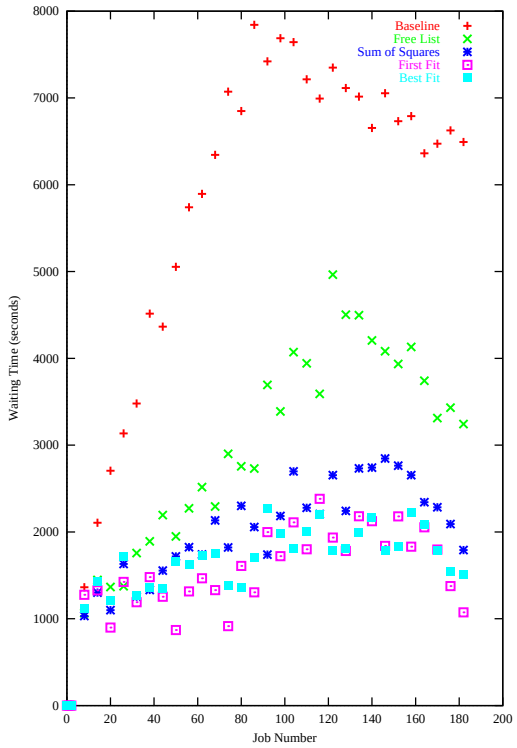

Figure 3: Individual job waiting times for different one-dimensional allocation heuristics

all algorithms since the effect of fragmentation should be a function of the amount of fragmentation and independent of how that placement decision was made.

We observe a weak correlation for both metrics. As expected, there is a stronger correlation of completion time to the average number of communication hops because this is a closer match to the topology of the job placement. We are encouraged that the general span metric, which can be easily computed, still tracks this correlation, albeit more weakly. We show the similar plot for bounding box perimeter that gives an intermediate strength correlation in Figure 5. None of these metrics captures the full environment in which a job is run.

4.6 Concluding Remarks on One-Dimensional Reduction

We are cautiously optimistic that the simple, general allocation methods discussed in this section will improve the performance of Cplant systems and apply to more general systems. Our experiments support the use of span as a fragmentation metric for the design of algorithms and as a measure of locality. Jobs with large span do generally take longer. However, the relationship between span and completion time is not very tight. More work is needed to determine how much of this variability is inherent in the problem and how much results from the imprecision of using span.

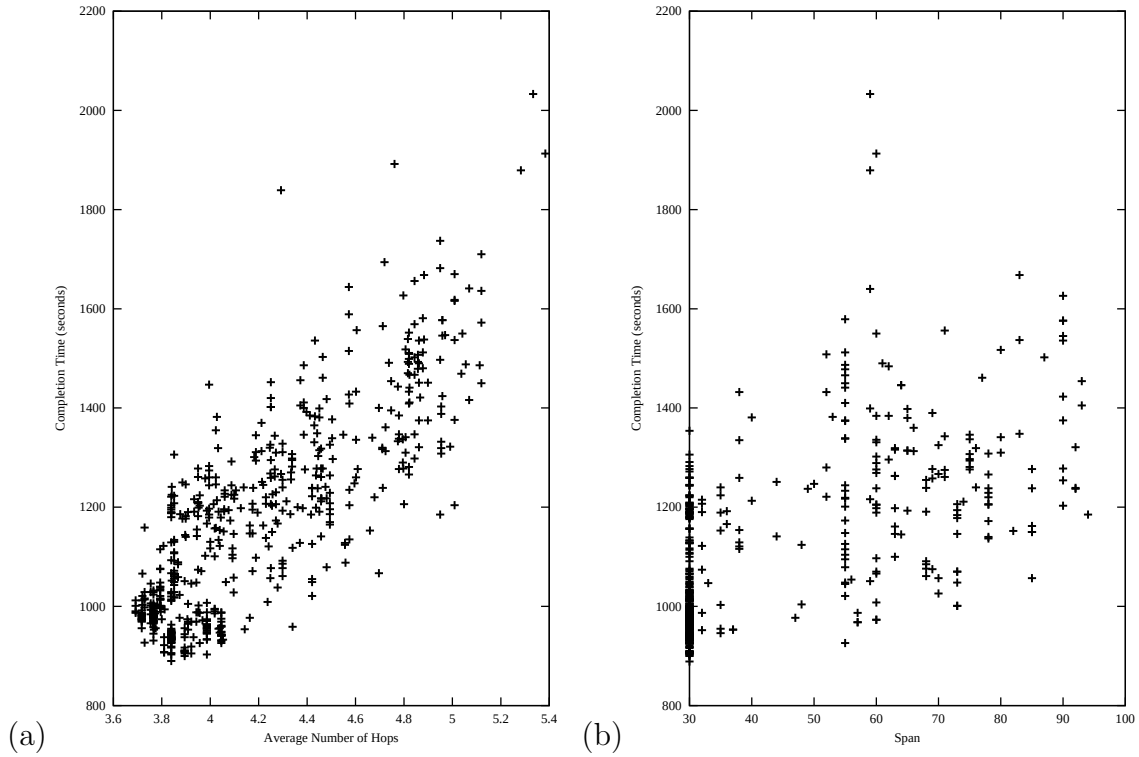

Figure 4: Correlation of span and average pairwise distance with completion time.

We also think that finding the minimum span for a given machine and set of jobs is an interesting theoretical problem. It is related to, yet distinct from, well-studied problems such as memory allocation and bin packing. We have a simple reduction to show that finding the exact minimum span is NP-hard, but do not yet know if it is approximable.

Theorem 1 *Given a set of jobs, it is NP-hard to determine offline if they can be allocated with maximum span at most k.*

Proof: Our proof closely follows the proof that dynamic memory allocation is NP-hard by Stockmeyer [100]. We give a reduction from 3-PARTITION: Given a set of positive integers $A = \{a_1, a_2, \ldots, a_{3m}\}$ with $\sum_{i=1}^{3m} a_i = mB$ and $B/4 < a_i < B/2$ for all i, can A be partitioned into disjoint sets $A_1, A_2, \ldots, A_m$ such that $\sum_{x \in A_i} x = B$ for each A_i? This problem is known to be strongly NP-hard [45], meaning that it remains NP-hard if the input size is taken to be the value of the numbers rather than their length in binary.

The basic outline of our proof is to force the algorithm to place "divider" jobs every B processors. Then, if a job arrives corresponding to each a_i and requesting a_i processors, a contiguous allocation of these jobs corresponds to a solution to the 3-PARTITION instance.

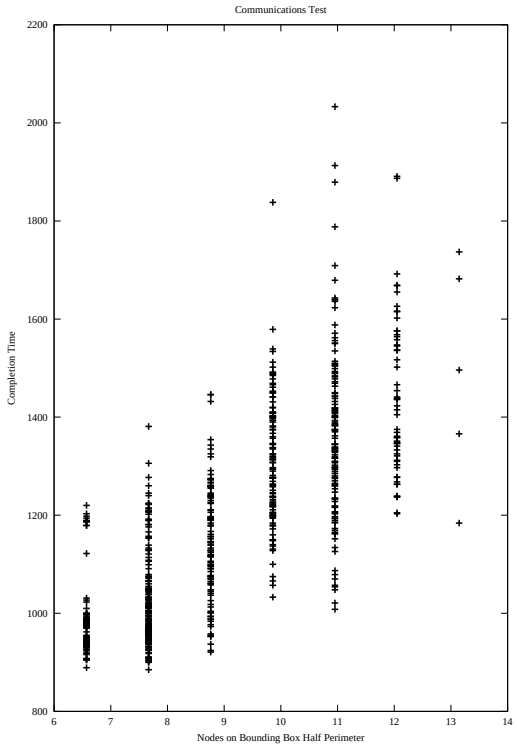

Figure 5: Correlation of bounding box half perimeter with completion time.

Rather than using large jobs, however, all jobs in our instance require 1 or 2 processors. The idea is to replace a job using k processors with $2k-1$ jobs. First k jobs $J_1, J_2, \ldots, J_k$ requesting a single processor arrive. Then job J_1 finishes and the following happens for $i = 2, 3, \ldots, k$:

- Job J_i finishes.
- A job J arrives requesting 2 processors.
- Job J finishes.
- A job J'_{i-1} arrives requesting 1 processor.

Finally, a job J'_k arrives requesting 1 processor. The 2-processor jobs force adjacent-numbered jobs to be allocated adjacently if the span is to be at most 2. Thus, after these procedure the order the jobs are allocated in is either $J'_1 J'_2 \ldots J'_{k-2} J'_k J'_{k-1}$, $J'_1 J'_2 \ldots J'_k$ or the reverse of one of these orders. We say that a set of jobs are allocated *roughly sequentially* if they are in one of these orders.

Using this idea, it is easy to follow our proof outline on a machine with $mB+m-1$ processors. First, force the algorithm to allocate $mB+m-1$ jobs sequentially. Then cause all jobs to finish except those in positions $B, 2B+1, 3B+2, \ldots, (m-1)B+m-2$.

(Note that the same jobs are in each of these positions in each of the roughly sequential orders.) Next, mB jobs arrive, each requesting a single processor. Finally, make sure that a_i of these are in roughly sequential order for each element $a_i \in A$.

This instance can be allocated with maximum span 2 exactly when the elements of A can be partitioned into groups of size B, i.e. exactly when the given instance of 3-PARTITION has a solution. ∎

Since splitting a job of size 2 yields a span of at least 3, this proof shows that it is NP-hard to give a 3/2 approximation to the maximum span. The same proof also proves the following:

Theorem 2 *Given a set of jobs, it is NP-hard to determine offline if they can be allocated with sum of spans at most k.*

We have also studied these problems in the online setting, where the standard (worse-case) model is competitive analysis [99]. We have been able to show that no online algorithm for minimizing maximum span can achieve a competitive ratio better than $\Omega(n)$ even for randomized strategies.

Theorem 3 *No online randomized algorithm for minimizing maximum span can achieve a competitive ratio better than $\Omega(n)$.*

Proof: By Yao's Lemma [105], it is sufficient to show that no deterministic algorithm can be better than $\Omega(n)$-competitive on a given randomized input. Now we specify an input of n jobs, which should be allocated on a machine of size $(n+1)/2$ (we assume that n is odd). Select distinct $x_1, x_2 \in \{1, 2, \ldots, (n+1)/2\}$ uniformly at random. Job J_i is schedule at time 0 and finishes at time i for $1 \leq i \leq (n+1)/2$. Each of these jobs also have a partner job. The partner for J_i with $i \neq x_1, x_2$ is scheduled at time i and finishes at time $n+1$. The partner for jobs J_{x_1} and J_{x_2} are scheduled at time x_1 and x_2 respectively, and finish at time n. The instance's final job is scheduled at time n and finishes at time $n+1$. The final job requires 2 processors and all other jobs require 1 processor.

Any deterministic algorithm will put the first $(n+1)/2$ jobs into a particular permutation. The partner jobs are then forced into the corresponding permutation since there is only one available processor when each is scheduled. The final job must be allocated to the two processors used by the partners of J_{x_1} and J_{x_2}. Since x_1 and x_2 were chosen randomly, the expected span of this job is $\Omega(n)$. Since the optimal algorithm can allocate this job with a span of 2 by giving the initial jobs an appropriate permutation, the competitive ratio is $\Omega(n)$. ∎

Future work for one-dimensional reduction include non-greedy allocation methods for jobs that cannot be allocated a contiguous interval. In particular, Sum-of-Squares-like algorithms are more likely to leave flexibility in the allocation options for future jobs. On some Cplant machines, once a job has span of half the machine size, it

effectively consumes bandwidth across the entire machine. Our hope is that additional flexibility would allow us to avoid such situations.

Section 5 shows how we have started to improve the allocation further by considering the actual processor topology rather than working entirely within a linear ordering of the processors. When the processors are arranged as a mesh, this makes the allocation problem a multidimensional packing problem, but other processor topologies such as toruses do not have obvious analogs in the packing literature. See Section 5 for processor-allocation algorithms minimizing the average number of communication hops between the processors assigned to a job on a grid architecture.

It may also be beneficial to consider scheduling and processor allocation together. Currently the allocator is forced to allocate jobs passed from the scheduler even if these jobs must be highly fragmented. Combining these modules might allow more intelligent decisions to be made, but any replacement would need to provide other functionality of the scheduler such as preventing starvation and allocating resources fairly between users.

Our experiments were limited by the small size of our test machine and the specialized nature of the test jobs/stream. Fully rigorous testing will be very challenging because even our limited test suite required 4.5 to 6 hours per run. In order to do these runs, we must take a system away from other users. This is particularly challenging for the 1000+ node production systems. Therefore our future work will have to rely on simulation to some extent. However, these simulations must convincingly account for the effects of locality on job completion time.

Intentionally Left Blank

5 Minimizing the Average Number of Communication Hops

This section gives processor-allocation algorithms for minimizing the average number of communication hops between the processors assigned to a job on a grid architecture. Let N be the number of processors in the grid and let n be the number of free processors. If a system allocates each parallel program greedily to minimize the average number of communication hops, then the allocation problem can be expressed as the following clustering problem:

Given a set P of n points in $\Re^d$, find a subset of k points with minimum average pairwise L_1 distance. This section gives exact and approximate algorithms for minimizing this clustering metric. One of these algorithms has been implemented on Cplant, a supercomputer at Sandia National Laboratories.

In this section we present the following results:

- We give a linear-time exact algorithm for our clustering problem in one dimension.
- We give an efficient 2-approximation algorithm for our clustering problem in $\Re^d$, for any constant d. We also present lower bounds for this algorithm.
- We give an efficient polynomial-time approximation scheme (PTAS) for the clustering problem in $\Re^2$.

These results are also reported in [12]. We have formulated the exact problem as a quadratic program which can also be solve as either a semidefinite program [84] or an integer linear program [21].

The remainder of the section is organized as follows. Subsection 5.1.1 gives a linear-time exact algorithm for our clustering problem in one dimension. Subsection 5.2 gives a 2-approximation for general point sets. Subsection 5.1.2 gives fast algorithms for small clusters in the plane. Subsections 5.3 and 5.4 give PTASs in the plane. We conclude with Subsection 5.5.

5.1 Exact Algorithms for Special Cases

5.1.1 One-Dimensional Allocation

We use the following structural property of optimal solutions:

Theorem 4 *In one-dimension, there always exists an optimal solution that is convex.*

Sketch of Proof. Assume that the lemma is false. Remove a "hole" and note that this decreases the sum of pairwise distances. ∎

We first consider the one-dimensional clustering problem, in which all the available processors form a horizontal line. This one-dimensional problem is simpler than the multi-dimensional problem because in one dimension there are only a linear number of convex clusters, and by Theorem 4 there always exists an optimal convex cluster. Thus, a naïve algorithm is to test each convex cluster and choose the cluster that minimizes the average pairwise distance. This naïve algorithm trivially runs in $O(nk^2)$ time. With an $O(k)$ method for calculating pairwise distances within clusters, the running time improves to $O(nk)$.

In this section we present a linear-time dynamic-programming solution to the one dimensional problem. To begin, we require some notation. Let p_i be the position of the i-th free processor in the linear ordering. Let $\delta_{i+1} = \text{dist}(p_i, p_{i+1}) = |p_i - p_{i+1}|$ be the distance between the i-th and $i+1$-st free processors. Let P_i be the i-th convex set of processors, that is, $P_i = \{p_i, p_{i+1}, \ldots, p_{i+k-1}\}$. Let D_i denote the pairwise distances between the processors in set P_i. Thus, the optimal solution is

$$\min_{1 \le i \le n-k+1} \{D_i\}.$$

Rather than calculate D_i "from scratch," we calculate D_i quickly from D_{i-1}. To do so, we maintain the following variables:

- The *linear span* as previously defined, i.e., distance between the leftmost and rightmost processor in D_i:

$$S_i = \text{dist}(p_i, p_{i+k-1}) = \sum_{j=1}^{k-1} \delta_{i+j}.$$

- The distance from all processors in set P_i to the leftmost processor p_i:

$$L_i = \sum_{j=1}^{k-1} (k-j)\delta_{i+j}.$$

- The distance from all processors in set P_i to the rightmost processor p_{i+k-1}:

$$R_i = \sum_{j=1}^{k-1} j\delta_{i+j}.$$

The algorithm is as follows:

1. Initialize the three variables L_1, R_1, and D_1 (in $O(k)$ time).

2. Repeatedly update the variables (in $O(1)$ time) as follows:

$$\begin{aligned} L_{i+1} &= \sum_{j=1}^{k-1}(k-j)\delta_{i+j+1} = L_i - (k-1)\delta_{i+1} + S_{i+1}. \\ R_{i+1} &= \sum_{j=1}^{k-1} j\delta_{i+j+1} = R_i - S_i + (k-1)\delta_{i+k}. \\ D_{i+1} &= D_i - L_i + R_{i+1}. \end{aligned}$$

3. Return the minimum value of D_i ever seen.

The one-dimensional problem can be solved in linear time because of this efficient dynamic programming implementation:

Theorem 5 *The clustering algorithm runs in time $O(n+k) = O(n)$.*

The algorithms easily adapts to run on a one dimensional torus (a ring). For this generalization we make two copies of the points $p_1, p_2, \ldots, p_n$, so that our problem instance contains points $p_1, p_2, \ldots, p_{2n}$, where $\text{dist}(p_n, p_{n+1})$ is the distance from p_n to p_1 along the wraparound, and for $1 \leq i \leq n-1$, $\text{dist}(p_i, p_{i+1}) = \text{dist}(p_{n+i}, p_{n+i+1})$. Then we run the algorithm as described above. If the optimal solution uses wraparound, then it will be found using some of the original processors and some of the copied processors.

5.1.2 Fast Algorithms for Small Clusters

Lemma 1 *Let P be a set of n points in the plane. The smallest subset of P of size 3 (in the L_1 metric) can be found in $O(n \log n)$ time.*

Proof: Let $S = \{s_0, s_1, s_2\}$ be a minimal weight subset of P. We label the x- and y-coordinates of a point $s \in S$ by some (x_a, y_b) with $0 \leq a < 3$ and $0 \leq b < 3$ such that $x_0 \leq x_1 \leq x_2$ and $y_0 \leq y_1 \leq y_2$. The weight of S is equal to $2(x_2 - x_0) + 2(y_2 - y_0)$. Consider the smallest Steiner star of S. Its length is equal to $(x_2 - x_0) + (y_2 - y_0)$. We can use this fact to show that the smallest subset of size 3 is also the smallest Steiner star of size 3. So S can be used to form a Steiner star of size 3 of minimal weight. Let c be the center of the smallest Steiner star of S.

The three points in S are the three closest points to c. If not, there would have been a smaller Steiner star. Therefore S corresponds to a cell on the order-3 Voronoi diagram of P. Since this diagram can be found in $O(n \log n)$ time and has a linear combinatorial complexity, the lemma follows. ∎

5.2 2-Approximation for General Point Sets

Given n points in $\Re^d$, we want to find k points to minimize the average pairwise L_1 distance. Define the *median of a set of k points in* $\Re^1$ to be the "middle" point, or if k is even, either of the two middle points. Define the *median of a set of k points in* $\Re^2$ to be a point in $\Re^2$ having the median x-coordinate and the median y-coordinate of the points in the set. The median of a set of k points in $\Re^d$ is defined analogously. Thus, the median of the set need not belong to the set. If k is odd, then there is a unique median, whereas if k is even, then there are 2^d choices of median.

The algorithm is as follows:

> Build the set of all possible medians by drawing a horizontal and vertical line through every point, and consider all the $O(n^2)$ intersection points. For each possible median do:
>
> 1. Take the k points closest to the possible median.
> 2. Compute the total pairwise distance between all k points.
>
> Return the set of k points with smallest pairwise distance.

While the 2-approximation is reminiscent of the MC algorithm (Section 3 and [75]), there are crucial differences. The MC algorithm builds balls only around free processors rather than around all possible medians. The work of Krumke et al. [64] implies that MC is a 4-approximation for average pairwise distance when shell 0 is a 1×1 submesh (a reasonable choice when a desired shape is not given). Our 2-approximation is also similar to that of Krumke et al, but they consider points in a general metric space and only center the allocated processors on a free processor.

5.2.1 Analysis

For any point p and cluster of k points in $\Re^d$, define the *star from p* to be the union of the paths extending from p to the k points. The *length* of the star is the total length of these paths.

