Generalized Delaunay Mesh Refinement: From Scalar to Parallel

Andrey N. Chernikov and Nikos P. Chrisochoides

Department of Computer Science
College of William and Mary
Williamsburg, VA 23185
{ancher,nikos}@cs.wm.edu

Summary. The contribution of the current paper is three-fold. First, we generalize the existing sequential point placement strategies for guaranteed quality Delaunay refinement: instead of a specific position for a new point, we derive a selection disk inside the circumdisk of a poor quality triangle. We prove that any point placement algorithm that inserts a point inside the selection disk of a poor quality triangle will terminate and produce a size-optimal mesh. Second, we extend our theoretical foundation for the parallel Delaunay refinement. Our new parallel algorithm can be used in conjunction with any sequential point placement strategy that chooses a point within the selection disk. Third, we implemented our algorithm in C++ for shared memory architectures and present the experimental results. Our data show that even on workstations with a few cores, which are now in common use, our implementation is significantly faster the best sequential counterpart.

1 Introduction

In this paper we address theoretical and practical aspects for the development of both scalar and parallel Delaunay mesh generation algorithm and software that satisfy the following requirements:

1. allow to construct well-shaped elements with bounded minimal angle;
2. produce graded meshes, i.e., meshes with element size specified by a user-defined function;
3. offer proofs of termination and size optimality;
4. allow to use custom point placement strategies (e.g., circumcenter, off-center, etc.);
5. replace the solution of a difficult domain decomposition problem with an easier data distribution approach without relying on the speculative execution model [10, 19];
6. offer performance improvement over the best available sequential software, even on workstations with just a few hardware cores.

We describe our solution which satisfies all of these requirements. Although the extension of the method to three dimensions is still is progress, we present a complete practical parallel two-dimensional guaranteed quality graded mesh generator. In such applications as the direct numerical simulations of turbulence in cylinder flows with very large Reynolds numbers [12] and coastal ocean modeling for predicting storm surge and beach erosion in real-time [25], three-dimensional simulations are conducted using two-dimensional meshes. In both cases, 2D mesh generation is taking place in the xy-plane and it is replicated in the z-direction in the case of cylinder flows or using bathemetric contours in the case of coastal ocean modeling applications.

The field of sequential guaranteed quality Delaunay refinement has been extensively studied, see [8, 13, 16, 20, 23] and the references therein. However, new ideas and improvements keep being introduced. One of the basic questions being studied is where to insert additional (so-called Steiner) points into an existing mesh in order to improve the quality of the elements. Ruppert's [20] and early Chew's [8] algorithms use circumcenters of poor quality triangles. Later, Chew [9] suggested to use randomized insertion of *near-circumcenter* points for three-dimensional Delaunay refinement, with the goal of avoiding slivers. Recently, Üngör [24] proposed to insert specially chosen points which he calls *off-centers*; this method allows to produce smaller meshes in practice and it was implemented in the popular sequential mesh generation software the Triangle [22]. We expect that other optimization techniques can be used to select positions for new points. Indeed, in Subsection 2.2 we give an example of a point placement strategy that in some cases allows to achieve even smaller meshes than the off-center method, albeit at significant computation cost. Since one would not like to redesign the parallel algorithm and software to accommodate each of the point placement techniques, in this paper we generalize the sequential Delaunay refinement approaches and develop a framework which allows to use custom point selection strategies. In particular, we derive a *selection disk* for the position of a new point with respect to a poor quality triangle and prove that any point placement technique with the only restriction that it selects a point inside the selection disk will terminate and produce a size-optimal guaranteed quality mesh. While the use of Chew's [9] *picking-sphere* is restricted to produce only meshes with constant density, the use of our selection disk allows to obtain graded size-optimal meshes.

The domain decomposition problem for parallel mesh generation is formulated as follows [11, 14, 15]. Given a domain $\Omega \subset \mathbb{R}^n$, construct the separators $S_{ij} \subset \mathbb{R}^{n-1}$, $S_{ij} \subset \Omega$, such that Ω is decomposed into *subdomains* Ω_i:

$$\Omega = \bigcup_{i=1}^{N} \Omega_i, \quad \Omega_i \cap \Omega_j = S_{ij}, \quad i, j = 1, \ldots, N, \quad i \neq j,$$

while the separators do not create very small angles and other features that will force the degradation of the mesh quality. Linardakis and Chrisochoides [14, 15] described a Medial Axis Domain Decomposition Method for two-dimensional geometries. However, the solution is based on the Medial Axis Transform which is very difficult and expensive to construct for three-dimensional geometries. Another approach is to partition and refine the mesh simultaneously [10, 19], such that when a conflict is detected between concurrently inserted points, some of the point insertions are canceled, which leads to high computation and communication costs.

In [4–7] we showed that the domain decomposition problem for the parallel Delaunay refinement can be replaced with an easier data distribution problem. We proved that an auxiliary spatial data structure, like a uniform 2D (or 3D) lattice and a quadtree (or an octree), can be constructed in such a way that the points introduced in certain regions of Ω, that correspond to separated regions from the above data structure, do not have any dependences and can be inserted concurrently.

In [5, 7] we presented the theory and the implementation for the parallel construction of guaranteed quality uniform two-dimensional meshes which use a uniform lattice as an auxiliary spatial data structure. In [4, 6] we presented the theoretical foundation for the construction of non-uniform (graded) meshes for the circumcenter point insertion method [8, 20, 23]. In this paper, we present the algorithm for the generalized point insertion method, describe our implementation and the experimental results using the off-center point insertion strategy [24].

Ruppert's sequential Delaunay refinement algorithm has quadratic worst-case running time, even though in most practical cases the time is linear with respect to the output size [20, 23]. Recently, Miller [16] proposed a Delaunay refinement algorithm which runs in optimal $\mathcal{O}(n \log n + m)$ time, where n is the size of the input, and m is the size of the output. He achieved this improvement by introducing a priority queue, where the skinny triangles are ordered by their diameter (equivalently, circumradius), and the triangles with the largest diameter are refined first. As it can be seen further in the paper, our parallel algorithm also gives priority to triangles with large circumradii, which allows to eliminate quadratic running time for pathological input geometries.

Cheng et al. [3] developed an algorithm to remove sliver tetrahedra from an existing Delaunay mesh. The algorithm pumps the weights of the vertices and flips the edges to obtain a new triangulation of the same point set. The maximum weight that can be assigned to a point is bounded by a function of the distance to the nearest point. This relates to the choice of radius for our selection disk which depends on the length of the shortest edge of a triangle.

In Section 2 we introduce the background for the sequential Delaunay refinement, define the selection disk for the insertion of Steiner points, and present the proofs of termination and size optimality to show that a Delaunay refinement algorithm which chooses points inside the selection disks of skinny triangles terminates and produces size optimal meshes. Then, in Section 3 we describe our parallel implementation and experimental results. Section 4 concludes the paper.

2 Point Insertion for Sequential Delaunay Refinement

2.1 Delaunay Refinement Background

Let the mesh $\mathcal{M} = (V, T, S)$ consist of a set $V = \{p_i\}$ of vertices, a set $T = \{t_i = \triangle(p_u p_v p_w) \mid p_u, p_v, p_w \in V\}$ of triangles, and a set $S = \{s_i = p_u p_v \mid p_u, p_v \in V\}$ of constrained segments. We will denote an edge of a triangle as $e(p_i p_j)$. The input to a planar triangular mesh generation algorithm includes a description of *domain* $\Omega \subset \mathbb{R}^2$, which is permitted to contain holes or have more than one connected component. We will use a *Planar Straight Line Graph* (PSLG) [22] to delimit Ω from the rest of the plane. Each segment in the PSLG is considered *constrained* and must appear (possibly as a union of smaller subsegments) in the final mesh. The vertices of the PSLG are a subset of the final set of vertices in the mesh.