Lemma 2 *For any set of k points, the shortest star is centered at their median.*

Proof: Consider what happens to the length of the star for a set of points when we move the center to a median of the set of points. With out loss of generality, we will consider only what happens in the x-dimension. Let d equal the distance in the x-dimension we moved the center. At most half the points moved up to d further from the center, and at least half the points moved d closer to the center. ∎

The following corollary establishes a connection between the length of the star centered at the median of k points and the pairwise distances between the points.

Corollary 3 *For any set of k points, the length of the star centered at the median times k is a lower bound on the sum of the (double counted) pairwise distances between the k points.*

Proof: The sum of the (double counted) pairwise distances between the k points exactly equals the sum of the lengths of the k stars centered at each of the k points. The corollary follows directly. ∎

Corollary 3 leads to a lower bound on OPT, where distances are double counted.

Lemma 4 *The length of the shortest star times k is a lower bound on OPT.*

Proof: Consider the optimal cluster, that is, the cluster that minimizes the sum of the pairwise distances between the points. By Corollary 3, a lower bound on OPT is k times the length of the star centered at the median. Therefore, a lower bound on OPT is k times the length of the minimum star. ∎

We now give an upper bound on the cost of the cluster found by our algorithm:

Theorem 6 *The total pairwise distances, double counted, is at most $(2 - 2/k)OPT$.*

Proof: We overestimate distances by routing all paths through the median. These detours can only make the distances larger. For example, two points in the same quadrant have their distance increased by first going to the center and then going back. Thus, every distance is routed along the shortest star (from the median of the k chosen points).

How many times is each edge of the star covered? Each of the k chosen points covers its adjacent edge of the star $k - 1$ times for all the connections to the other points, plus it covers every other edge of the star (of which there are $k - 1$) once, for a total of $2k - 2$ per chosen point. Thus, the total pairwise distance, double counted, is at most $2k - 2$ times the length of the shortest star.

Therefore, we have a $(2 - 2/k)$-approximation. ∎

5.2.2 Tightness

Now we give instances which lower bound the approximation ratio achieved by our algorithm. Note that these lower bounds limit the performance of the algorithm and are not lower bounds showing the hardness of the problem in general.

Returning the smallest star is a 2-approx First we consider a slight variant of our algorithm. Rather than computing the sum of pairwise distances for the points in each star, merely return the smallest star. By the reasoning above, this algorithm is also a 2-approximation. To show that it is not better than a 2-approximation, consider an input consisting of the following 4 sets of points:

A is $k-1$ points at $(0,0)$,
B is 1 point at $(k,0)$,
C is $k/2$ points at $(-1-\epsilon, 222k)$, and
D is $k/2$ points at $(1+\epsilon, 222k)$.

The star computed from $(0,0)$ includes the points in A and B and has length k. This is smaller than $(k/2)(2+2\epsilon) = k(1+\epsilon)$, the length of the stars computed from $(0, 222k)$, $(-1-\epsilon, 222k)$, or $(1+\epsilon, 222k)$.

Thus, the algorithm selects $A \cup B$. The sum of pairwise distances for these points is $(k-1)k = k^2 - k$. The optimal set of points is $C \cup D$, which have sum of pairwise distances $(k/2)(k/2)(2+2\epsilon) = (1/2)k^2(1+\epsilon)$. For $\epsilon = 1/k^2$, the ratio between these approaches 2 as $k \to \infty$.

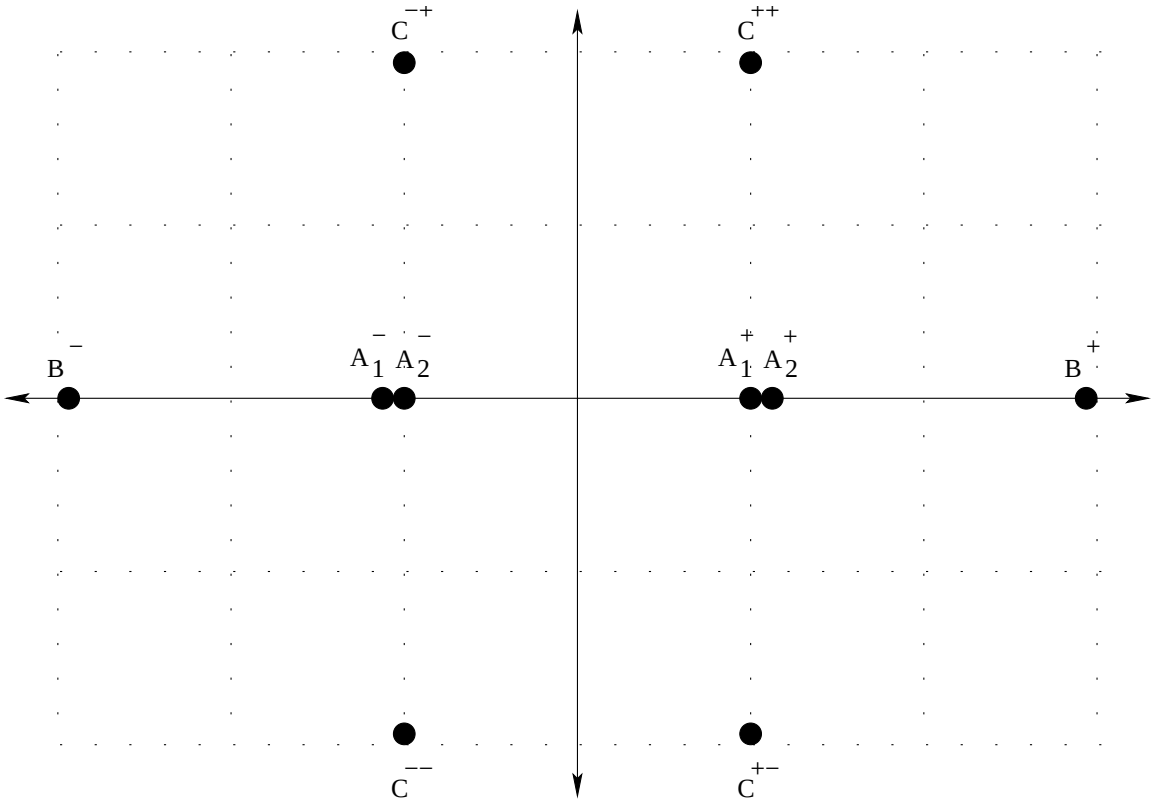

Figure 6: Lower bound for real algorithm

Lower bound for real algorithm Consider the following sets of points shown in Figure 6:

$A_1^{\pm}$ are $k/8$ points at $(\pm 1, 0)$,
$A_2^{\pm}$ are $3k/8$ points at $(\pm(1+\epsilon), 0)$,
$B^{\pm}$ are $k/8$ points at $(\pm(3-\epsilon), 0)$, and
$C^{\pm\pm}$ are $k/8$ points at $(\pm 1, \pm(2-\epsilon))$.

The optimal selection is all the A's. The best center is at any of the A points or B points, which yield $7k^2/8$, for an approximation ratio of $7/4$.

Generalizing to 3d:

$A_1^{\pm}$	are $k/12$ points at $(\pm 1, 0, 0)$,
$A_2^{\pm}$	are $5k/12$ points at $(\pm(1+\epsilon), 0, 0)$,
$B^{\pm}$	are $k/12$ points at $(\pm(3-2\epsilon), 0, 0)$,
$C_y^{\pm\pm}$	are $k/12$ points at $(\pm 1, \pm(2-\epsilon), 0)$, and
$C_z^{\pm\pm}$	are $k/12$ points at $(\pm 1, 0, \pm(2-2\epsilon))$,

which gives $\rho \geq 11/6$.

Generalizing to higher dimensions should be fairly straightforward because no additional points are added (other than those that are rotations of C).

5.2.3 Efficient Implementation Ideas

The most time-consuming step of our algorithm is finding the k nearest free processors from each free processor. In the plane, the fastest known solution is a data structure by Eppstein and Erickson [36]. It requires $O(n \log n)$ preprocessing time to build and then $O(\log n + k)$ time for each query, for a total of $O(n \log n + nk)$ to find all sets of nearest neighbors. Their algorithm is specific to finding nearest neighbors under the L_1 metric; Dickerson et al. [35] give a slightly slower solution which works for other metrics. In 3 dimensions, Vaidya [103] gives an algorithm that runs in $O(kn \log n)$. This algorithm has the same asymptotic running time in any higher fixed dimension d, but the O-notation hides a d^d term. The running time of known solutions are at least exponential in d, even when only looking for the single nearest neighbor. This has motivated efforts to find approximate nearest neighbors. An overview of exact algorithms and one approximation is given by Arya et al. [6]. Note that in our application, d is typically 3 or less because the meshes must be laid out in 3-dimensional space.

5.2.4 Other Algorithm Ideas

Picking median element This algorithm is simply to try each grid point p as a potential median element; select the k nearest processors such that p is the median of the selected processors. (To implement this, repeatedly take pairs of processors from opposite quadrants.)

The current best lower bound for this algorithm comes from the following instance shown in Figure 7:

$A_1^{\pm}$	are $k/4$ points at $(\pm 1, \pm\epsilon)$,
$A_2^{\pm}$	are $k/4$ points at $(\mp\epsilon, \pm 1)$,
$B^{\pm}$	are $k/4$ points at $(\pm(1+2\epsilon), 0)$, and
$C^{\pm}$	are $k/4$ points at $(\pm(3-\epsilon), 0)$.

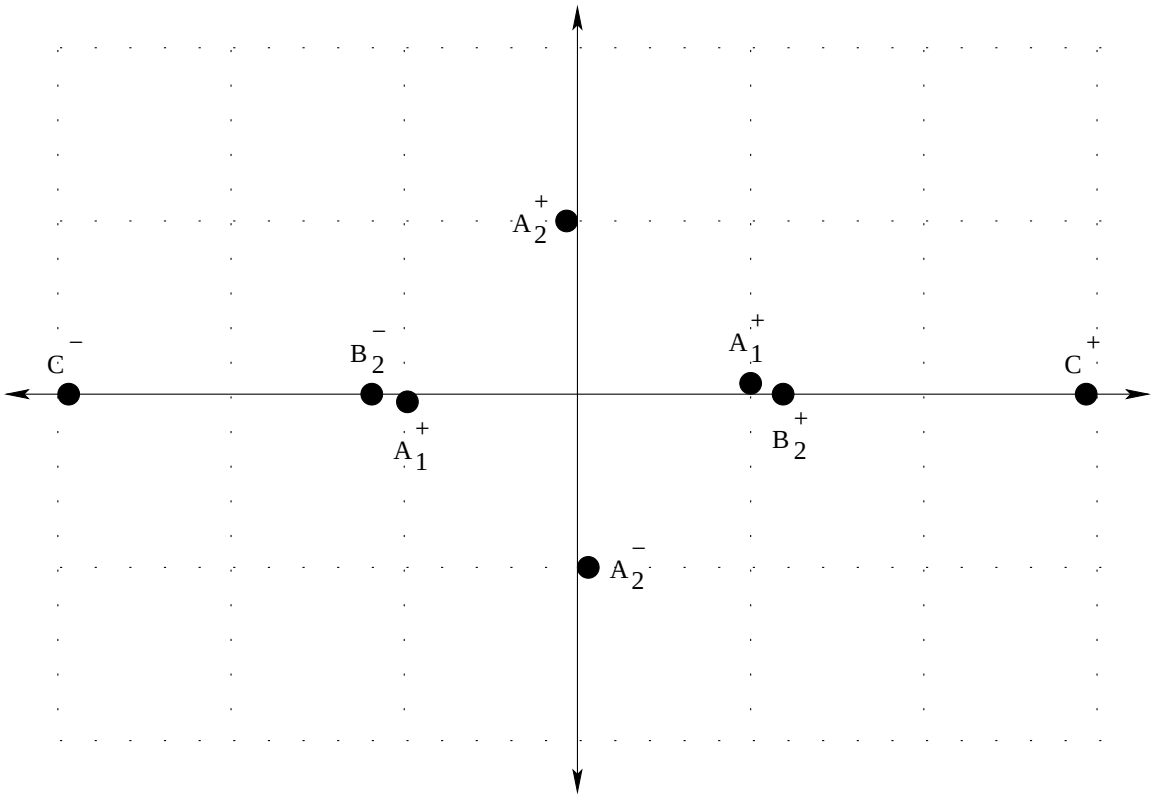

Figure 7: Lower bound for algorithm that tries each grid point as a potential median

The optimal solution is $A_1^{\pm} \cup B^{\pm}$, which gives sum of pairwise distances $k^2/2$.

The algorithm considers the following:

- all the A groups, which gives sum of pairwise distances $3k^2/4$, and
- $A_1^+ \cup A_2^+ \cup C^+ \cup B^+$ (could also get B^- or center around the A^- points), which gives sum of pairwise distances $3k^2/4$.

In both cases, the ratio is 3/2. Note that if the algorithm also considers midpoints, it can get set $A_1^+ \cup A_2^+ \cup B^+ \cup B^-$ for sum of pairwise distances $5k^2/8$ and ratio 5/4.

Figure 8: One-dimensional lower bound for algorithm that also considers the midpoints between grid points

Algorithm with Midpoints When the algorithm also considers the midpoints between grid points, it finds the optimal solutions in the examples in Subsection 5.2.2. However, it does not in the following example shown in Figure 8:

A is $k(1/4 - x/2)$ points at $-1 + \epsilon$,
B is kx points at $-\epsilon$,
C is $k/2$ points at 0, and
D is $k/2$ points at 1,

where $x = (\sqrt{(2)} - 1)/2$.

The optimal is B, C, and $k(1/2 - x)$ points from D. Ignoring epsilons, this has sum of pairwise distances $k^2 x$.

The algorithm will consider the following two sets of points:

- A, B, C, and $k(1/4 - x/2)$ points from D, and
- C and D.

Both of these yield a sum of pairwise distances of $k^2/4$, for a ratio of $1/(4x) = (\sqrt{2} + 1)/2 \approx 1.207$.

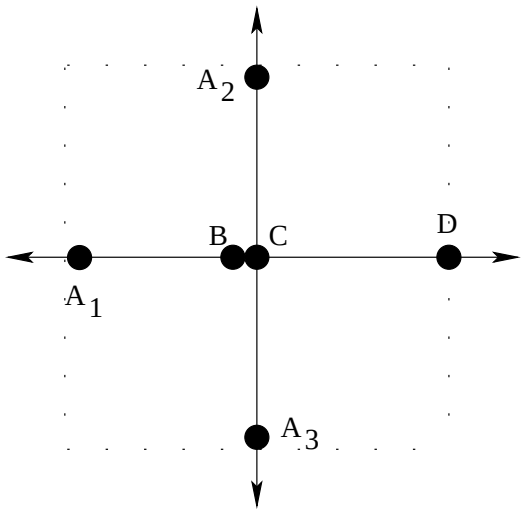

Figure 9: Two-dimensional lower bound for algorithm that also considers the midpoints between grid points

In 2D, the instance, shown in Figure 9,

A_1 is $k(1/8 - x/4)$ points at $(-1 + \epsilon, 0)$,
A_2 is $k(1/8 - x/4)$ points at $(0, 1 - \epsilon)$,
A_3 is $k(1/8 - x/4)$ points at $(0, -1 + \epsilon)$,
B is kx points at $(-\epsilon, 0)$,
C is $k/2$ points at $(0, 0)$, and
D is $k/2$ points at $(1, 0)$,

where $x = \sqrt{3} - 1.5$, gives a ratio $\rho = (3\sqrt{3} + 5)/8 \approx 1.275$.

Generalizing to higher dimensions, $x = (\sqrt{2d(2d-1)} - 2d + 1)/2$ and $\rho \to 4/3$.

Incremental improvement The algorithm runs in 2 steps for each possible center p. First, find the k open processors nearest to p, as the current algorithm does. Second, consider other nearby processors seeing of the star to the selected processors is smaller than the star of one of the selected processors. If so, add the processor with smaller star to the solution and remove the processor with larger star.

Although it seems like a good idea, this incremental improvement scheme does not seem to improve the approximation ratio. Tweaking the example in Subsection 5.2.2 so that the B and C groups have 1 more processor and the A_1 groups have 3 less makes the algorithm select a local minimum solution so no changes will be made. Perhaps considering moving a fraction of k processors could be made to work, but it's unclear how to do this efficiently.

5.3 PTAS for minimal weight subset, L_1 metric

The basic idea is identical to the one used by Fekete and Meijer in an earlier paper [40]. We find (by enumeration) a subdivision of an optimal solution into $m \times m$ rectangular cells C_{ij}, each of which must contain a specific number k_{ij} of selected points. From each cell C_{ij}, the points are selected in a way that guarantees that the total distance to all other cells except for the $m-1$ cells in the same "horizontal" strip or the $m-1$ cells in the same "vertical" strip is minimized. As it turns out, this can be done in a way that the total neglected distance within the strips is bounded by a small fraction of the weight of an optimal solution, yielding the desired approximation property. See Figure 10 for the setup.

For ease of presentation we assume that k is a multiple of m and $m > 2$. Approximation algorithms for other values of k can be constructed in a similar fashion. Consider an optimal solution of k points, denoted by OPT. Furthermore consider a division of the plane by a set of $m+1$ x-coordinates $\xi_0 \leq \ldots \leq \xi_1 \leq \xi_m$. Let $X_i := \{p = (x, y) \mid \xi_i \leq x \leq \xi_{i+1}, 0 \leq i < m\}$ be the vertical strip between coordinates ξ_i and ξ_{i+1}. By enumeration of possible choices of $\xi_0, \ldots, \xi_m$ we may assume that the ξ_i have the property that, for an optimal solution, from each of the m strips X_i precisely k/m points of P are chosen. (A small perturbation does not change optimality or approximation properties of solutions. This shows that in case of several points sharing the same coordinates, ties may be broken arbitrarily; in that case, points on the boundary between two strips may be considered belonging to one or the other of those strips, whatever is convenient to reach the appropriate number of points in a strip.)

Let $w(S, T)$ be the sum of all the distances between points in S and points in T. Let $w_x(S, T)$ and $w_y(S, T)$ be the sum of all the x- and y- distances between points in S and points in T respectively. So $w(S, T) = w_x(S, T) + w_y(S, T)$. Let $w(S) = w(S, S)$, $w_x(S) = w_x(S, S)$ and $w_y(S) = w_y(S, S)$.

Let $S = \{s_0, s_1, \ldots, s_{k-1}\}$ be a minimal weight subset of P, where k is an integer greater than 1. We will label the x- and y-coordinates of a point $s \in S$ by some

(x_a, y_b) with $0 \leq a < k$ and $0 \leq b < k$ such that $x_0 \leq x_1 \leq \ldots \leq x_{k-1}$ and $y_0 \leq y_1 \leq \ldots \leq y_{k-1}$. (Note that in general, $a \neq b$ for a point $s = (x_a, y_b)$.) We can derive the following equation:

$$w_x(S) = (k-1)(x_{k-1} - x_0) + (k-3)(x_{k-2} - x_1) + \ \ldots$$
$$w_y(S) = (k-1)(y_{k-1} - y_0) + (k-3)(y_{k-2} - y_1) + \ \ldots$$

We show that there is a polynomial time approximation scheme (PTAS), i.e., for any fixed positive ε, there is a polynomial approximation algorithm that finds a solution that is within $(1 + \varepsilon)$ of the optimum.

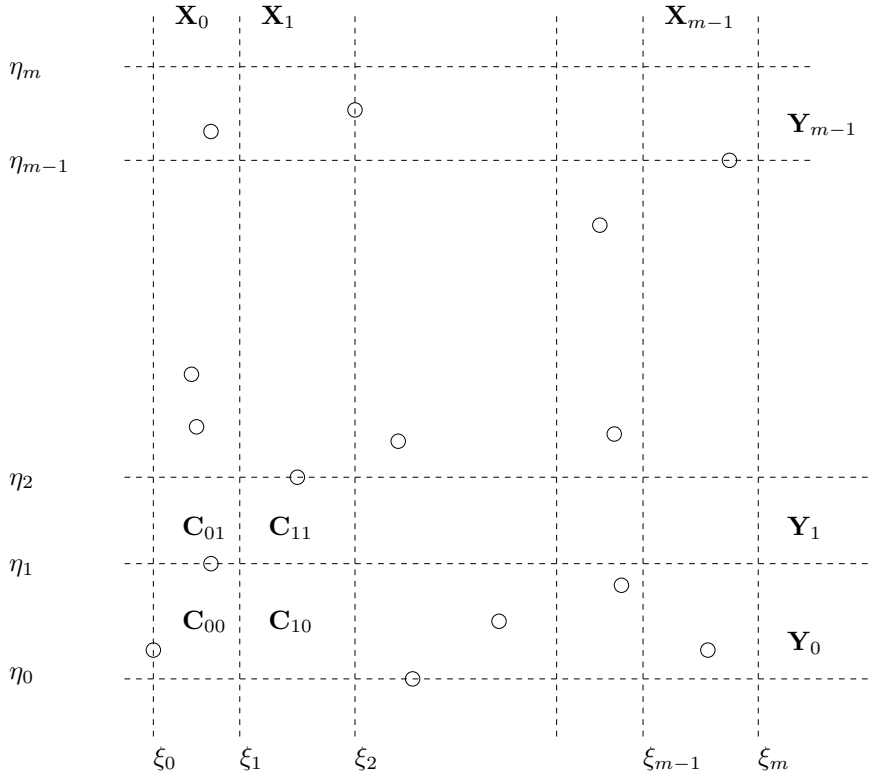

Figure 10: Dividing the point set in horizontal and vertical strips.

In a similar manner, suppose we know $m + 1$ y-coordinates $\eta_0 \leq \eta_1 \leq \ldots \leq \eta_m$ such that from each horizontal strip $Y_i := \{p = (x, y) \mid \eta_i \leq y \leq \eta_{i+1}, 0 \leq i < m\}$ a subset of k/m points are chosen for an optimal solution.

Let $C_{ij} := X_i \cap Y_j$, and let k_{ij} be the number of points in OPT that are chosen from C_{ij}. Since

$$\sum_{0 \leq i < m} k_{ij} = \sum_{0 \leq j < m} k_{ij} = k/m,$$

we may assume by enumeration over the $O(k^m)$ possible partitions of k/m into m pieces that we know all the numbers k_{ij}.

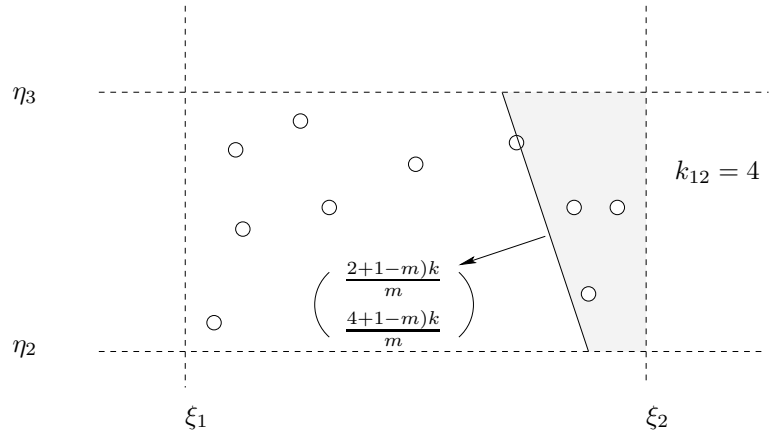

Figure 11: Select points in cell C_{12}.

Finally, define the vector $\nabla_{ij} := ((2i + 1 - m)k/m, (2j + 1 - m)k/m)$. Now our approximation algorithm is as follows: from each cell C_{ij}, choose some k_{ij} points that are minimal in direction ∇_{ij}, i.e., select points $p = (x, y)$ for which $(x(2i + 1 - m)k/m, y(2j + 1 - m)k/m)$ is minimal. For an illustration, see Figure 11.

It can be shown that if we select point in this way from each cell, we minimize the sum of the x-distances from each point in C_{ij} to points not in X_i and the y-distances to points not in Y_j.

(Overlap between the selections from different cells is avoided by proceeding in lexicographic order of cells, and choosing the k_{ij} points among the candidates that are still unselected.) Let HEU be the point set selected in this way.

It is clear that HEU can be computed in polynomial time. We will proceed by a series of lemmas to determine how well $w(HEU)$ approximates $w(OPT)$. In the following, we consider the distances involving points from a particular cell C_{ij}. Let HEU_{ij} be the set of k_{ij} points that are selected from C_{ij} by the heuristic, and let OPT_{ij} be a set of k_{ij} points of an optimal solution that are attributed to C_{ij}. Let $HEU_{i\bullet}$, $OPT_{i\bullet}$, $HEU_{\bullet j}$ and $OPT_{\bullet j}$ be the set of k/m points selected from X_i and Y_j by the heuristic and an optimal algorithm respectively. Finally $\overline{HEU}_{i\bullet} := HEU \setminus HEU_{i\bullet}$, $\overline{HEU}_{\bullet j} := HEU \setminus HEU_{\bullet j}$, $\overline{OPT}_{i\bullet} := OPT \setminus OPT_{i\bullet}$ and $\overline{OPT}_{\bullet j} := OPT \setminus OPT_{\bullet j}$.

5.3.1 Outline

This whole proof is a bit heavy on notation. Maybe this little subsection will help. Notice that

$$\begin{aligned} w(HEU) &= \sum_{i,j} [w_x(HEU_{ij}, \overline{HEU}_{i\bullet}) + w_y(HEU_{ij}, \overline{HEU}_{\bullet j})] + \sum_i w_x(HEU_{i\bullet}) + \\ &\quad \sum_j w_y(HEU_{\bullet j}). \end{aligned}$$

We first show that the first part is smaller that $w(OPT)$. We then show that the second and third part are small fractions of $w(HEU)$.

5.3.2 Details

Lemma 5

$$w_x(HEU_{ij}, \overline{HEU}_{i\bullet}) + w_y(HEU_{ij}, \overline{HEU}_{\bullet j}) \leq w_x(OPT_{ij}, \overline{OPT}_{i\bullet}) + w_y(OPT_{ij}, \overline{OPT}_{\bullet j}).$$

Proof: Consider a point $p \in OPT_{ij} \setminus HEU_{ij}$. Thus, there is a point $p' \in HEU_{ij} \setminus OPT_{ij}$ that was chosen by the heuristic instead of p. Let $p - p' = h = (h_x, h_y)$. When replacing p' in HEU by p, we increase the x-distance to the ik/m points left of C_{ij} by h_x, while decreasing the x-distance to $(m - i - 1)k/m$ points right of C_{ij} by h_x. In the balance, this yields a change of $((2i + 1 - m)k/m)h_x$. Similarly, we get a change of $((2j + 1 - m)k/m)h_y$ for the y-coordinates. Since p' was chosen to minimize the inner product $\langle p', \nabla_{ij} \rangle$ we know that the inner product $\langle h, \nabla_{ij} \rangle \geq 0$, so the overall change of distances is positive.

Performing these replacements for all points in $HEU \setminus OPT$, we can transform HEU to OPT, while increasing the sum of distances $w_x(HEU_{ij}, \overline{HEU}_{i\bullet}) + w_y(HEU_{ij}, \overline{HEU}_{\bullet j})$ to the sum

$$w_x(OPT_{ij}, \overline{OPT}_{i\bullet}) + w_y(OPT_{ij}, \overline{OPT}_{\bullet j}).$$

∎

Corollary 6

$$\sum_{i,j} w_x(HEU_{ij}, \overline{HEU}_{i\bullet}) + w_y(HEU_{ij}, \overline{HEU}_{\bullet j}) \leq w(OPT).$$

In the following two lemmas we show that

$$\sum_i w_x(HEU_{i\bullet})$$

is a small fraction of $w(HEU)$. Similar proofs can be given for

$$\sum_j w_y(HEU_{\bullet j}).$$

Lemma 7

$$\sum_{0<i<m-1} w_x(HEU_{i\bullet}) \;\leq\; \frac{w_x(HEU)}{2(m-2)}.$$

Proof: Let $\delta_i = \xi_{i+1} - \xi_i$. Since $i(m-i-1) \geq m-2$ for $0 < i < m-1$, we have for $0 < i < m-1$

$$w_x(HEU_{i\bullet}) \leq \frac{k^2}{2m^2}\delta_i \leq \frac{ik}{m}\frac{(m-i-1)k}{m}\delta_i \ \frac{1}{2(m-2)}.$$

Since HEU has ik/m and $(m-i-1)k/m$ points to the left of ξ_i and right of ξ_{i+1} respectively, we have

$$w_x(HEU) \geq \sum_{0<i<m-1} \frac{ik}{m}\frac{(m-i-1)k}{m}\delta_i$$

so

$$\sum_{0<i<m-1} w_x(HEU_{i\bullet}) \leq \frac{1}{2(m-2)} w_x(HEU).$$

■

Lemma 8 *For $i = 0$ and $i = m-1$ we have*

$$w_x(HEU_{i\bullet}) \ \leq \ \frac{w_x(HEU)}{m-1}.$$

Proof: Without loss of generality assume $i = 0$. Let $x_0, x_1, \cdots, x_{(k/m)-1}$ be the x-coordinates of the points $p_0, p_1, \ldots, p_{(k/m)-1}$ in $HEU_{0\bullet}$. So

$$\begin{aligned}
w_x(HEU_{0\bullet}) &= \left(\frac{k}{m}-1\right)\left(x_{\frac{k}{m}-1} - x_0\right) + \left(\frac{k}{m}-3\right)\left(x_{\frac{k}{m}-2} - x_1\right) + \ \ldots \\
&\leq \left(\frac{k}{m}-1\right)(\xi_1 - x_0) + \left(\frac{k}{m}-3\right)(\xi_1 - x_1) + \ \ldots \\
&\leq \frac{k}{m}(\xi_1 - x_0) + \frac{k}{m}(\xi_1 - x_1) + \ \ldots \\
&\leq \frac{k}{m}(\xi_1 - x_0) + \frac{k}{m}(\xi_1 - x_1) + \ \ldots \ + \frac{k}{m}\left(\xi_1 - x_{\frac{k}{m}-1}\right).
\end{aligned}$$

Since $\xi_1 - x_j \leq x - x_j$ where $0 \leq j < k/m$ and x is the x-coordinate of any point in $\overline{HEU}_{0\bullet}$ and since there are $(m-1)k/m$ points in $\overline{HEU}_{0\bullet}$, we have

$$\xi_1 - x_j < \frac{m}{(m-1)k} w_x(p_j, \overline{HEU}_{0\bullet})$$

so

$$w_x(HEU_{0\bullet}) \ \leq \ \frac{k}{m}\frac{m}{(m-1)k} \sum_{0\leq i<\frac{k}{m}} w_x(p_i, \overline{HEU}_{0\bullet})$$

$$\begin{aligned} &\leq \frac{1}{m-1} \sum_{0 \leq i < \frac{k}{m}} w_x(p_i, \overline{HEU}_{0\bullet}) \\ &= \frac{1}{m-1} w_x(HEU_{0\bullet}, \overline{HEU}_{0\bullet}) \\ &\leq \frac{1}{m-1} w_x(HEU). \end{aligned}$$

■

5.3.3 Result

We can combine these three lemmas to get the result we need.

$$\begin{aligned} w(HEU) &= \sum_{i,j} w_x(HEU_{ij}, \overline{HEU}_{i\bullet}) + w_y(HEU_{ij}, \overline{HEU}_{\bullet j}) + \sum_i w_x(HEU_{i\bullet}) + \\ &\quad \sum_j w_y(HEU_{\bullet j}) \\ &\leq w(OPT) + \frac{1}{2(m-2)}(w_x(HEU) + w_y(HEU)) + \frac{2}{m-1}(w_x(HEU) + \\ &\quad w_y(HEU)) \\ &= w(OPT) + \frac{1}{2(m-2)} w(HEU) + \frac{2}{m-1} w(HEU). \end{aligned}$$

So

$$w(HEU)(1 - \frac{1}{2(m-2)} - \frac{2}{m-1}) \leq w(OPT).$$

5.4 Alternate PTAS

Now we present another PTAS based on different ideas. Intuitively, this second PTAS tries to guess the median point of the optimal solution and the optimal solution's "shape". To limit the search space, we show that "far away" points can only be included in the optimal solution if all "nearby" points are also part of the optimal solution.

Assume no two points are on the same row or column by breaking ties. Guess a median (n^2 possible), up to four extremal points (at most n^4 possible 4-tuples), and four $\epsilon_1 k$ boundary lines or *core lines*, which are lines with $\epsilon_1 k$ points of the solution to the left, to the right, above, or below (at most n^3 possible 4-tuples); see Figure 12.

Based on the extremal points, we know the *bounding box* for the solution. No points outside of the bounding box can be in the solution. If the bounding box has fewer than k points, then the extremal points are invalid and we guess another set

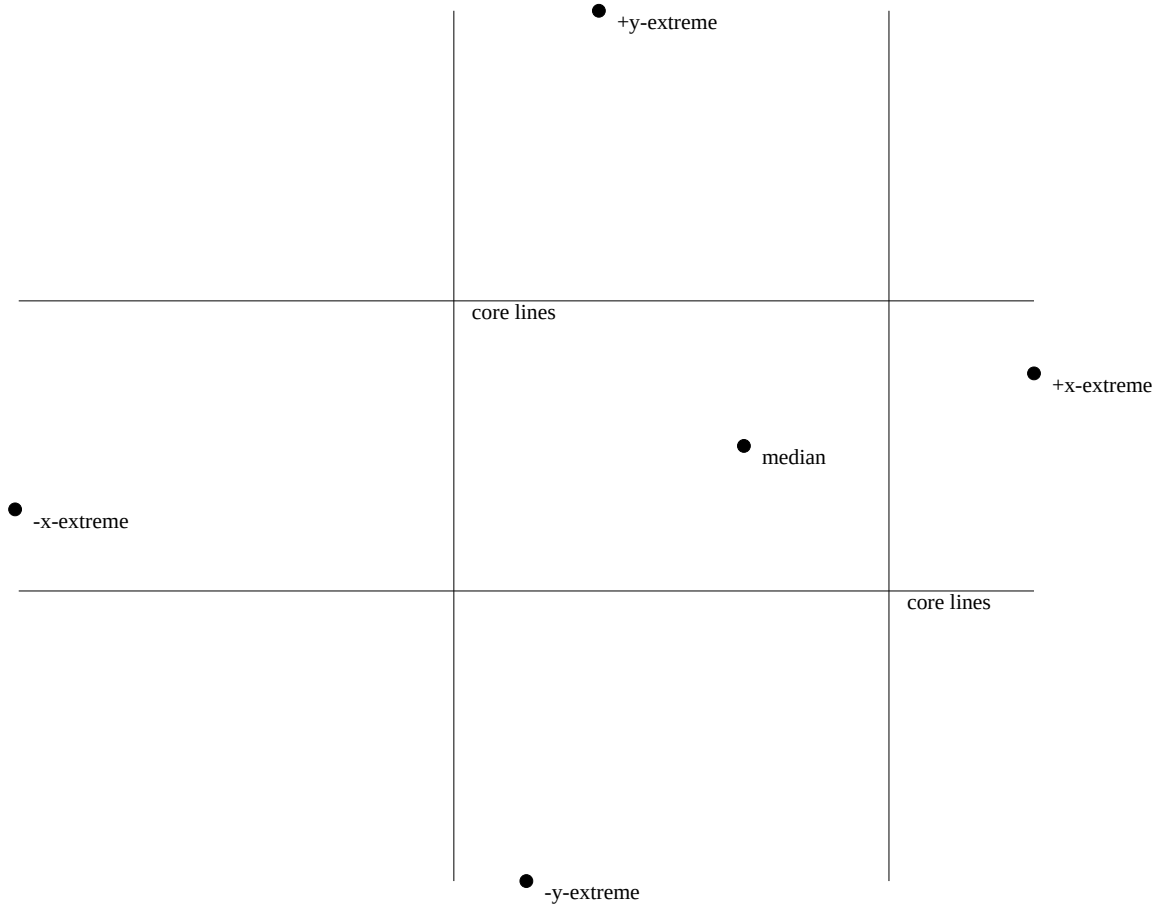

Figure 12: Guesses for median, up to four extremal points, and four core lines

of extremal points. We next compute the *forced region* of the solution. This region is defined by the extremal points, the median, and the four $\epsilon_1 k$ boundary lines; note that the forced region may extend beyond the bounding box. All points within the forced region must belong to the solution. If the forced region contains more than k points or contains any points outside of the bounding box, then the above guesses are invalid and we guess again.

The region between the bounding box and the forced region is called the *uncertain region*. The uncertain region contains points that may or may not be in the solution. Let the forced region contain some number $k - x$ of points for $0 \leq x \leq k - 2$, which we know are in the solution. We have to choose the remaining x points from the uncertain region.

In order to choose these remaining x points, we divide the uncertain region into $1/\text{poly}(\epsilon)$ cells, such that the cell structure has the following crucial property: Con-

sider the family of solutions that selects n_i points from cell i; thus,

$$\sum_i n_i = x.$$

All solutions in this family have cost within a factor of $1 + \epsilon$ of each other.

The PTAS is as follows:

PTAS: Iterate over all possible choices of median, extremal points, and $\epsilon_1 k$ boundary lines

1. Compute the bounding box and the forced region, and determine the number of points in each region.
2. If there are too few points in the bounding box or too many points in the forced region, then begin next iteration.
3. Otherwise, let $k - x$ be the number of points in the forced region, where $0 \leq x \leq k - 2$, meaning that we choose x points from the uncertain region.
4. Divide the uncertain region into cells $1, 2, 3, \ldots, C = 1/\text{poly}(\epsilon)$. Iterate over all families of solutions, i.e., choices of $n_1, n_2, n_3, \ldots, n_C$, such that

$$\sum_i n_i = x.$$

5. For each $n_1, n_2, n_3, \ldots, n_C$, calculate the cost of a sample solution.

Return the best solution.

We show that this algorithm has the following performance:

Theorem 7 *The running time for the PTAS to compute a solution of cost $O((1 + \epsilon)\text{OPT})$ is $O(n^9 k^C)$, where $C = 1/poly(\epsilon)$.*

Recall that the additive contribution to the objective function from a given point is the star centered at that point. Therefore, to prove the $(1 + \epsilon)$ bound on our algorithm, it suffices to create cells such that the following property holds for all cells.

Definition 9 (cell property) *Consider any two points p and p' in a cell i, and a candidate set of $k - 1$ other points. Let S_p and $S_{p'}$ be the length of the stars centered at p and p', respectively, where $S_p < S_{p'}$. Then $S_{p'} - S_p < \epsilon S_p$.*

5.4.1 Dividing the Uncertain Region into Cells

We now show how to divide the uncertain region into cells and prove that these cells obey the cell property. We divide the uncertain region into two parts based on proximity to the median. In each region we use a distinct method of partitioning the region and proving the cell property. The first region we call the *core*, a rectangle defined by the four $\epsilon_1 k$ boundary lines. The second region we call the *mantle*, the remainder of the bounding box.

We will use the following structural property of a two-dimensional set of points.

Observation 10 *Consider a set of k points in $\Re^2$ and place the coordinate system so that the median is at the origin. Let k_i be the number of points in quadrant i, $i = 1, \ldots, 4$. Then, $k_1 = k_3$, and $k_2 = k_4$. For any pair of points lying in diagonally opposite quadrants (quadrants 1 and 3, or 2 and 4), there exists a shortest L_1 path between these two points that passes through the median/origin.*

The first observation follows from the definition of the median: There are a total of $k/2$ points in every two adjacent quadrants.

Dividing the Core into Cells We now prove that the dimensions of the core are small relative to average inter-point distance $\textsc{optave} = \textsc{opt}/k^2$.

Lemma 11 *At most $\eta \leq 2k\sqrt{\textsc{optave}/\delta}$ points are at least distance δ from the median.*

Proof: Among the η most distant points, let η_i denote the number of points in quadrant i. Observation 10 guarantees that for each distant point there is a matching point in the diagonally opposite quadrant, where the shortest paths travel through the median. Note that this matching point may or may not be a distant point, but it does not matter to the analysis. Therefore a lower bound on $\textsc{opt}$ is

$$\sum_{i=1}^{4} \eta_i^2 \delta \leq \textsc{opt} = k^2 \textsc{optave}.$$

This sum only accounts for the distances between the distant points to the median. Because $\sum_{i=1}^{4} \eta_i^2$ is minimized when $\eta_1 = \eta_2 = \eta_3 = \eta_4 = \eta/4$,

$$\eta^2 \delta / 4 \leq k^2 \textsc{optave}.$$

Therefore,

$$\eta \leq 2k\sqrt{\textsc{optave}/\delta}.$$

∎

Corollary 12 *There are at most $k\epsilon_1$ points at distance $4\textsc{optave}/\epsilon_1^2$ from the median.*

Proof: Let $\delta = 4\text{OPTAVE}/\epsilon_1^2$ and the corollary follows from Lemma 11. ∎

Corollary 13 *The core has dimension at most* $8\text{OPTAVE}/\epsilon_1^2 \times 8\text{OPTAVE}/\epsilon_1^2$.