There are two commonly used parameters that control the quality of mesh elements: an upper bound on the circumradius-to-shortest edge ratio (which is equivalent to a lower bound on a minimal angle [17]) and an upper bound on the element area. We will denote the circumradius-to-shortest edge ratio of triangle t as $\rho(t)$ and the area of triangle t as $\Delta(t)$. The former upper bound is usually fixed and given by a constant value $\bar{\rho}$, while the latter can vary and be controlled by some user-defined grading function $\bar{\Delta}(\cdot)$, which can be defined either over the set of triangles or over Ω, depending on the implementation.

Let us call the open disk corresponding to a triangle's circumscribed circle its *circumdisk*. We will use symbols $\bigcirc(t)$ and $r(t)$ to represent the circumdisk and the circumradius of triangle t, respectively. A mesh is said to satisfy the *Delaunay property* if the circumdisk of every triangle does not contain any of the mesh vertices [13, 23].

Delaunay mesh generation algorithms start with constructing an initial mesh, which conforms to the input PSLG, and then refine this mesh until the element quality constraints are met. In this paper, we focus on parallelizing the Delaunay refinement stage, which is usually the most memory- and computation-expensive. The general idea of Delaunay refinement is to insert additional (Steiner) points inside the circumdisks of poor quality triangles, which causes these triangles to be destroyed, until they are gradually eliminated and replaced by better quality triangles.

We will extensively use the notion of *cavity* [13] which is the set of triangles in the mesh whose circumdisks include a given point p_i. We will denote $\mathcal{C}(p_i)$ to be the cavity of p_i and $\partial\mathcal{C}(p_i)$ to be the set of edges which belong to only one triangle in $\mathcal{C}(p_i)$, i.e., external edges. For our analysis, we will use the Bowyer-Watson (B-W) point insertion algorithm [2, 26]:

$$\begin{aligned} V' &\leftarrow V \cup \{p_i\}, \\ T' &\leftarrow T \setminus \mathcal{C}(p_i) \cup \{\triangle(p_i p_j p_k) \mid e(p_j p_k) \in \partial\mathcal{C}(p_i)\}, \end{aligned} \qquad (1)$$

where $\mathcal{M} = (V, T, S)$ and $\mathcal{M}' = (V', T', S')$ represent the mesh before and after the insertion of p_i, respectively.

Sequential Delaunay algorithms treat *constrained* segments differently from triangle edges [20, 23]. A vertex p is said to *encroach upon* a segment s, if it lies within the open diametral disk of s [20]. When a new point is about to be inserted and it happens to encroach upon a constrained segment s, another point is inserted in the middle of s instead [20], and a cavity of the segment's midpoint is constructed and triangulated according to (1).

The proofs of termination and size optimality of Delaunay refinement algorithms [20, 23] explore the relationships between the insertion radius of a point and that of its parent. The *insertion radius* $R(p)$ of point p is defined as the length of the shortest edge connected to p immediately after p is inserted into the mesh [23]. The *parent* $\hat{p}$ of point p is the vertex which is "responsible" for the insertion of p [23]. In particular, if p is inserted inside the circumdisk of a poor quality triangle, $\hat{p}$ is the most recently inserted vertex of the shortest edge of that triangle. If p is a midpoint of an encroached segment, $\hat{p}$ is the point (possibly rejected for insertion) that encroaches upon that segment. If p is an input vertex, it has no parent. In addition, the proofs require that $\bar{\rho} \geq \sqrt{2}$.

The local feature size function $\mathrm{lfs} : \mathbb{R}^2 \rightarrow \mathbb{R}$ for a given point p is equal to the radius of the smallest disk centered at p that intersects two non-incident vertices or segments of PSLG $\mathcal{X}$ [20]. $\mathrm{lfs}(p)$ satisfies the Lipchitz condition:

Lemma 1 (Lemma 1 in Ruppert [20], Lemma 2 in Shewchuk [23]). *Given any PSLG $\mathcal{X}$ and any two points p_i and p_j in the plane, the following inequality holds:*

$$\mathrm{lfs}(p_i) \leq \mathrm{lfs}(p_j) + \|p_i - p_j\| \tag{2}$$

2.2 Delaunay Refinement Using Selection Disks

Traditionally, Steiner points are selected in the circumcenters of poor quality triangles [8, 20, 23]. However, Chew [9] chooses Steiner points randomly inside a *picking sphere* to avoid slivers. His goal is to construct a mesh with constant density; therefore he proves the termination, but not the size-optimality of the mesh. Ruppert [20] and Shewchuk [23] proved that if $\bar{\rho} \geq \sqrt{2}$, then Delaunay refinement with circumcenters terminates and, furthermore, produces size-optimal meshes. In this context, size-optimality means that the number of triangles in the resulting mesh will be within a constant multiple of the smallest possible number of triangles satisfying the input constraints.

Recently, Üngör [24] introduced a new type of Steiner points called *off-centers*. The idea is based on the observation that sometimes the triangles created as a result of inserting circumcenters of skinny triangles are also skinny and require further refinement. It combines the advantages of advancing front methods, which can produce very well-shaped elements in practice, and Delaunay methods, which offer theoretical guarantees. Üngör showed that the use of off-centers allows to significantly decrease the size of the final mesh in practice. Consider Figure 1(left). Suppose $\triangle(p_k p_l p_m)$ is skinny: $\rho(\triangle(p_k p_l p_m)) > \bar{\rho}$. If

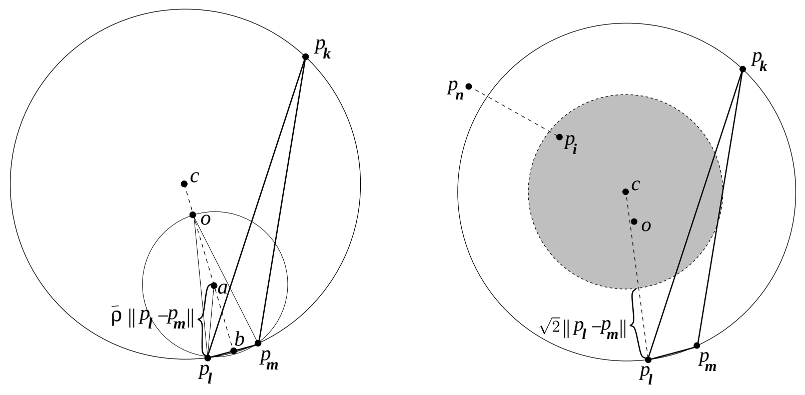

Fig. 1. **(Left)** Delaunay refinement with off-centers [24]. **(Right)** The selection disk (shaded) for the insertion of a Steiner point.

we insert its circumcenter c, the new triangle $\triangle(cp_lp_m)$ may also be skinny. In this case, instead of inserting c, Üngör suggests to insert the *off-center* o chosen on the perpendicular bisector of the shortest edge $e(p_lp_m)$ in such a way that the new triangle $\triangle(op_lp_m)$ will have circumradius-to-shortest edge ratio equal to exactly $\bar{\rho}$. While eliminating additional point insertions, this strategy creates triangles with the longest possible edges, such that one can prove termination of the algorithm and size-optimality of the result.

However, circumcenters and off-centers are not the only possible positions for inserting the Steiner points, either sequentially or in parallel, such that one can prove termination and size-optimality.

Definition 1. *If t is a poor quality triangle with circumcenter c, shortest edge length l, circumradius r, and circumradius-to-shortest edge ratio $\rho = r/l > \bar{\rho} \geq \sqrt{2}$, then the* selection disk *for the insertion of a Steiner point that would eliminate t is the open disk with center c and radius $r - \sqrt{2}l$.*

For example, in Figure 1(right) $e(p_lp_m)$ is the shortest edge of a skinny triangle $\triangle(p_kp_lp_m)$ and c is its circumcenter. The selection disk is the shaded disk with center c and radius $r(\triangle(p_kp_lp_m)) - \sqrt{2}\|p_l - p_m\|$.

Below we prove that any point inside the selection disk of a triangle can be chosen for the elimination of the triangle, and that the generalized Delaunay refinement algorithm which chooses Steiner points inside the selection disks terminates and produces size-optimal meshes.