Proof: There are exactly $k\epsilon_1$ points to the right, left, above, and below the core. Therefore, by Corollary 12, the distance from the median to each of these lines can be most $4\text{OPTAVE}/\epsilon_1^2$. ∎

We divide the core into equal-sized cells that are rectangles having dimension at most $\epsilon\text{OPTAVE}/2 \times \epsilon\text{OPTAVE}/2$. Thus, we divide the core into a grid of $16\epsilon^{-1}\epsilon_1^{-2} \times 16\epsilon^{-1}\epsilon_1^{-2}$ cells for a total of $256\epsilon^{-2}\epsilon_1^{-4}$ equal-sized cells.

Lemma 14 *The cells in the core obey the cell property.*

Proof: The maximum distance between any two points in a cell in the core is ϵOPTAVE. Therefore the difference between the stars of any two points in the same cell is at most $(k-1)\epsilon\text{OPTAVE}$. ∎

Dividing the Uncertain Region in the Mantle into Cells Let D be the maximum vertical or horizontal distance from the median to any of the extremal points, and let R be the maximum distance from the median to a $k\epsilon_1$ line. If $D = O(R)$ then we divide the entire bounding box into cells in the same manner that we divide the core, and by similar reasoning all cells obey the cell property. The difficult case is when $D = \Omega(R)$, and we address this case in the rest of this subsection.

We divide the mantle into $O(D^2/\text{poly}(\epsilon))$ equal-size cells with dimensions at most $c\epsilon D \times c\epsilon D$, where constant $0 < c < 1$ is determined below. Unlike in the core, not all of these cells obey the cell property. However, we prove that any cell that does not obey the cell property lies within the forced region, and therefore all points in this cell must belong to the k points in the candidate solution.

We next prove that any cell that is sufficiently far away from the median has the cell property.

Lemma 15 *For any* $0 < \alpha < 1$ *there exists a* c *such that the following holds: Any cell at distance at least* αD *from the median having size at most* $c\epsilon D \times c\epsilon D$ *obeys the cell property.*

Proof: Because the cell is at distance at least αD from the median, the cell is at distance at least $\alpha D/2$ from at least $k/2$ of the points. Therefore the cost of the star of a point in the cell is at least $k\alpha D/4$. In order to guarantee the cell property, we want the difference between the stars of any two points in the cell to be at most $\epsilon k\alpha D/4$. Therefore we set $c = \alpha/8$ so that the cell has dimension $\epsilon\alpha D/8 \times \epsilon\alpha D/8$. Therefore, any two points in the cell are at distance at most $\epsilon\alpha D/4$ from each other, and therefore the difference between their two stars is that most $\epsilon\alpha D(k-1)/4$. ∎

We now exhibit an α such that all points that are at distance at most αD from the median are in the forced region. To do so we describe the structure of the forced region in more detail.

Lemma 16 *Consider any extremal point p. Any point q that lies on a shortest path from p to the median must be in the candidate solution.*

Proof: Assume for the sake of contradiction that there exists a point q that lies on a shortest path from p to the median, where q is not in the candidate solution. Consider moving a point along the shortest path from p to q of length $\text{dist}(p, q)$. As it makes each step along this path, the point approaches at least $k/2+1$ other points (the points on the other side of the median and q) and moves away from at most $k/2-1$ points. Therefore the cost of p's star is greater than the cost of q's star by at least $2\text{dist}(p, q)$. ∎

A consequence of Lemma 16 is that any optimal solution must be convex. Unfortunately, this requirement is insufficient to guarantee that points near the core are in the forced region.

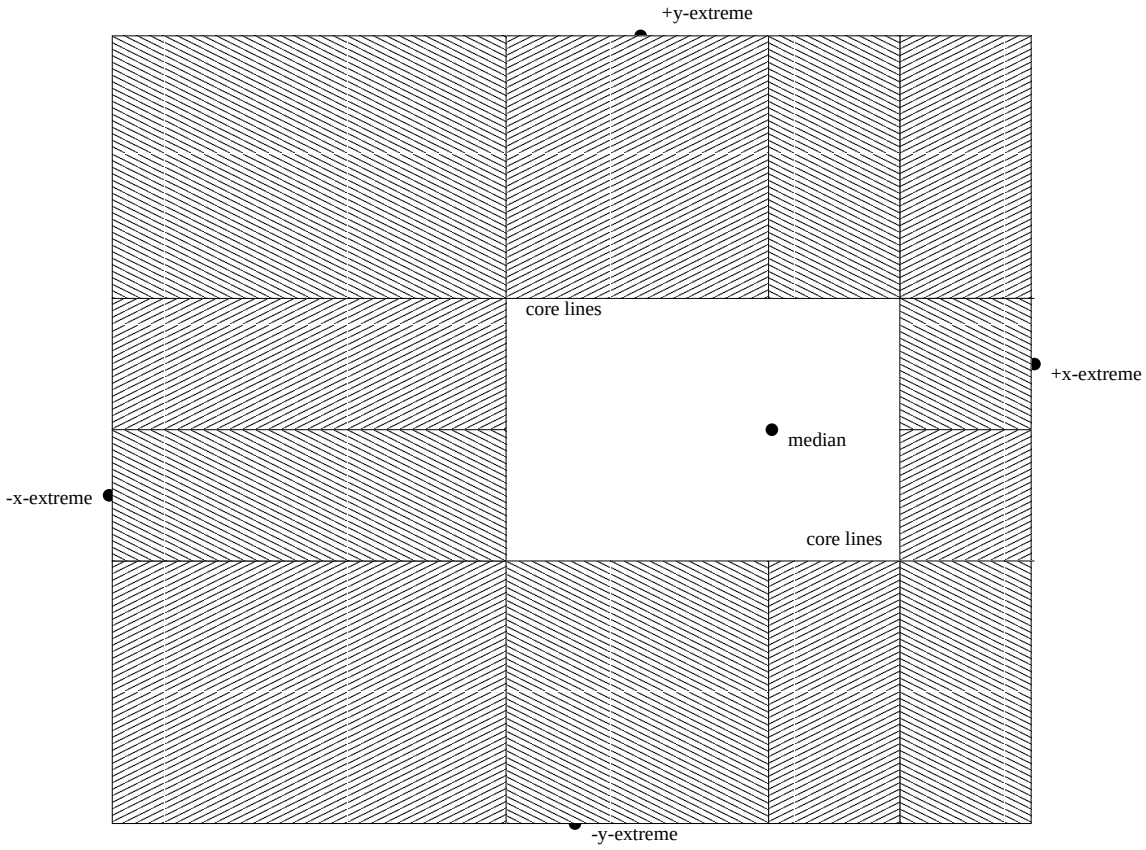

Figure 13: The twelve regions of the mantle

Instead, we use a more precise description of the forced region. Let p be a point in the forced region of the mantle. The four core lines and the quadrants defined by the median partition the mantle into twelve regions; see Figure 13. While p can lie in any of these regions, by symmetry we only have to consider two regions:

- *Region A* — the region to the *left* of both vertical core lines, *above* the median, and *below* the top core line, and
- *Region B* — the region to the *left* of both vertical core lines and *above* both horizontal core lines.

We use the following operational definition of the forced region: A point p is in the forced region if we can prove (from the location of the core and median) that the star at point p is smaller than the star of (at least) one of the extremal points. The proof that p is in the forced region will be by contradiction. Suppose that the optimal solution does not contain open point p in the forced region. We could remove the extremal point from the candidate solution and replace it with p, thus obtaining a better solution.

Lemma 17 *Let point p in region A be in the forced region. Consider the line L_1 with slope 1 and the vertical line L_2 that both pass through p. The subregion of A to the right of both L_1 and L_2 forms part of the forced region.*

Proof: Call the subregion of A to the right of both L_1 and L_2 A_{force}. Any point in A_{force} *below* p (and thus to the right of L_2) is on a shortest path from p to the median. Therefore from Lemma 16 the subregion of A_{force} below p forms part of the forced region.

Consider any point q in A_{force} that is *above* p (and thus to the right of L_1). Suppose that a shortest path from p to q travels a distance x to the right and a distance y up. Because q is to the right of L_1, $x \geq y$.

Consider moving a point from p to q along this shortest path. When this point moves to the right a distance x, it approaches at least $(1 - \epsilon_1)k$ points and moves away from at most $\epsilon_1 k$ points because p and q are both to the left of both vertical core lines. Therefore the cost of the point's star decreases by at least $x(1-2\epsilon_1)$. Next the point moves up a distance y. The point approaches at least $\epsilon_1 k$ points and moves away from at most $(1 - \epsilon_1)k$ points because p and q are between the horizontal core lines. Therefore the cost of the point's star increases by at most $y(1-2\epsilon_1)k$. Because $y \leq x$, the cost of the star decreases; because point p is in the forced region, so is point q. ∎

Lemma 18 *Let point p in region B be in the forced region. Consider the line L_1 with slope $1-2\epsilon_1$ and the vertical line L_2, where L_1 and L_2 pass through p. The subregion of B to the right of both L_1 and L_2 forms part of the forced region.*

Proof: Call the subregion of B to the right of both L_1 and L_2 B_{force}. Any point in B_{force} *below* p (and thus to the right of L_2) is on is a shortest path from p to the median. Therefore from Lemma 16 the subregion of B_{force} below p forms part of the forced region.

Consider any point q in B_{force} that is *above* p (and thus to the right of L_1). Suppose that a shortest path from p to q travels a distance x to the right and a distance y up. Because q is to the right of L_1, $y \leq (1-2\epsilon_1)x$.

Consider moving a point from p to q along this shortest path. When this point moves to the right a distance x, it approaches at least $(1-\epsilon_1)k$ points and moves away from at most $\epsilon_1 k$ points because p and q are both to the left of the vertical core lines. Therefore the cost of the point's star decreases by at least $x(1-2\epsilon_1)$. Next the point moves up a distance y. The point may move away from all k points because p and q are above both horizontal core lines. Therefore the cost of the point's star increases by at most ky. Overall, the change in the cost of the star is $ky - x(1-2\epsilon_1) < 0$ so the cost of the star decreases. Because point p is in the forced region, so is point q. ∎

Lemmas 17 and 18 explain why the forced region may extend beyond the bounding box. In this case the forced region outside of the bounding box must be empty for us to have a valid candidate solution.

We now compute the parameter α from earlier in the section.

Lemma 19 *Let R be the radius of the core. Then any point at distance $D-4D\epsilon_1-2R$ is in the forced region. Therefore, $\alpha = 1-4\epsilon_1-2R/D$.*

Proof: We calculate a rough approximation of the bounding box to show that any cells that are in the uncertain region are far from the median. We start from an extremal point p that has the farthest x or y distance from the median. Without loss of generality, assume that p lies on the left vertical boundary of the bounding box and that the x-distance between p and the median is D.

Let point p_1 lie at the intersection point of the bounding box in the negative x-axis. From Lemma 17 we know that p_1 is in the forced region. Now consider line L_1 that passes through point p_1 and has slope $1-2\epsilon$. Follow L_1 until it intersects with the leftmost (vertical) core line, and call the intersection point p_2. Then consider horizontal line L_2 that passes through p_2, and follow L_2 until it intersects with the positive y-axis and call the intersection point p_3. Consider line L_3 with slope $-1/(1-2\epsilon)$ that passes through p_3, and follow L_3 until it intersects with the topmost (horizontal) core line. Call the intersection point p_4, and follow L_4 down until it intersects with the positive x-axis. Call this intersection point p_5. All points defined by the upper envelope of lines $L_1, \ldots, L_4$ connecting points $p_1, \ldots, p_5$ and the lower envelope of the x-axis connecting points p_1 to p_5 belong to the forced region. The same analysis is used to estimate the forced region below the x-axis.

We now compute a lower bound on the minimum distance from the median to a boundary of the forced region. As we travel from p_1 to p_5 along $L_1, \ldots, L_4$ we get increasingly closer to the median. Using simple algebra, since p_1 is that distance exactly D from the median, p_2 is a distance at least $(1-2\epsilon)D$ from the median, p_3 is a distance at least $(1-2\epsilon)D-R$ from the median, p_4 is a distance at least $(1-4\epsilon)D-R$ from the median, and p_5 is a distance at least $(1-4\epsilon)D-2R$ from the median; see Figure 14. ∎

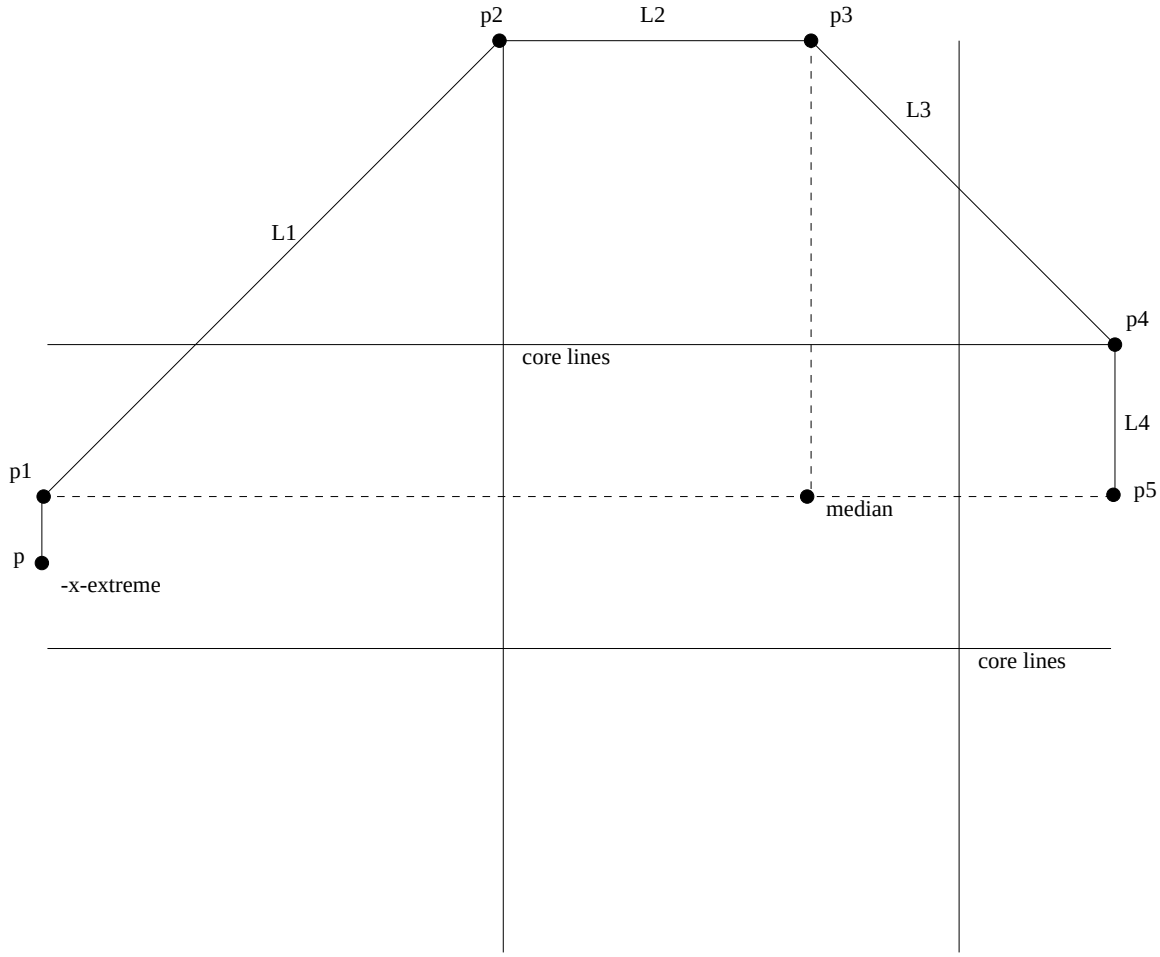

Figure 14: Lines $L_1, \ldots, L_4$ and points $p_1, \ldots, p_5$

5.5 Conclusion About Minimizing Sum of Pairwise Distances

For minimizing sum of pairwise distances, we have exact algorithms for special cases, several constant approximations, and two PTASes. We do not really know which PTAS would be more useful in practice, but both would involve substantial coding and potentially involve spending too much time on allocation decisions. Because the 2-approximation from Subsection 5.2 balances solution quality and simplicity, we believe this is the correct processor-allocation algorithm to implement in supercomputers. Indeed, it is slated for inclusion in Cplant System Software, Version 2.1, to be released.

Intentionally Left Blank

6 Continuous version of problem

In this section, we solve the continuum version of the allocation problem in Section 5. We treat the grid of processors as having a continuous two-dimensional planar distribution. For simplicity we call such a distribution a *city*. Our objective is to find the shape of a city that minimizes the average pairwise Manhattan distance between all points in that city. Fekete et al. [41, 42] consider the city-center problem under Manhattan distances: For a given city, find a point that minimizes the average distance. The techniques that we use to solve this problem are those that are commonly used in the solution of continuum mechanics and classical and quantum field theory problems; namely, variational methods [76, 78]. These results are also reported in [10].

6.1 Results

In this section we present the following results:

- We optimize a one-parameter family of boundary curves for a city. Specifically, we consider cities in the shape of an L_p ball around the origin, and we find the value of p that minimizes the average unitless Manhattan distance. It is easy to show that the diamond (L_1 ball) is a better shape for a city than a square (L_∞ ball). Furthermore, a circle (L_2 ball) is a better shape for a city that a diamond; apparently, the optimal city shape is rounded, even though the Manhattan distance is composed of vertical and horizontal measurements. In fact, the circle is *not* the optimal city shape. Using both analytical and numerical methods, we show that the average unitless Manhattan distance for this one-parameter family of boundary curves is obtained when $d \approx 1.81546$; see Table 6.4.4 and Figure 16. We also explore how rapidly the average unitless distance between points increases when $p < 1$, the domain in which the city is nonconvex. The contrast between the Manhattan distance and the Euclidean distance is noteworthy; for Euclidean distance it is easy to see that the L_2 ball minimizes the average L_2 distance.

- For the main result of the section, we determine the shape of the optimal city's boundary curve. We use variational methods [76, 78] to calculate a nonlinear differential equation satisfied by the boundary curve; see Figures 17 and 18. Then we give a numerical procedure for solving differential equation. The true minimum average unitless distance, as obtained in Section 6.5, is 0.650 245 952 951. It is remarkable that the one-parameter family of curves gives an extremely close approximation that differs by only $2.6 \times 10^{-4}\%$ from the minimum unitless distance.

6.2 Overview

In Subsection 6.3 we formulate this optimization problem in mathematical terms. Then, in Subsection 6.4 we consider a one-parameter family of cities and find the shape of the optimal city in this limited family. From this calculation we can see that the optimal city is not circular. Next, in Subsection 6.5 we carry out a full variational calculation and determine an interesting nonlinear differential equation satisfied by the boundary of the optimal city. The optimal city is nearly circular in shape but is not a disk.

Finally, in Subsection 6.6 we offer some concluding observations and suggest some further avenues for investigation. Our principal conclusions are that so long as the allocation of processors are clustered in a tight shape having no holes or only small holes, the allocation will be extremely close to optimal.

6.3 Formulation of the Problem

Let $w(x)$ be the upper boundary of the region occupied by the city. Without loss of generality we may assume that $w(x)$ is positive. Because the Manhattan distance is north-south symmetric we may assume that the lower bound of the region is $-w(x)$. Also, because the Manhattan distance is east-west symmetric we may assume that $w(x) = w(-x)$. The boundary of the city must be continuous, so we assume that $w(x)$ crosses the x-axis at the point $x = a$: $w(a) = w(-a) = 0$.

Furthermore, because the north-south and east-west directions are of equal weight we expect the shape of the city to be symmetric about the 45° lines $y = x$ and $y = -x$. This means that we may decompose the function $w(x)$ into two functions, $w(x) = g(x)$ below the line $y = x$ and $w(x) = h(x)$ above the line $y = x$:

$$w(x) = \begin{cases} h(x) & (0 \le x \le b), \\ g(x) & (b \le x \le a), \end{cases} \tag{1}$$

where b marks the point on the x-axis where the boundary curve $w(x)$ crosses the line $y = x$. The symmetry about the line $y = x$ implies that $h(x)$ is the *functional* inverse of $g(x)$: $h(x) = g^{-1}(x)$.

Finally, we define the length scale of this problem by choosing, without loss of generality, to work in units such that $b = 1$. We summarize the properties of the functions $w(x)$, $h(x)$, and $g(x)$ as follows:

$$\begin{aligned} w(0) &= h(0) = a, \\ w(1) &= h(1) = g(1) = 1, \\ h(x) &= g^{-1}(x) \quad (0 \le x \le 1), \\ w(x) &\ge 0 \quad (-a \le x \le a). \end{aligned} \tag{2}$$

The functions $g(x)$ and $h(x)$ are illustrated in Figure 15. The curve that outlines the city is symmetric with respect to reflections about the x axis $y = 0$, the y axis $x = 0$,

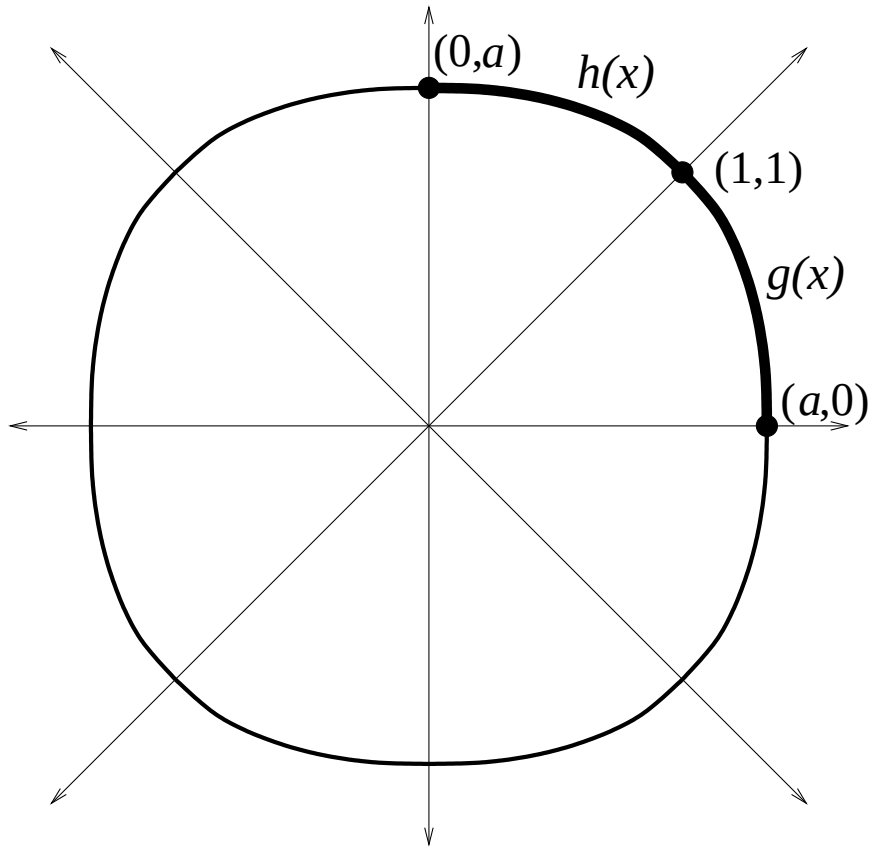

Figure 15: Definition of the notation used in Section 6.

and the 45° lines $y = \pm x$. In the upper-half plane this outline curve is called $w(x)$. Note that $w(x)$ crosses the y and x axes at the points $(0, a)$ and $(\pm a, 0)$. The slope of $w(x)$ is 0 at $x = 0$ and infinite at $x = a$. Also, $w(x)$ crosses the line $y = x$ at the point $(1, 1)$, and at this point the slope is -1. The portion of the function $w(x)$ in the range $0 \le x \le 1$ is called $h(x)$ and the portion of $w(x)$ in the range $1 \le x \le a$ is called $g(x)$. The functions $h(x)$ and $g(x)$ are inverses of one another because $w(x)$ is symmetric about the line $y = x$.

Let the functional $A[w]$ represent the area of the city whose boundary curve is shown in Figure 15. Note that $A[w]$ is four times the area in the positive quadrant:

$$A[w] = 4 \int_{x=0}^{a} dx\, w(x)\,. \tag{3}$$

Let the functional $M[w]$ represent the integrated sum of the distances between all pairs of points in the city. Then,

$$M[w] = 2 \int_{x=-a}^{a} dx \int_{y=-w(x)}^{w(x)} dy \int_{u=-a}^{a} du \int_{v=-w(u)}^{w(u)} dv\, |x - u|\,. \tag{4}$$

The factor of two in this equation arises because we have summed only over east-west distances. North-south distances make an equal contribution.

Our objective is to describe the shape of a region of given area for which the average pairwise distance between points in the region is minimized. That is, our objective is to find the function $w(x)$ that minimizes the functional $M[w]$ subject to the constraint that the area $A[w]$ is held fixed. Note that $M[w]$ has units of $[\text{LENGTH}]^5$ and that $A[w]$ has units of $[\text{LENGTH}]^2$. Thus, a *dimensionless* measure of the average pairwise distance is given by the ratio $D[w]$:

$$D[w] = \frac{M[w]}{(A[w])^{5/2}} \,. \tag{5}$$

To prepare for the calculations to be done in the next two subsections we simplify the functional $M[w]$. First, we perform the y and v integrals in (4):

$$M[w] = 8 \int_{x=-a}^{a} dx \int_{u=-a}^{a} du \, |x-u| w(x) w(u) \,.$$

Then, we decompose the integration region to eliminate the absolute value signs:

$$\begin{aligned} M[w] \;\; = \;\; & 16 \int_{x=0}^{a} dx \int_{u=0}^{a} du \, (x+u) w(x) w(u) + 16 \int_{x=0}^{a} dx \int_{u=0}^{x} du \, (x-u) w(x) w(u) \\ & +16 \int_{x=0}^{a} dx \int_{u=x}^{a} du \, (u-x) w(x) w(u) \,. \end{aligned}$$

Finally, we split apart the terms, change variable names, and merge integrands so that all limits are the same. Our final result is

$$M[w] = 64 \int_{x=0}^{a} dx \, x w(x) \int_{u=0}^{x} du \, w(u) \,. \tag{6}$$

6.4 Special Cases and Optimization of a One-Parameter Family of Boundary Curves

Let us begin by illustrating the calculation of the dimensionless functional $D[w]$ in (5) in terms of $A[w]$ in (3) and $M[w]$ in (6) for some elementary shapes.

6.4.1 Square

Consider a square city defined by the function $w(x) = 1$ $(-1 \le x \le 1)$. For this geometry $a = 1$, $h(x) = 1$ and $g(x)$ is a vertical line connecting the points $(1, 1)$ to $(1, 0)$. The area of this square is $A = 4$. We calculate M in (6) as follows:

$$M = 64 \int_{x=0}^{1} dx \, x w(x) \int_{u=0}^{x} du \, w(u) = 64 \int_{x=0}^{1} dx \, x \int_{u=0}^{x} du = \frac{64}{3} \,. \tag{7}$$

Thus,

$$D = \frac{M}{A^{5/2}} = \frac{2}{3} \approx 0.666\,667 \,. \tag{8}$$

6.4.2 Diamond

Next, consider a diamond-shaped city for which $w(x) = 2 - |x|$ $(-2 \le x \le 2)$. For this case $a = 2$, the area of the city is $A = 8$, and

$$M = 64 \int_{x=0}^{2} dx\, xw(x) \int_{u=0}^{x} du\, w(u) = 64 \int_{x=0}^{2} dx\, x(2-x) \int_{u=0}^{x} du\, (2-u) = \frac{1792}{15}\,. \quad (9)$$

Thus, for a diamond-shaped city

$$D = \frac{M}{A^{5/2}} = \frac{7}{15}\sqrt{2} \approx 0.659\,966\,. \quad (10)$$

6.4.3 Disk

Now let us consider a circular city, whose boundary is given by $w(x) = \sqrt{2 - x^2}$ with $-\sqrt{2} \le x \le \sqrt{2}$. Here, $a = \sqrt{2}$, the area of the city is $A = 2\pi$, and

$$M = 64 \int_{x=0}^{\sqrt{2}} dx\, xw(x) \int_{u=0}^{x} du\, w(u) = 64 \int_{x=0}^{\sqrt{2}} dx\, x(2-x^2) \int_{u=0}^{x} du\, (2-u^2) = \frac{2048}{45}\sqrt{2}\,.$$

Thus, for a circular city

$$D = \frac{M}{A^{5/2}} = \frac{512}{45\pi^{5/2}} \approx 0.650\,403\,. \quad (11)$$

6.4.4 The Family of Curves $w(x) = (2 - |x|^p)^{1/p}$

Evidently, a disk is a better shape for a city than a square or a diamond. Apparently, the optimal city shape is rounded even though the Manhattan distance is composed of vertical and horizontal measurements. On the basis of these calculations one might wonder if a disk is indeed the optimal shape for a city. In fact, a disk is not optimal, as we now show. Consider a one-parameter family of boundary functions of the form $w(x) = (2 - |x|^p)^{1/p}$ $(-2^{1/p} \le x \le 2^{1/p})$. For this family of functions we have $a = 2^{1/p}$. Note that this family includes a square $(p = \infty)$, a diamond $(p = 1)$, and a disk $(p = 2)$ as special cases.

It is noteworthy that the function $D(p)$ is quite insensitive to the value of p: The values of D for three such different city shapes, a square, a diamond, and a disk, differ by only 3%. We now show that a disk is not the optimal shape for a city by studying how the value of D depends on p. The area of the city, as a function of p, is given by

$$A(p) = 4 \int_{x=0}^{a} dx\, (a^p - |x|^p)^{1/p} = \frac{2a^2}{p} \frac{\Gamma^2(1/p)}{\Gamma(2/p)}\,, \quad (12)$$

where the Gamma function is defined by $\Gamma(\alpha) \equiv \int_{x=0}^{\infty} dx\, e^{-x} x^{\alpha-1}$ and we have used the standard integration formula [48, 38]

$$\int_{x=0}^{1} dx\, x^{\alpha-1}(1-x)^{\beta-1} = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}\,. \quad (13)$$

The expression for $M(p)$ cannot be evaluated in closed form and may be left as a double integral:

$$M(p) = 64a^5 \int_{x=0}^{1} dx\, x(1-x^p)^{1/p} \int_{u=0}^{x} du\, (1-u^p)^{1/p}\,. \tag{14}$$

For numerical purposes we convert this expression for $M(p)$ to an infinite sum. We do so by expanding the expression $(1-u^p)^{1/p}$ into a binomial series and using (13) to perform the double integration:

$$M(p) = 16\frac{\Gamma(1+1/p)}{p\Gamma(-1/p)} \sum_{n=0}^{\infty} \frac{\Gamma(n-1/p)\Gamma(n+3/p)}{n!(1+np)\Gamma(n+1+4/p)}\,. \tag{15}$$

This series converges like $n^{-3-2/p}$ for large n and is therefore useful for obtaining numerical results.

Boundary shape	p	$D(p)$
Concave boundaries:		
plus sign	0	∞
	1/64	16, 134.274
	1/32	82.153 831
	1/16	6.122 774
	1/8	1.719 920
	1/4	0.940 692
	1/2	0.723 183
astroid	2/3	0.686 397
Convex boundaries:		
diamond	1	0.659 966
	1.81	0.650 247 797
	1.812	0.650 247 700
	1.814	0.650 247 646
optimal	1.815 46	0.650 247 634
	1.816	0.650 247 636
	1.818	0.650 247 670
	1.82	0.650 247 746
disk	2	0.650 403
square	∞	0.666 667

Table 6: Numerical values for $D(p) = M(p)A^{-5/2}(p)$ for the one-parameter family of boundary curves $w(x) = (a^p - x^p)^{1/p}$ with $a^p = 2$.

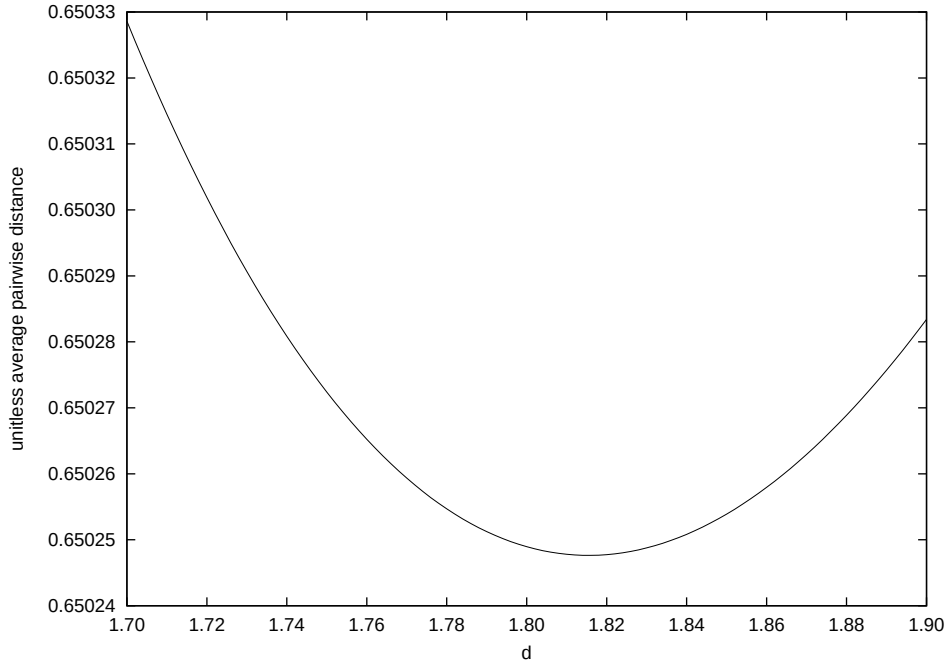

Figure 16: Plot of $D(p) = M(p)A^{-5/2}(p)$ for the class of boundary curves $w(x) = (a^p - x^p)^{1/p}$ with $a^p = 2$ in the range $1.7 \le p \le 1.9$.

We have evaluated $D(p) = M(p)A^{-5/2}(p)$ numerically for many values of p in the range $0 \le p < \infty$. (See Table 6.4.4 and Figure 16.) The function $D(p)$ is infinite at $p = 0$, falls to a minimum just below $p = 2$, and then rises and levels off at $p = \infty$. The optimal city, that is, the city for which $D(p)$ is minimized, is nearly circular, but is definitely not a disk ($p = 2$).

We expect the shape of an optimal city to be convex. The function $w(x) = (a^p - x^p)^{1/p}$ is convex when $p \ge 1$ and concave when $p < 1$. Thus, we expect the optimal value of p for this class of functions to be in the range $p > 1$. Indeed, by interpolating the values in Table 6.4.4 we find that $D(p)$ attains its minimum of 0.650 247 634 near $p = 1.81546$. It is remarkable that this one-parameter family of curves gives an extremely close approximation that differs by only $2.6 \times 10^{-4}\%$ from the true minimum of the functional $D[w]$. The true minimum, as obtained in Subsection 6.5, is 0.650 245 952 951.

Nevertheless, it is interesting to study the function $D(p)$ for values of p less than 1 to see what happens when the city is not convex. For example, when $p = \frac{2}{3}$, the boundary curve $w(x)$ becomes an *astroid*.[4] It is remarkable that even for this concave city shape the value of $D(2/3) \approx 0.686\,397$ does not differ much from the value of $D(2) \approx 0.650\,403$ for a circular city.

[4]An astroid is the path of the center of mass of a ladder as it slides down a wall with one end of the ladder in contact with the wall and the other in contact with the floor.

The growth of the function $D(p)$ for small p can be obtained by using Laplace's method to evaluate the double integral for $M(p)$ in (14) in the limit of small p. The asymptotic behavior of $M(p)$ is given by

$$M(p) \sim \frac{64\pi}{p\sqrt{3}} 27^{-1/p} \quad (p \to 0+).$$

Using the Stirling formula to approximate the Gamma functions in $A(p)$ in (12) we obtain

$$D(p) \sim \frac{2p^{1/4}}{\pi^{1/4}\sqrt{3}} \left(\frac{32}{27}\right)^{1/p} \quad (p \to 0+)\,. \tag{16}$$

This asymptotic approximation is extremely accurate. At $p = \frac{1}{32}$ the numerical value of $D(p)$ is 82.153 831, while this asymptotic formula predicts the value 83.763 and at $p = \frac{1}{64}$ the numerical value of $D(p)$ is 16, 134.274 096, while this asymptotic formula predicts the value 16, 179. The relative error in the latter prediction is about 0.3%.

6.5 Variational Determination of the Boundary Curve

We now use variational methods to determine a differential equation satisfied by the boundary curve. We begin by simplifying the functionals $A[w]$ and $M[w]$.

6.5.1 Exploiting the Symmetry Across $y = x$

To prepare for our variational calculation we decompose $w(x)$ into its separate components $g(x)$ and $h(x)$. Then, using symmetry arguments, we eliminate all dependence on $g(x)$. Our optimization procedure will then determine the value of a. To begin we note that the area $A[w]$ in (3) is eight times the area in the positive octant above the line $y = x$:

$$A[h] = 8\int_{x=0}^{1} dx\,[h(x) - x] = 8\int_{x=0}^{1} dx\,h(x) - 4\,. \tag{17}$$

Next we turn to the quantity $M[w]$ in (6) and decompose $w(x)$ into $h(x)$ and $g(x)$:

$$\begin{aligned} M[g,h] &= 64\int_{x=0}^{1} dx\,xh(x)\int_{u=0}^{x} du\,h(u) \\ &\quad +64\int_{x=1}^{a} dx\,xg(x)\left[\int_{u=0}^{1} du\,h(u) + \int_{u=1}^{x} du\,g(u)\right]. \end{aligned} \tag{18}$$

Note that from (17) we can evaluate one of the integrals in terms of $A[h]$:

$$\int_{u=0}^{1} du\,h(u) = \frac{A}{8} + \frac{1}{2}\,.$$

Thus, $M[g,h]$ is

$$\begin{aligned} M[g,h] &= 64\int_{x=0}^{1} dx\, xh(x)\int_{u=0}^{x} du\, h(u) + (8A[h]+32)\int_{x=1}^{a} dx\, xg(x) \\ &\quad +64\int_{x=1}^{a} dx\, xg(x)\int_{u=1}^{x} du\, g(u)\,. \end{aligned}$$

To replace $g(x)$ by $h(x)$, we exploit the inverse relationship between the two functions: $h(x) = g^{-1}(x)$. Thus, if $g(x) = \gamma$ and $x : 1 \to a$, we have $\gamma : 1 \to 0$. Hence, we can write

$$x = g^{-1}(\gamma) = h(\gamma)\,,$$

and

$$dx = h'(\gamma)\, d\gamma = \frac{1}{g'(\gamma)}\, d\gamma\,.$$

Similarly, we let $g(u) = \alpha$, for $\alpha : 1 \to 0$ (and $u : 1 \to a$), implying that $u = g^{-1}(\alpha) = h(\alpha)$. Therefore, $du = h'(\alpha)\, d\alpha = \frac{1}{g'(\alpha)}\, d\alpha$. We then change the integration variables to obtain

$$\begin{aligned} M[h] &= 64\int_{x=0}^{1} dx\, xh(x)\int_{u=0}^{x} du\, h(u) - (4A[h]+16)\int_{x=0}^{1} dx\, x[h^2(x)]' \\ &\quad +32\int_{x=0}^{1} dx\, x[h^2(x)]'\int_{u=x}^{1} du\, uh'(u)\,. \end{aligned} \tag{19}$$

6.5.2 Preparing $M[h]$ for Functional Differentiation

It will be necessary to calculate the functional derivative of $M[h]$. To simplify this procedure we use integration by parts to remove all instances of the derivative function $h'(x)$. Integrating by parts four times and simplifying gives the following expression:

$$\begin{aligned} M[h] &= 64\int_{x=0}^{1} dx\, xh(x)\int_{u=0}^{x} du\, h(u) - 4A[h] - \frac{16}{3} + (4A[h]-16)\int_{x=0}^{1} dx\, h^2(x) \\ &\quad +\frac{32}{3}\int_{x=0}^{1} dx\, xh^3(x) + 32\int_{x=0}^{1} dx\, h^2(x)\int_{u=x}^{1} du\, h(u)\,. \end{aligned} \tag{20}$$