Remark 1. The radius of Chew's picking sphere is fixed and is equal to one half of the length of the shortest possible edge in the final mesh [9]. We generalize the idea of his picking sphere to the selection disk, such that the radius of the selection disk varies among the triangles and depends on the length of the shortest edge of each particular triangle.

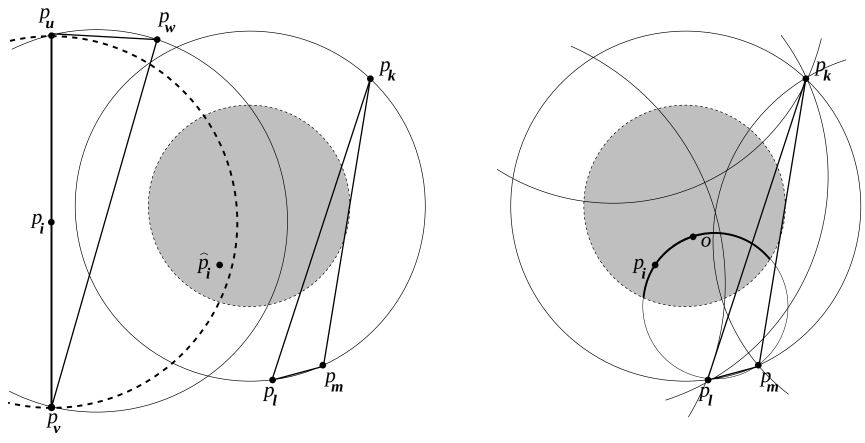

Fig. 2. **(Left)** $\hat{p}_i$ is a Steiner point within a selection disk of a poor quality triangle which encroaches upon a constrained segment $e\left(p_u p_v\right)$. **(Right)** An optimization-based method for the selection of a Steiner point within a selection disk of a poor quality triangle.

Remark 2. Üngör's off-center always lies inside the selection disk.

Remark 3. As $\rho\left(\triangle\left(p_k p_l p_m\right)\right)$ approaches $\sqrt{2}$, the selection disk shrinks to the circumcenter c of the triangle. If, furthermore, $\rho\left(\triangle\left(p_k p_l p_m\right)\right) \leq \sqrt{2}$, the selection disk vanishes, which coincides with the fact that the triangle $\triangle\left(p_k p_l p_m\right)$ cannot be considered skinny.

The proofs of termination and size-optimality of Delaunay refinement with circumcenters in [8, 20, 23] rely on the assumption that the insertion radius of the Steiner point is equal to the circumradius of the poor quality triangle. This assumption holds when the Steiner point is chosen in the circumcenter of a triangle, since by Delaunay property the circumdisk of the triangle is empty, and, hence, there is no vertex closer to the circumcenter than the vertices of this triangle. However, the above assumption does not hold if we pick an arbitrary point p_i within the selection disk, see Figure 1(right). Therefore, we need new proofs in the new context when Steiner points can be inserted anywhere within the selection disks of poor quality triangles.

Proof of Termination

Lemma 2. *If p_i is a vertex of the mesh produced by a Delaunay refinement algorithm which chooses points within selection disks of triangles with circumradius-to-shortest-edge ratios greater than $\bar{\rho} \geq \sqrt{2}$, then the following inequality holds:*

$$R\left(p_i\right) \geq C \cdot R\left(\hat{p}_i\right), \tag{3}$$

where we distinguish among the following cases:

(i) $C = \sqrt{2}$ *if* p_i *is a Steiner point chosen within the selection disk of a skinny triangle;*

Otherwise, let p_i *be the midpoint of subsegment* s*. Then*

(ii) $C = \frac{1}{\sqrt{2}}$ *if* $\hat{p}_i$ *is a Steiner point which encroaches upon* s*, chosen within the selection disk of a skinny triangle;*
(iii) $C = \frac{1}{2\cos\alpha}$ *if* p_i *and* $\hat{p}_i$ *lie on incident subsegments separated by an angle of* α *(with* $\hat{p}_i$ *encroaching upon* s*), where* $45° \leq \alpha \leq 90°$*;*
(iv) $C = \sin\alpha$ *if* p_i *and* $\hat{p}_i$ *lie on incident segments separated by an angle of* $\alpha \leq 45°$*.*

If p_i *is an input vertex, then*

$$R(p_i) \geq \operatorname{lfs}(p_i)\,. \tag{4}$$

Proof. We need to present new proofs only for cases (i) and (ii), since the proofs for all other cases are independent of the choice of the point within the selection disk and are given in [23].