6.5.3 Performing the Functional Differentiation

We must minimize $\frac{M[h]}{(A[h])^{5/2}}$, subject to the constraint that $h(1) = 1$. We impose this constraint by introducing a Lagrange multiplier λ:

$$D[h] = \frac{M[h]}{(A[h])^{5/2}} + [h(1)-1]\lambda\,.$$

Note that if we differentiate with respect to λ, we recover the original constraint,

$$\frac{\partial D[h]}{\partial \lambda} = 0 \quad \Rightarrow \quad h(1) = 1\,.$$

We will also need the functional derivative of $A[h]$:

$$\frac{\delta A[h]}{\delta h(z)} = \frac{\delta}{\delta h(z)} \left[8 \int_{x=0}^{1} dx \, [h(x) - x] \right] = 8 \, . \tag{21}$$

Calculating $\delta M[h]/\delta h(z)$ We now calculate the functional derivative of $M[h]$ in (20). To prepare for functional differentiation we introduce the Heaviside step function to remove all variables from the limits of integration. The Heaviside step function $H(x)$ is defined as follows:

$$H(x) \equiv \begin{cases} 1 & (x \geq 0), \\ 0 & (x < 0). \end{cases} \tag{22}$$

We perform the following differentiation:

$$\begin{aligned} \frac{\delta M[h]}{\delta h(z)} &= \frac{\delta}{\delta h(z)} \left[64 \int_{x=0}^{1} dx \, xh(x) \int_{u=0}^{1} du \, h(u) H(x-u) \right. \\ &\quad + (4A[h] - 16) \int_{x=0}^{1} dx \, h^2(x) - 4A[h] - \frac{16}{3} + \frac{32}{3} \int_{x=0}^{1} dx \, xh^3(x) \\ &\quad \left. +32 \int_{x=0}^{1} dx \, h^2(x) \int_{u=0}^{1} du \, h(u) \, H(u-x) \right] \\ &= 64z \int_{u=0}^{z} du \, h(u) + 64 \int_{x=0}^{1} dx \, xh(x) \int_{u=0}^{1} du \, \delta(z-u) H(x-u) - 32 \\ &\quad +32 \int_{x=0}^{1} dx \, h^2(x) + (8A[h] - 32) \, h(z) + 32zh^2(z) \\ &\quad +64h(z) \int_{u=0}^{1} du \, h(u) H(u-z) + 32 \int_{x=0}^{1} dx \, h^2(x) H(z-x) \, . \end{aligned} \tag{23}$$

Finally, we remove the Heaviside function and restore the limits on the integrals:

$$\begin{aligned} \frac{\delta M[h]}{\delta h(z)} &= 64z \int_{u=0}^{z} du \, h(u) + 64 \int_{x=z}^{1} dx \, xh(x) - 32 + 32 \int_{x=0}^{1} dx \, h^2(x) \\ &\quad + (8A[h] - 32) \, h(z) + 32zh^2(z) + 64h(z) \int_{u=z}^{1} du \, h(u) \\ &\quad +32 \int_{x=0}^{z} dx \, h^2(x) \, . \end{aligned} \tag{24}$$

Calculating $\delta D[h]/\delta h(z)$ Now that we have obtained the functional derivative of $M[h]$ in (24), we can calculate the functional derivative of $D[h]$:

$$\frac{\delta D[h]}{\delta h(z)} = \frac{1}{(A[h])^{5/2}} \frac{\delta M[h]}{\delta h(z)} - \frac{5}{2(A[h])^{7/2}} 8M[h] + \lambda \, \delta(z-1) \, . \tag{25}$$

Since the only term containing a delta function is the last term, when we set $\frac{\delta D[h]}{\delta h(z)} = 0$, we learn that $\beta = 0$ and that $h(1) = 1$.

Substituting the expressions for $\frac{\delta M[h]}{\delta h(z)}$ in (24) and $M[h]$ in (20) into this equation, we obtain

$$\begin{aligned} 0 \;=\; & 8A[h]z\int_{u=0}^{z} du\, h(u) + 8A[h]\int_{x=z}^{1} dx\, xh(x) + 6A[h] - 6A[h]\int_{x=0}^{1} dx\, h^2(x) \\ & +A[h](A[h]-4)h(z) + 4A[h]zh^2(z) + 8A[h]h(z)\int_{u=z}^{1} du\, h(u) \\ & +4A[h]\int_{x=0}^{z} dx\, h^2(x) - 160\int_{x=0}^{1} dx\, xh(x)\int_{u=0}^{x} du\, h(u) + \frac{40}{3} \\ & +40\int_{x=0}^{1} dx\, h^2(x) - \frac{80}{3}\int_{x=0}^{1} dx\, xh^3(x) \\ & -80\int_{x=0}^{1} dx\, h^2(x)\int_{u=x}^{1} du\, h(u)\,. \end{aligned} \tag{26}$$

6.5.4 Expressing the Boundary as a Differential Equation

The integro-differential equation (26) expresses the shape of the boundary of the optimal city. Our objective now is to convert this equation to an ordinary differential equation. We begin by differentiating (26) with respect to z:

$$0 = 8zh(z)h'(z) + (A[h]-4)h'(z) + 8\int_{u=0}^{z} du\, h(u) + 8h'(z)\int_{u=z}^{1} du\, h(u)\,. \tag{27}$$

Next, we introduce the function $f(x)$, whose derivative is $h(x)$:

$$f(x) \equiv \int_{u=0}^{x} du\, h(u)\,, \tag{28}$$

so that $f'(x) = h(x)$ and

$$f(0) = 0\,. \tag{29}$$

Note that from (2) we have

$$f'(1) = h(1) = 1\,. \tag{30}$$

Also, (28) and (17) imply that

$$A[f] = 8f(1) - 4\,. \tag{31}$$

Finally, substituting the function $f(z)$ into (27) we obtain the following differential equation:

$$0 = zf'(z)f''(z) + [2f(1)-1]f''(z) + f(z) - f''(z)f(z)\,. \tag{32}$$

Note that if we evaluate this differential equation at $z = 1$ and $z = 0$, then we obtain the two conditions

$$f''(1) = h'(1) = -1, \tag{33}$$

and

$$f''(0) = h'(0) = 0. \tag{34}$$

The first of these conditions shows that the slope of the boundary curve at the point $(1, 1)$, where $h(x)$ joins onto $g(x)$ in the positive quadrant, is -1. This result implies that the boundary curve has a continuous derivative at this point and thus there is no cusp there. Similarly, the second of these conditions shows that the boundary curve at the point $(0, a)$ is level. Thus, again there is no cusp at this point. In short, the boundary of the optimal city has no dimples.

6.5.5 Substituting $f(z)$ Into $M[h]$

We now show that the functional $M[h]$ simplifies dramatically when $M[h]$ is evaluated at the optimal boundary curve. We substitute $f(x)$ into (20). Then we use the differential equation (32) to simplify the result:

$$\begin{aligned} M[f] &= 64\int_{x=0}^{1} dx\, xf'(x)\int_{u=0}^{x} du\, f'(u) - 4A[f] - \frac{16}{3} + (4A[f] - 16)\int_{x=0}^{1} dx\, f'^2(x) \\ &\quad + \frac{32}{3}\int_{x=0}^{1} dx\, xf'^3(x) + 32\int_{x=0}^{1} dx\, f'^2(x)\int_{u=x}^{1} du\, f'(u)\,. \end{aligned} \tag{35}$$

Performing the indicated integrations and doing repeated integration by parts, we simplify this equation to

$$\begin{aligned} M[f] &= 64\int_{x=0}^{1} dx\, xf'(x)f(x) - 4A[f] - 32f(1) + 16 \\ &\quad + \{4A[f] + 32f(1) - 16\}\int_{x=0}^{1} dx\, f'^2(x) - 64\int_{x=0}^{1} dx\, x^2 f'^2(x)f''(x) \\ &\quad + 64\int_{x=0}^{1} dx\, xf(x)f'(x)f''(x)\,. \end{aligned} \tag{36}$$

Next, in the last two integrals we use the differential equation (32) to replace the expression $f''(x)f(x) - xf'(x)f''(x)$, which is cubic in f, by the quadratic expression $[2f(1) - 1]f''(x) + f(x)$. The resulting formula for $M[f]$ simplifies, after further integration by parts, to

$$M[f] = -64\int_{x=0}^{1} dx\, f^2(x) + 64f^2(1)\,, \tag{37}$$

where we have used the formula $A[f] = -4 + 8f(1)$ in (31).

6.5.6 Numerical Results

We have now completed the analytical work and reduced the problem of finding the boundary of the optimal city to a numerical computation. The numerical procedure is a follows: First, we guess a value, say c, for $f(1)$. Also, we know from (30) that

$f'(1) = 1$. From these two initial conditions we solve the differential equation (32) numerically to obtain the value of $f(0)$. But, from (29) we know that $f(0) = 0$. By repeated shooting, we determine the value of c that gives the result $f(0) = 0$. When we achieve this condition, we then learn that the value of $a = f'(0) = h(0) = 1.463\,110\,117\,728$ [see (2)]. The shape of the boundary $h(x)$ is given by the derivative of $f(x)$: $h(x) = f'(x)$. Once we have found $f(x)$, we then compute the value of $D[f]$ from

$$D[f] = \frac{M[f]}{(A[f])^{5/2}} = \frac{64 f^2(1) - 64 \int_{x=0}^{1} dx\, f^2(x)}{[8f(1) - 4]^{5/2}}\,. \tag{38}$$

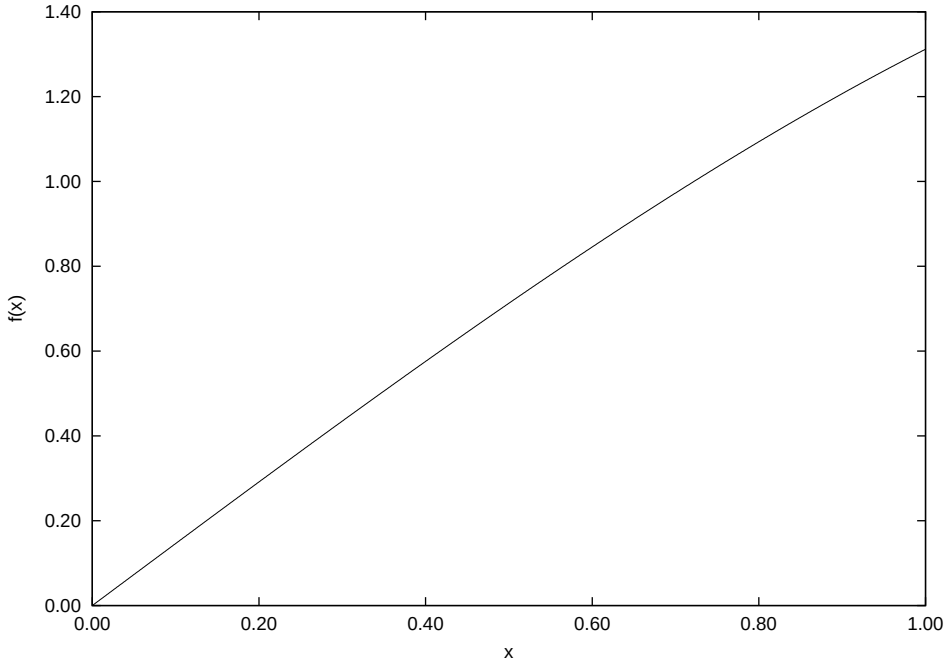

Figure 17: Plot of $f(x)$ for $0 \le x \le 1$.

Our numerical analysis reveals that the minimum value of $D[f]$ is $0.650\,245\,952\,951$, which is about a 2.6 parts-per million improvement in the optimal value of D obtained by using the one-parameter family of functions $w(x) = (2 - |x|^p)^{1/p}$. The optimal function $f(x)$ is displayed in Figure 17, and the optimal boundary curve $w(x)$ is shown in Figure 18. Note that $f(0) = 0$ and that $f'(1) = 1$. By repeated iteration (shooting) we find that $f(1) = 1.311\,794\,482\,332$. The derivative of the plotted function $f(x)$ gives the boundary of the optimal city and for this city the value of D is $0.650\,245\,952\,951$.

The function $w(x)$ is composed of $h(x)$ for $0 \le x \le 1$ and $g(x) = h^{-1}(x)$ for $x \ge 1$. Note that $h(0) = f'(0) = a = 1.463\,110\,117\,728$. The number a is the value of x at which $g(x)$ crosses the x axis. The function h continues onto the function g at the point $(1, 1)$, which is marked by a $+$ sign. For this optimal boundary curve the

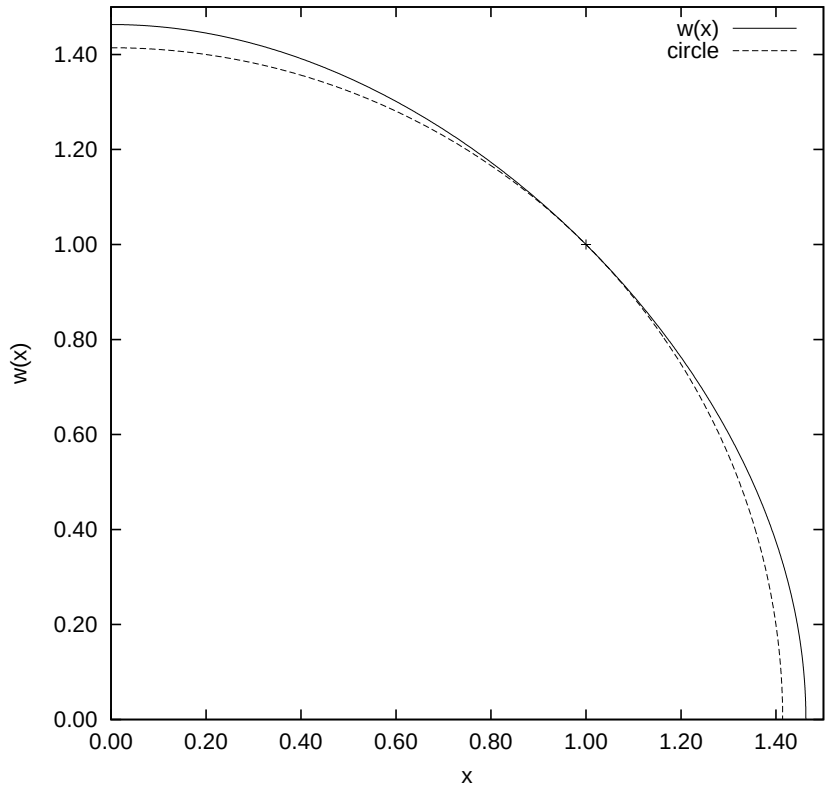

Figure 18: Plot of the function $w(x)$ for $x \geq 0$.

(minimum) numerical value of the functional D is 0.650 245 952 951. For purposes of comparison, the dotted line is a quarter circle, illustrating that the optimal city is close to circular.

6.6 Discussion and Conclusions on Continuous Problem

The principal conclusion that can be drawn from the calculations in this section is that if a city is tightly clustered and convex, then it is extremely close to optimal; that is, its value of $D[w]$, where w is the boundary curve of the city, is close (just a few percent off) to the optimal value 0.650 245 952 951. Thus, an allocation of computer processors, so long as they are tightly bunched and convex, will be close to optimal as measured by the average pairwise Manhattan distance metric.

We already know from Table 6.4.4 that if a city departs markedly from convexity (say, it is shaped like an astroid) then it will be far from optimal. It is also interesting that if the city has holes then the corresponding value of the functional D can also increase significantly. To illustrate we consider the case of an annular city, whose outside diameter is 1 and whose inside diameter is r. Note that the hole of radius r in the center of the city may be one of two possible types, a park or a lake. If the hole

is a park, then the notion of the Manhattan distance between two points in the city does not change (one can walk through the park). If the hole is a lake, one must walk around it, and the definition of the distance between two points in the city changes accordingly.

Let us consider the simpler case of a park. The area of this city is $A(r) = \pi(1-r^2)$. Also, we calculate that the pairwise Manhattan distance is

$$\begin{aligned} M(r) &= \frac{512}{45}\left(1+r^5\right) - \frac{8}{3}\pi\left(2r^2+r^4\right){}_2F_1\left(-\frac{1}{2},\frac{1}{2};3;r^2\right) \\ &\quad +\frac{2}{9}\pi\left(r^4-r^6\right){}_2F_1\left(\frac{1}{2},\frac{3}{2};4;r^2\right), \end{aligned}$$

where ${}_2F_1(a,b;c;z)$ is the hypergeometric function [48, 38]. The value of $M(r)$ for this city reduces to that in Table 6.4.4 for a disk when $r = 0$ and it vanishes when $r = 1$ (the annulus becomes a circle). For this city, the value of $D(r)$ in (5), which measures the optimality of the city shape, grows rapidly with r: $D(0) \approx 0.650$, $D(0.3) \approx 0.713$, $D(0.6) \approx 0.947$, $D(0.9) \approx 1.998$, $D(0.99) \approx 6.531$, and $\lim_{r\to 1} D(r) = \infty$.

There are many interesting possible continuations of this study. For example, one can consider cities whose dimension is higher than two. For such cities, the boundary is a surface rather than a curve, and thus the boundary is described by a partial differential equation rather than an ordinary differential equation. Hence, the calculations are much more elaborate and have not been considered here.

Another possible generalization of this work is to consider metrics other than the Manhattan distance (p-norms where $p \neq 1$). (Of course, the Euclidean metric, for which $p = 2$ is trivial, and in this case the optimal city is a disk.) Additionally, one could minimize the average of the *squares* of the pairwise distances for the city. Squares of distances are commonly considered in clustering applications.

Intentionally Left Blank

7 Scheduling

The primary focus of research in job scheduling has been to increase utilization and improve desired user metrics such as turnaround time and slowdown. Turnaround time is equal to waiting time plus runtime. Slowdown is turnaround time divided by runtime. Very little research so far has addressed fairness in job scheduling. The Cplant scheduler uses a "fair share" measure to order jobs in the queue, but how fair is the scheduler?

One possible approach to assessing fairness is as follows: For each job, find the number of jobs with a higher user usage-count that are serviced while the job waits. The problem with that approach is how do we account for "benign" backfilling that uses slots in the schedule not usable by this job? Instead, we propose assigning a "fair-start" time to a job when it is submitted by generating a non-backfilling, in order schedule based on fairness priority. If a job's actual start time does not exceed its fair-start time, the job is considered to have been fairly treated. Otherwise, it is considered to have been unfairly treated.

The original Cplant scheduler implemented no guarantee backfilling. With no guarantee backfilling none of the jobs submitted gets a scheduler reservation. Therefore, there is no blocking of jobs that can backfill onto open processors. We have studied the maximum average utilization of parallel computers in the absence of blocking [69].

To preventing starvation, the Cplant scheduler will drain the machine when a job has been waiting for more than twenty-four hours. During the time that Cplant is being drained, many processors can go idle while jobs that can fit on the machine are waiting. In order to increase processor utilization and decrease job waiting times, aggressive/easy backfilling was added to the Cplant scheduler drain [58]. With aggressive backfilling, a job that can fit on Cplant is scheduled during the time that the machine is being drained as long as the time requested for the job is no greater than the maximum remaining time for the jobs running on the machine. Therefore, one job gets a reservation and backfilling can be blocked by that job.

When Cplant became oversubscribed, the no guarantee backfilling and starvation drain made Cplant "unfair". We studied the consequences to "fairness" of various alternatives to the above schedular. This work is also reported in [93]. Some of this work was funded by CSRF and the CSRI.

This project also supported research into scheduling by Bunde [19]. This work began by showing a structural similarity between the uniprocessor Shortest Remaining Processing Time (SRPT) and Shortest Processing Time (SPT) schedules. The structural similarity is then used to prove that SPT is $(\Delta+1)/2$-competitive for total flow, where Δ is the ratio between the lengths of the longest and shortest jobs. This ratio is the best-possible for a nonpreemptive algorithm and an improvement on the ratio of $\Delta+1$ shown by Epstein and van Stee [37]. In the multiprocessor setting, the same argument gives an $O(\Delta \log \min\{n, \Delta\})$-competitive nonpreemptive algorithm,

the first nonpreemptive algorithm shown to be competitive for flow. A number of lower bounds are also shown.

Bunde [19] also considers offline scheduling by busy algorithms, i.e. those which can never be unnecessarily idle. This study was directly motivated by Sandia's desire to keep machines busy whenever possible. Unfortunately, it is shown that neither uniprocessor nor multiprocessor total flow can be approximated to within a factor of either $\Delta^{1-\epsilon}$ or $n^{1-\epsilon}$ for any constant $\epsilon > 0$ unless P=NP. The proof of these facts closely follows proofs by Kellerer et al. [61] and Leonardi et al. [66].

The remainder of this section is organized as follows. Subsection 7.1 describes the baseline Cplant scheduler. Subsection 7.2 describes some alternatives to increase fairness. Subsection 7.3 describes our simulation environment. Subsection 7.4 gives our results. Subsection 7.5 gives our conclusions.

7.1 Baseline Cplant Scheduler

The baseline Cplant scheduler implements the following:

- no guarantee backfilling,
- fair share queue priority using decaying node-hours, and
- jobs with a wait time of greater than twenty-four hours are considered to be starving and are treated as follows:
  - they are placed in a virtual queue by receiving a higher priority than non-starving jobs,
  - they are sorted in first come first serve (FCFS) order instead of by fair share, and
  - the head of the queue has a reservation (aggressive backfilling).

The negative implications of the baseline Cplant schedular are that jobs do not necessarily run in fair share order, unfair use of the machine is allowed by the no guarantee backfilling and FCFS "starvation queue" order, and unbounded wait times and starvation forces system administrators to start jobs manually. However, "good" utilization and average user metrics are likely.

7.2 Increasing Fairness

Suggestions to improve fairness include limiting runtime, increasing the starvation limit from twenty-four to seventy-two hours or greater, not allowing a "starvation reservation" for users who are "hogging" the machine, and using conservative backfilling. Limiting runtime would cap runtimes at seventy-two hours. This improves

fairness and user metrics by allowing "preemption". Scripts have already been developed to help ease the potential burden on the user, and minimal impact is expected on fair long jobs.

Increasing the starvation limit from twenty-four to seventy-two hours or greater reduces the "unfairness" due to the FCFS starvation queue. This would still prevent jobs from starving and minimize impact on standard average user metrics and utilization is expected. However, this does not address the lack of fairness due to no guarantees.

Not allowing a starvation reservation for users who are hogging the machine introduces fairness to the "virtual" starvation queue. Again, minimal impact on standard user metrics and utilization is expected. This is a simple change to the existing scheduler and minimal impact to the normal users.

Using conservative backfilling would eliminate starvation altogether by giving each job a reservation when it arrives. This would give a deterministic worst case start time upon submittal of each job. Conservative backfilling would have a FCFS "feel", because each job receives an initial reservation in arrival order. Backfill would be sorted by "fair-share", but an unfair job can still delay a fair job.

To remove the FCFS feel of conservative backfilling, dynamic reservations could be recompute based on fairness when a new job arrives. A job can never delay a more "fair" job. Starvation is possible, but the user has control over it. If a user does not submit additional jobs or add artificial dependencies, progress in the queue is guaranteed. This implements the spirit of the fair share policy. No unfair job will ever delay a more fair job

7.3 Simulation Environment

We built an event-driven simulator to test the fairness strategies from Subsection 7.2. The objective of the simulator is to exhibit tendencies rather than to predict behaviors precisely. Our simulations suggest that the alternatives individually and in combination tend to increase fairness. A variety of performance metrics gauge the performance of the alternatives.

The simulator was run on traces from December 2002–June 2003. These trace files contain data about all jobs submitted to Ross, a Cplant machine configured as a three dimensional mesh with 1024 compute processors. The trace file includes the times that the jobs were dispatched to the scheduler, the numbers of processors requested, the running times requested, and the actual running times.

7.4 Results

Simulation results are presented for the baseline Cplant policy, small tweaks to the baseline, and more fundamental changes to the baseline.

7.4.1 Small Tweaks

The baseline Cplant policy has a starvation limit of 24 hours. Any job can starve, and there is no maximum runtime. We made small tweaks to each of these individually and all of them combined.

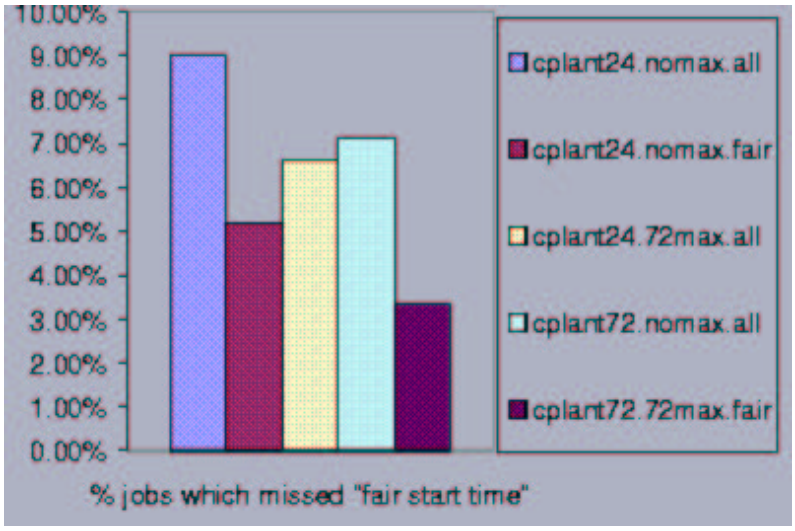

Figure 19: The percentage of jobs that missed their fair start time for the baseline and the small tweaks.

The starvation limit was changed to 72 hours. The runtime was limited to 72 hours, and only fair jobs could starve. The names in the legends for Figures 19 and 20 give these parameter choices in this order. All the changes reduced the percentage of jobs that missed their fair start time. However, combining all three "enhancements" shows the most improvement.

Loss of capacity is generally slightly lower. We report loss of capacity instead of utilization because in simulations, utilization is distorted by the tail effect of the last jobs run. Loss of capacity is the percentage of nodes idle when the smallest job waiting cannot start.

Figures 21 and 22 show each users wait time as a function of their usage for the baseline and all three tweaks combined. Note that "heavy" users with high wait time under the baseline benefit when all three tweaks are combined. Even though all three tweaks combined did manage to reduce the "extreme" wait time for mid-range users, a very light user actually gets worst. This still seems unfair.

7.4.2 Fundamental Changes

Conservative backfilling and dynamic reservations are the more fundamental changes described in Subsection 7.2. Since the 72 hour runtime limit is compatible with these changes, we also consider it with these changes.

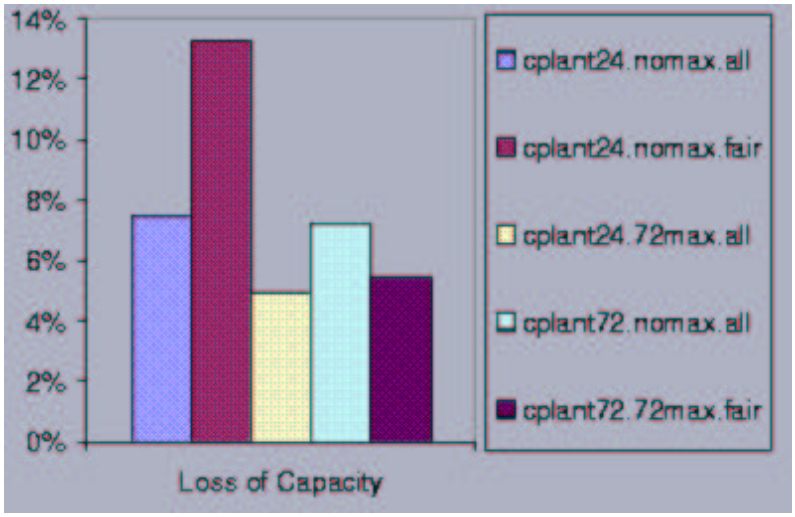

Figure 20: The loss of capacity for the baseline and the small tweaks.

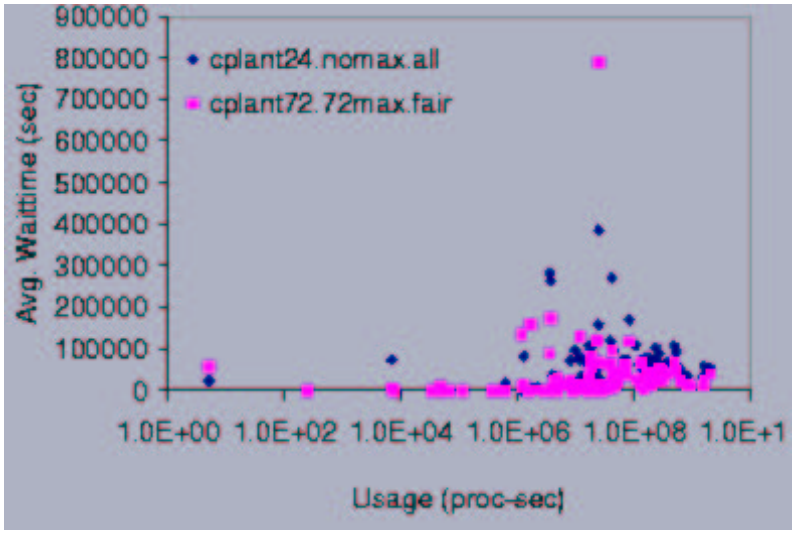

Figure 21: The average waiting time for each user as a function of their usage in processor seconds for the baseline and all tweaks combined.

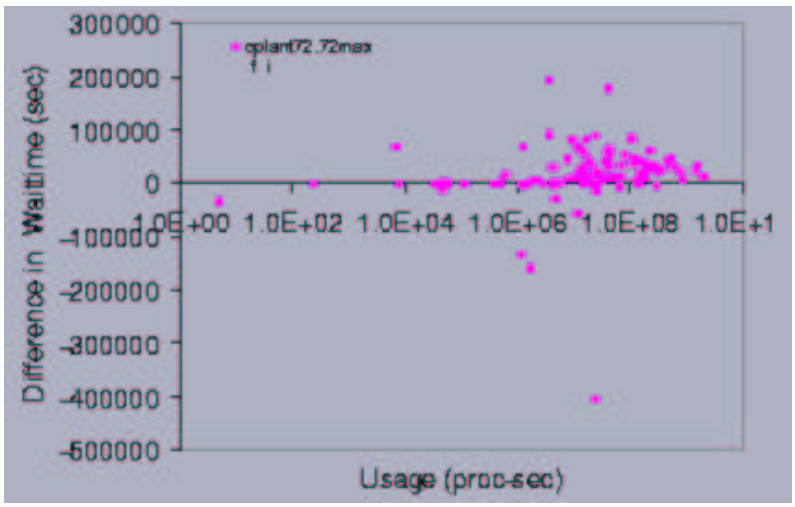

Figure 22: The difference in average waiting time for each user as a function of their usage in processor seconds for the baseline and all tweaks combined.

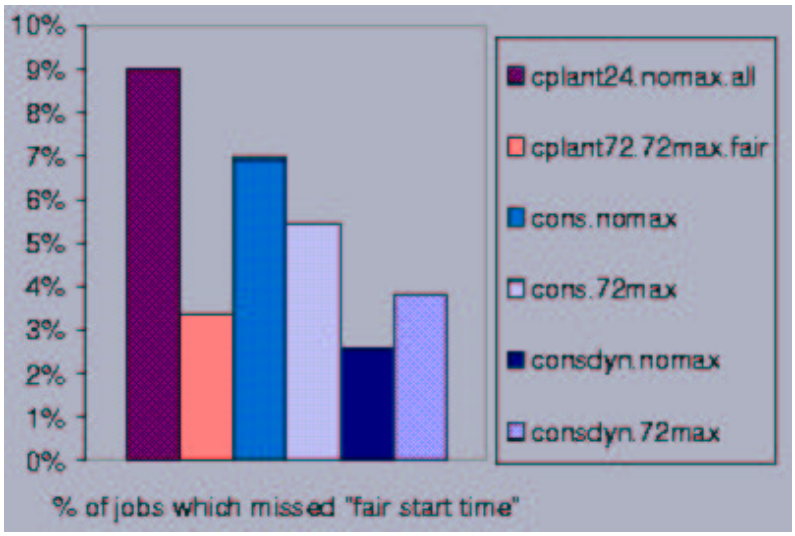

Figure 23: The percentage of jobs that missed their fair start time for the baseline, all three tweaks combined, and more fundamental changes.

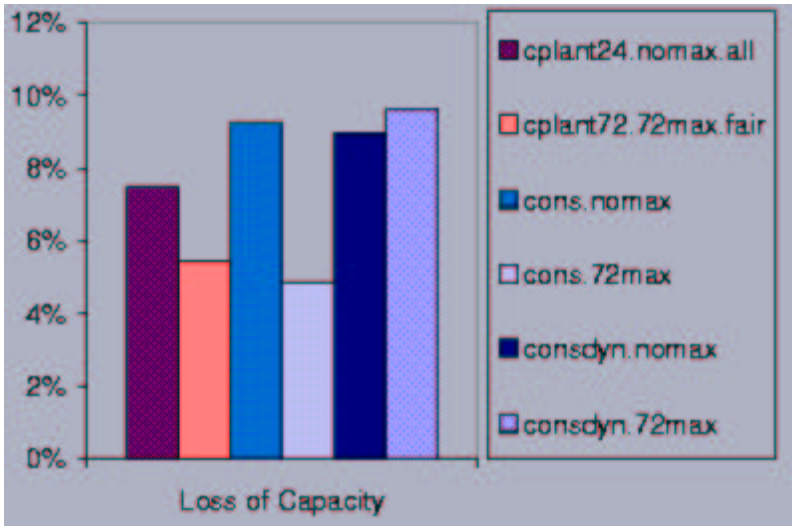

Figure 24: The loss of capacity for the baseline, all three tweaks combined, and more fundamental changes.

Figure 23 and 24 show the results for the baseline, the best policy from Subsection 7.4.1, and these more fundamental changes. Figure 23 shows that it is possible to further reduce the percentage of jobs that miss their fair start time with conservative backfilling and dynamic reservations. However, conservative backfilling without dynamic reservations can be bad for fairness, and there is a small increase (approximately three percent) in loss of capacity (for dynamic reservations).

Figures 25 and 26 show that heavy users are appropriately penalized, and light users are given better treatment. However, medium users can still perform worse than heavy users.

Even though the seventy-two hour runtime limit increased the number of jobs that missed their fair start time, Figures 27 and 28 show that when the seventy-two hour runtime limit is added to conservative backfilling with dynamic reservations, most users improve, and there is not dramatic increase in wait time.

7.5 Conclusions on Scheduling

We have proposed a new way to quantitatively assessing how well fairness is implemented by a scheduling policy. The current Cplant scheduling policy causes unfair treatment of about 10% of jobs. The effect of several possible changes to scheduling policy were evaluated through simulations. These changes included changing the starvation threshold (from 24 to 72 hours), imposing a maximum time limit for jobs, disallowing unfair jobs from the starvation queue, and using reservations for all jobs (conservative backfill variations) instead of a starvation queue mechanism. Simulations show that modifications can reduce unfairness to under 3% of jobs.

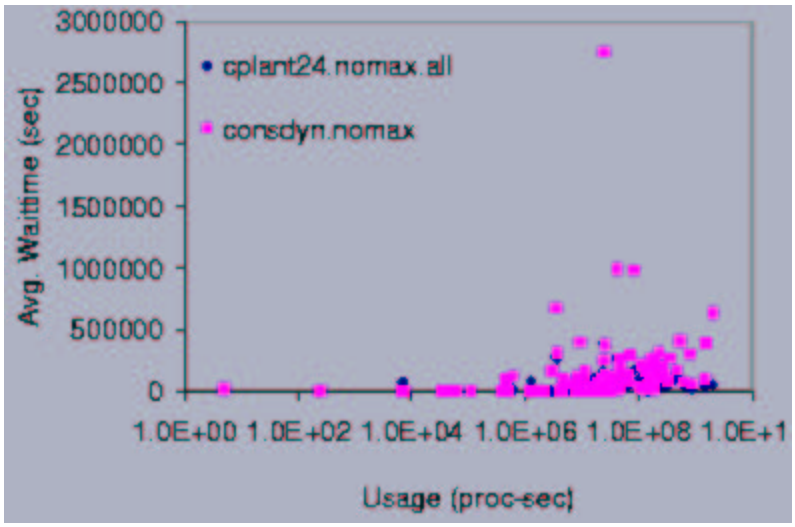

Figure 25: The average waiting time for each user as a function of their usage in processor seconds for the baseline and conservative backfilling with dynamic reservations.

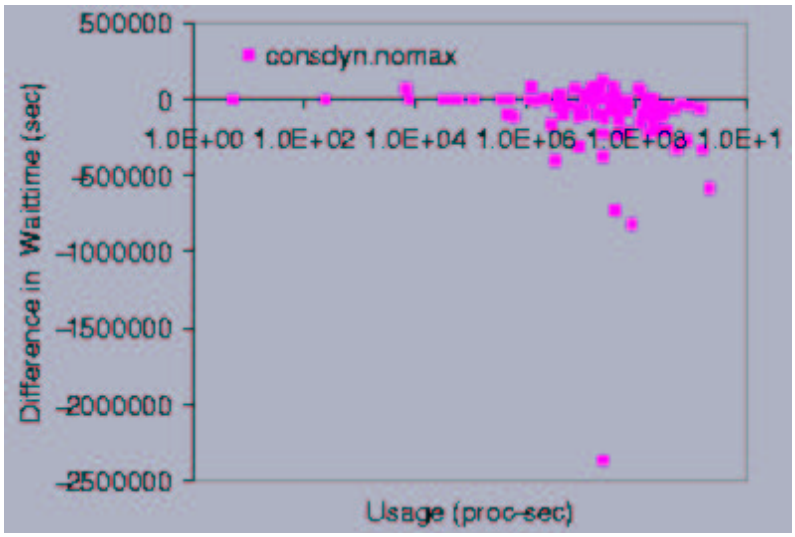

Figure 26: The difference in average waiting time for each user as a function of their usage in processor seconds for the baseline and conservative backfilling with dynamic reservations.

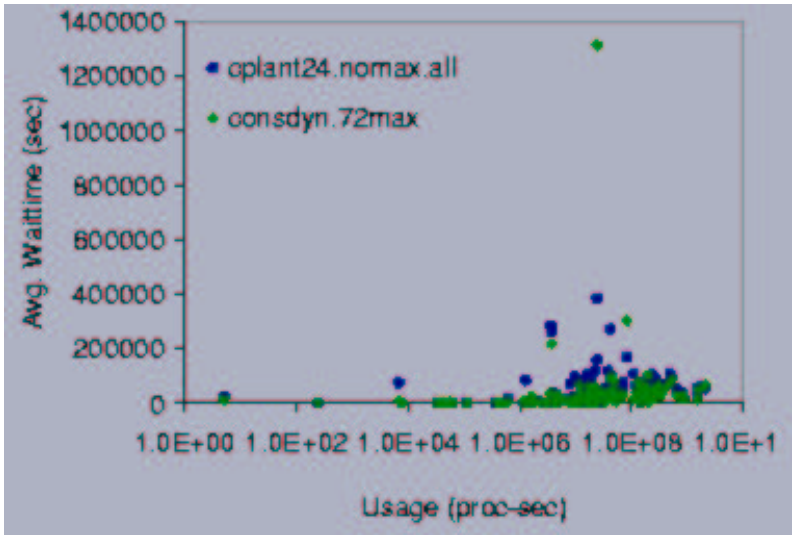

Figure 27: The average waiting time for each user as a function of their usage in processor seconds for the baseline and conservative backfilling with dynamic reservations and seventy-two hour runtime limits.

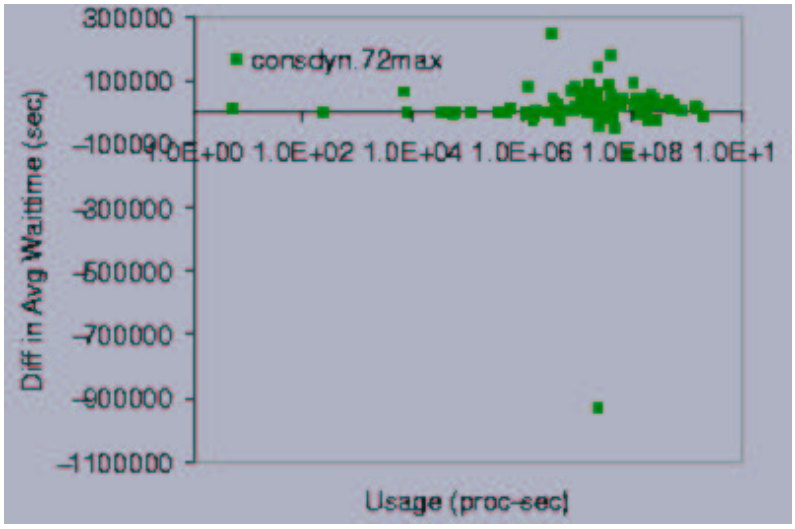

Figure 28: The difference in average waiting time for each user as a function of their usage in processor seconds for the baseline and conservative backfilling with dynamic reservations and seventy-two hour runtime limits.

Intentionally Left Blank

8 Conclusions

For commodity-based parallel computing systems, we have provided algorithmic support in three areas: routing, allocation and scheduling. Based on a generalization of Dally and Seitz routing, we provided shortest-path routes for Antarctica. Recent experiments on Antarctica indicate that these routes may decrease completion time for some application codes by at least fifty percent over the routes generated by Myricom's simple_routes program. Based on this work in Cplant routing, we were also able to have an impact on the routing for Red Storm. The Red Storm routing work was funded by CSRF.

Our accomplishments in allocation can be grouped into one of three areas: one-dimensional reduction, minimizing the number of communication hops, and the continuum problem. Preliminary simulations and Cplant experiments indicate that both space-filling curves and one-dimensional packing improve processor locality compared to the sorted free list strategy previously used on Cplant. These new allocation strategies are implemented in Release 2.0 of the Cplant System Software that was phased into the Cplant systems at Sandia by May 2002.

Experimental results then demonstrated that the *average number of communication hops* between the processors allocated to a job strongly correlates with the job's completion time. To minimize the number of communication hops, we achieved the following results: we found a linear-time exact algorithm in one dimension. We found an efficient two-approximation algorithm in d dimensions, for any constant d. We also found some lower bounds for this algorithm. We found two efficient polynomial-time approximation schemes in two dimensions. We have formulated the exact problem in d dimensions as a quadratic program which can be solved as either a semidefinite program or an integer linear program. Because the two-approximation balances solution quality and simplicity, we believe this is the correct processor-allocation algorithm to implement in supercomputers. Indeed, it is slated for inclusion in Cplant System Software, Version 2.1, to be released. In one preliminary test with an application code, the two-approximation allocation ran sixteen percent faster than the allocation given by the one-dimensional reduction above [47].

For the continuum problem, we formulated this optimization problem in mathematical terms. Then we considered a one-parameter family of clusters and found the shape of the optimal cluster in this limited family. From this calculation we saw that the optimal city is not circular. Next we carried out a full variational calculation and determined an interesting nonlinear differential equation satisfied by the boundary of the optimal cluster. The optimal cluster is nearly circular in shape but is not a disk. Our principal conclusions are that so long as the allocation of processors are clustered in a tight shape having no holes of only small holes, the allocation will be extremely close to optimal.

In scheduling, we proposed a new way to quantitatively assess how well fairness is implemented by a scheduling policy. The current Cplant scheduling policy

causes unfair treatment of ten percent of jobs. The effect of several possible changes to scheduling policy were evaluated through simulations. These changes included changing the starvation threshold (from twenty-four to seventy-two hours), imposing a maximum time limit for jobs, disallowing unfair jobs from the starvation queue, and using reservations for all jobs (conservative backfill variations) instead of a starvation queue mechanism. Simulations show that modifications can reduce unfairness to under three percent of jobs. Some of this work was also supported by CSRF and the CSRI.

References

[1] J. Alber and R. Niedermeier. On multidimensional Hilbert indexings. *Theory of Computing Systems*, 33:295–312, 2000.

[2] V. V. Alexandrov, A. I. Alexeev, and N. D. Gorsky. A recursive algorithm for pattern recognition. In *Proc. IEEE Intl. Conf. Pattern Recognition*, pages 431–433, 1982.

[3] S. Aluru and F. Sevilgen. Parallel domain decomposition and load balancing using space-filling curves. In *Proc. 4th International Conference on High-Performance Computing*, pages 230–235, 1997.

[4] A. Ansari and A. Fineberg. Image data compression and ordering using Peano scan and lot. *IEEE Trans. on Consumer Electronics*, 38(3):436–445, 1992.

[5] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense NP-hard problems. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 284–293, 1995.

[6] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *JACM*, 45:891–923, 1998.

[7] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense graph. *Journal of Algorithms*, 34:203–221, 2000.

[8] Y. Bartal, M. Charikar, and D. Raz. Approximating min-sum k-clustering in metric spaces. In *Proc. 33rd Symp. on Theory of Computation*, pages 11–20, 2001.

[9] S. Baylor, C. Benveniste, and Y. Hsu. Performance evaluation of a massively paralel I/O subsystem. In R. Jain, J. Werth, and J. Browne, editors, *Input/Output in parallel and distributed computer systems*, volume 362 of *The Kluwer International Series in Engineering and Computer Science*, chapter 13, pages 293–311. Kluwer Academic Publishers, 1996.

[10] C. M. Bender, M. A. Bender, E. D. Demaine, and S. P. Fekete. What is the optimal shape of a city? unpublished manuscript, 2003.

[11] M. Bender, B. Bradley, G. Jagannathan, and K. Pillaipakkamnatt. The robustness of the sum-of-squares algorithm for bin packing. unpublished manuscript, 2003.

[12] M. A. Bender, D. P. Bunde, E. D. Demaine, S. P. Fekete, V. J. Leung, and H. M. C. A. Phillips. Communication-aware processor allocation for supercomputers. unpublished manuscript, 2003.

[13] M. A. Bender, E. Demaine, and M. Farach-Colton. Cache-oblivious B-trees. In *Proc. 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 399–409, 2000.

[14] M. A. Bender, Z. Duan, J. Iacono, and J. Wu. A locality-preserving cache-oblivious dynamic dictionary. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 29–38, 2002.

[15] S. Bhattacharya and W.-T. Tsai. Lookahead processor allocation in mesh-connected massively parallel computers. In *Proc. 8th International Parallel Processing Symposium*, pages 868–875, 1994.

[16] R. Boppana and M. M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32(2):180–196, 1992.

[17] R. Brightwell, L. A. Fisk, D. S. Greenberg, T. Hudson, M. Levenhagen, A. B. Maccabe, and R. Riesen. Massively parallel computing using commodity components. *Parallel Computing*, 26(2-3):243–266, 2000.

[18] G. S. Brodal, R. Fagerberg, and R. Jacob. Cache oblivious search trees via binary trees of small height (extended abstract). In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2002.

[19] D. Bunde. Approximating total flow time. Master's thesis, Univ. Illinois at Urbana-Champaign, Dec 2002.

[20] D. Bunde and V. Leung. Scheduling without co-allocation. unpublished manuscript, 2003.

[21] R. D. Carr. private communication, June 2003.

[22] C. Chang and P. Mohapatra. Improving performance of mesh connected multicomputers by reducing fragmentation. *Journal of Parallel and Distributed Computing*, 52(1):40–68, 1998.

[23] S. Chatterjee, A. R. Lebeck, P. K. Patnala, and M. S. Thottethodi. Recursive array layouts and fast matrix multiplication. In *Proc. 11th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 222–231, 1999.

[24] P.-J. Chuang and N.-F. Tzeng. An efficient submesh allocation strategy for mesh computer systems. In *Proc. International Conf. on Distributed Computer Systems*, pages 256–263, 1991.

[25] F. R. K. Chung, M. R. Garey, and D. S. Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods*, 3:66–76, 1982.

[26] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 2. PWS Publishing Company, 1997.

[27] D. Coppersmith and P. Raghavan. Multidimensional online bin packing: Algorithms and worst case analysis. *Operations Research Letters*, 8:17–20, 1989.

[28] Cray Inc. Network queuing environment. `http://www.cray.com/products/software/nqe.html`.

[29] J. Csirik, D. Johnson, C. Kenyon, J. Orlin, P. Shor, and R. Weber. On the sum-of-squares algorithm for bin packing. In *Proc. 32nd Annual ACM Symposium on Theory of Computation (STOC)*, pages 208–217, 2000.

[30] J. Csirik, D. Johnson, C. Kenyon, P. Shor, and R. Weber. A self-organizing bin packing heuristic. In *Proc. Algorithm Engineering and Experimentation: International Workshop (ALENEX)*, volume 1619 of *Springer Lecture Notes in Computer Science*, pages 246–265, 1999.

[31] J. Csirik and A. van Vliet. An on-line algorithm for multidimensional bin packing. *Operation Research Letters*, 13:149–158, 1993.

[32] J. Csirik and G. Woeginger. On-line packing and covering problems. In A. Fiat and G. Woeginger, editors, *On-Line Algorithms—The State of the Art*, Lecture Notes in Computer Science, chapter 7. Springer-Verlag, 1998.

[33] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[34] N. W. Dauchy. private communication, May 2001.

[35] M. Dickerson, R. Drysdale, and J.-R. Sack. Simple algorithms for enumerating interpoint distances and finding k nearest neighbors. *International J. Computational Geometry and Applications*, 2(3):221–239, 1992.

[36] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete and Computational Geometry*, 11(3):321–350, 1994.

[37] L. Epstein and R. van Stee. Optimal on-line flow time with resource augmentation. In *Proc. 13th Symp. Fund. Comp. Theory*, number 2138 in LNCS, pages 472–482. Springer-Verlag, 2001.

[38] A. Erdélyi, W. Magnus, F. Oberhettinger, and F. G. Tricomi. *Higher Transcendental Functions*, volume 1. McGraw-Hill, New York, 1953.

[39] Eric Weisstein's World of Mathematics. Hilbert curve. `http://mathworld.wolfram.com/HilbertCurve.html`.

[40] S. Fekete and H. Meijer. Maximum dispersion and geometric maximum weight cliques. *Algorithmica*, to appear.

[41] S. P. Fekete, J. S. B. Mitchell, and K. Weinbrecht. On the continuous weber and k-Median problems. In *Proceedings of the 16th Annual Symposium on Computational Geometry (SCG)*, pages 70–79, 2000.

[42] S. P. Fekete, J. S. B. Mitchell, and K. Weinbrecht. On the continuous weber and k-Median problems. *Operations Research*, 2003. To appear.

[43] J. D. Frens and D. S. Wise. Auto-blocking matrix-multiplication or tracking BLAS3 performance from source code. *ACM SIGPLAN Notices*, 32(7):206–216, 1997.

[44] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 285–297, 1999.

[45] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[46] C. Gotsman and M. Lindenbaum. On the metric properties of discrete space-filling curves. *IEEE Trans. on Image Processing*, 5(5):794–797, 1996.

[47] S. Goudy. private communication, January 2003.

[48] I. S. Gradsteyn and I. M. Ryzhik. *Table of Integrals, Series, and Products*. Academic, New York, 1965.

[49] M. Griebel and G. W. Zumbusch. Hash-storage techniques for adaptive multilevel solvers and their domain decomposition parallelization. In J. Mandel, C. Farhat, and X.-C. Cai, editors, *Proc. Domain Decomposition Methods, (DD)*, number 218 in Contemporary Mathematics, pages 279–286, Providence, 1998. AMS.

[50] N. Guttmann-Beck and R. Hassin. Approximation algorithms for minimum sum p-clustering. *Disc. Appl. Math.*, 89:125–142, 1998.

[51] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21:133–137, 1997.

[52] J. Hàstad. Testing of the long code and hardness for clique. In *Proc. 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19, 1996.

[53] D. Hilbert. Über die stetige abbildung einer linie auf ein flachenstück. *Math. Ann.*, 38:459–460, 1891.

[54] P. Indyk. A sublinear time approximation scheme for clustering in metric spaces. In *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.

[55] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1973.

[56] D. S. Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8:272–314, 1974.

[57] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3:256–278, 1974.

[58] J. R. Johnston. private communication, October 2001.

[59] S. Kamata, R. O. Eason, and Y. Bandou. A new algorithm for N-dimensional Hilbert scanning. *IEEE Trans. on Image Processing*, 8(7):964–973, 1999.

[60] S. Kamata, R. O. Eason, and E. Kawaguchi. An implementation of the Hilbert scanning algorithm and its application to data compression. *IEICE Trans. on Information and Systems*, E76-D(4):420–428, Apr. 1993.

[61] H. Kellerer, T. Tautenhahn, and G. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proc. 28th Symp. on Theory of Computing*, pages 418–426, 1996.

[62] G. Kortsarz and D. Peleg. On choosing a dense subgraph. In *Proceedings of the 34th IEEE Annual Symposium on Foundations of Computer Science*, pages 692–701, Palo Alto, CA, 1993.

[63] P. Krueger, T.-H. Lai, and V. Dixit-Radiya. Job scheduling is more important than processor allocation for hypercube computers. *IEEE Trans. on Parallel and Distributed Systems*, 5(5):488–497, 1994.

[64] S. Krumke, M. Marathe, H. Noltemeier, V. Radhakrishnan, S. Ravi, and D. Rosenkrantz. Compact location problems. *Theoretical Computer Science*, 181(2):379–404, 1997.

[65] Lawrence Livermore National Laboratory. Advanced Simulation and Computing (ASCI). `http://www.llnl.gov/asci/`.

[66] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proc. 29th Symp. on Theory of Computation*, pages 110–119, 1997.

[67] V. J. Leung. Deadlock free routing on red storm. unpublished manuscript, 2003.

[68] V. J. Leung, E. M. Arkin, M. A. Bender, D. P. Bunde, J. R. Johnston, A. Lal, J. S. B. Mitchell, C. A. Phillips, and S. S. Seiden. Processor allocation on Cplant: achieving general processor locality using one-dimensional allocation strategies. In *Proc. 4th IEEE International Conference on Cluster Computing*, pages 296–304, 2002.

[69] V. J. Leung and J. M. DeLaurentis. Maximum utilization of parallel computers. In *Proc. Hawaii International Conference on Statistics*, 2002.

[70] K. Li and K.-H. Cheng. A two-dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system. *Journal of Parallel and Distributed Computing*, 12:79–83, 1991.

[71] V. Lo, K. Windisch, W. Liu, and B. Nitzberg. Non-contiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Transactions on Parallel and Distributed Computing*, 8(7), 1997.

[72] M. G. Luby, J. S. Naor, and A. Orda. Tight bounds for dynamic storage allocation. *SIAM Journal on Discrete Mathematics*, 9(1):155–166, 1996.

[73] J. Mache and V. Lo. Dispersal metrics for non-contiguous processor allocation. Technical Report CIS-TR-96-13, University of Oregon, 1996.

[74] J. Mache and V. Lo. The effects of dispersal on message-passing contention in processor allocation strategies. In *Proc. Third Joint Conference on Information Sciences, Sessions on Parallel and Distributed Processing*, volume 3, pages 223–226, 1997.

[75] J. Mache, V. Lo, and K. Windisch. Minimizing message-passing contention in fragmentation-free processor allocation. In *Proc. 10th International Conf. Parallel and Distributed Computing Systems*, pages 120–124, 1997.

[76] J. E. Marsden and T. S. Ratiu. *Introduction to Mechanics and Symmetry.* Springer, New York, 1999.

[77] Y. Matias and A. Shamir. A video scrambling technique based on space filling curves. In *Proc. Advances in Cryptology (CRYPTO)*, volume 293 of *Lecture Notes in Computer Science*, pages 398–417. Springer-Verlag, 1987.

[78] A. Mercier. *Variational Principles of Physics.* Dover, New York, 1963.

[79] B. Moon, H. V. Jagadish, C. Faloutsos, and J. Saltz. Analysis of the clustering properties of Hilbert space-filling curve. *IEEE Trans. on Knowledge and Data Engineering*, 13(1):124–141, 2001.

[80] S. Moore and L. Ni. The effects of network contention on processor allocation strategies. Technical Report MSU-CPS-ACS-106, Michigan State University, 1995.

[81] S. Moore and L. Ni. The effects of network contention on processor allocation strategies. In *Proc. 10th International Parallel Processing Symposium*, pages 268–274, 1996.

[82] NASA. The portable batch system. `http://www.nas.nasa.gov/Software/PBS/`.

[83] R. Niedermeier, K. Reinhardt, and P. Sanders. Towards optimal locality in mesh-indexings. In *Proc. 11th Intl Symp on Fund. Computation Theory*, volume 1279 of *LNCS*, pages 364–375, 1997.

[84] O. D. Parekh. private communication, June 2003.

[85] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Math. Annalen*, pages 157–160, 1890.

[86] J. R. Pilkington and S. B. Baden. Dynamic partitioning of non-uniform structured workloads with spacefilling curves. *IEEE Transactions on Parallel and Distributed Systems*, 7(3):288–300, 1996.

[87] H. Prokop. Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

[88] N. Rahman, R. Cole, and R. Raman. Optimized predecessor data structures for internal memory. In *Proc. 5th Workshop on Algorithms Engineering (WAE)*, 2001.

[89] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42:299–310, 1994.

[90] J. M. Robson. An estimate of the store size necessary for dynamic storage allocation. *Journal of the ACM*, 18(3):416–423, 1971.

[91] J. M. Robson. Bounds for some functions concerning dynamic storage allocation. *Journal of the ACM*, 21(3):491–499, 1974.

[92] R. Rubin. Node allocation algorithms for parallel systems. unpublished manuscript, 2001.

[93] G. M. Sabin, P. Sadayappan, and V. J. Leung. Fairness in job scheduling on Cplant. unpublished manuscript, 2003.

[94] H. Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.

[95] S. Sahni and T. Gonzalez. P-complete approximation problems. *JACM*, 23(3):555–565, 1976.

[96] Sandia National Laboratories. The Computational Plant Project. `http://www.cs.sandia.gov/cplant`.

[97] S. Seiden and R. van Stee. New bounds for multi-dimensional packing. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 486–495, 2002.

[98] J. Sgall. On-line scheduling. In A. Fiat and G. Woeginger, editors, *On-Line Algorithms—The State of the Art*, Lecture Notes in Computer Science, chapter 9. Springer-Verlag, 1998.

[99] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[100] L. Stockmeyer. Personal communication, 2001.

[101] V. Subramani, R. Kettimuthu, S. Srinivasan, J. Johnson, and P. Sadayappan. Selective buddy allocation for scheduling parallel jobs on clusters. In *Proc. 4th IEEE International Conference on Cluster Computing*, 2002.

[102] K. S. Thyagarajan and S. Chatterjee. Fractal scanning for image compression. In *Conference Record of the 25th Asilomar Conference on Signals, Systems and Computers*, pages 467–471, 1992.

[103] P. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom*, 4:101–115, 1989.

[104] C. T. Vaughan. private communication, August 2003.

[105] A. Yao. Towards a unified measure of complexity. In *Proc. 18th Symp. Found. Computer Science*, pages 222–227, 1977.

[106] Y. Zhu. Efficient processor allocation strategies for mesh-connected parallel computers. *J. Parallel and Distributed Computing*, 16:328–337, 1992.

UNLIMITED RELEASE
INITIAL DISTRIBUTION:

10 State University of New York
Dept. of Computer Science
Attn: M. A. Bender
Stony Brook, NY 11794-4400

1 University of Illinois
Dept. of Computer Science
Attn: D. P. Bunde
Urbana, IL 61801

1	MS	0310	R. W. Leland, 9220
1		0312	W. J. Camp, 9200
1		0316	J. B. Aidun, 9235
1		0316	S. S. Dosanjh, 9233
1		0316	M. D. Rintoul, 9212
1		0318	P. D. Heermann, 9227
1		0318	J. E. Nelson, 9216
1		0318	P. Yarrington, 9230
1		0323	D. L. Chavez, 1011
1		0819	R. M. Summers, 9231
1		0820	P. F. Chavez, 9232
1		0847	T. D. Blacker, 9226
1		0847	S. A. Mitchell, 9211
1		1110	R. D. Carr, 9215
1		1110	N. W. Dauchy, 9224
1		1110	J. M. DeLaurentis, 9214
1		1110	D. W. Doerfler, 9224
1		1110	S. P. Goudy, 9224
1		1110	J. R. Johnston, 9223
10		1110	V. J. Leung, 9215
10		1110	C. A. Phillips, 9215
1		1110	N. D. Pundit, 9223
1		1110	D. E. Womble, 9214
1		1111	B. A. Hendrickson, 9215
1		1111	C. T. Vaughan, 9224
1		9018	Central Technical Files, 8945-1
2		0899	Technical Library, 9616

Intentionally Left Blank