Case (i) By the definition of a parent vertex, $\hat{p}_i$ is the most recently inserted endpoint of the shortest edge of the triangle; without loss of generality let $\hat{p}_i = p_l$ and $e(p_l p_m)$ be the shortest edge of the skinny triangle $\triangle(p_k p_l p_m)$, see Figure 1(right). If $e(p_l p_m)$ was the shortest edge among the edges incident upon p_l at the time p_l was inserted into the mesh, then $\|p_l - p_m\| = R(p_l)$ by the definition of the insertion radius; otherwise, $\|p_l - p_m\| \geq R(p_l)$. In either case,

$$\|p_l - p_m\| \geq R(p_l)\,. \tag{5}$$

Now we can derive the relation between the insertion radius of point p_i and the insertion radius of its parent $\hat{p}_i = p_l$:

$$\begin{aligned} R(p_i) &> \sqrt{2}\|p_l - p_m\| && \text{(by the construction of the selection disk)} \\ &\geq \sqrt{2}R(p_m)\,. && \text{(from (5))} \end{aligned}$$

Hence, $R(p_i) > \sqrt{2}R(\hat{p}_i)$; choose $C = \sqrt{2}$.

Case (ii) Let $\hat{p}_i$ be inside the selection disk of a skinny triangle $\triangle(p_k p_l p_m)$, such that $\hat{p}_i$ encroaches upon $e(p_u p_v)$, see Figure 2(left). Since the edge $e(p_u p_v)$ is part of the mesh, there must exist some vertex p_w such that p_u, p_v, and p_w form a triangle. Because p_w is outside of the diametral circle of $e(p_u p_v)$, the circumdisk $\bigcirc(\triangle(p_u p_v p_w))$ has to include point $\hat{p}_i$. Therefore, if $\hat{p}_i$ were inserted into the mesh, $\triangle(p_u p_v p_w)$ would be part of the cavity $\mathcal{C}(\hat{p}_i)$ and the edges connecting $\hat{p}_i$ with p_u and p_v would be created. Therefore,

$$R(\hat{p}_i) \leq \min(\|\hat{p}_i - p_u\|, \|\hat{p}_i - p_v\|) < \sqrt{2}\frac{\|p_u - p_v\|}{2} = \sqrt{2}R(p_i)\,;$$

choose $C = \frac{1}{\sqrt{2}}$. □

Theorem 1 (Theorem 4 in Shewchuk [23]). *Let* $\mathrm{lfs}_{\min}$ *be the shortest distance between two non-incident entities (vertices or segments) of the input PSLG. Suppose that any two incident segments are separated by an angle of at least* $60°$*, and a triangle is considered to be skinny if its circumradius-to-shortest edge ratio is larger than* $\bar{\rho}$*, where* $\bar{\rho} \geq \sqrt{2}$*. Ruppert's algorithm will terminate with no triangulation edge shorter than* $\mathrm{lfs}_{\min}$*.*

The proof of this theorem in [23] is based on the result of Lemma 3 in [23], which establishes the relations (3) and (4) in the context of circumcenter point insertion. Otherwise, the proof is independent of the specific position of inserted points. Since we proved (3) and (4) in the context of inserting arbitrary points within selection disks, this theorem also holds in this context.

Proof of Good Grading and Size Optimality

The quantity $D(p)$ is defined as the ratio of $\mathrm{lfs}(p)$ over $R(p)$ [23]:

$$D(p) = \frac{\mathrm{lfs}(p)}{R(p)}. \tag{6}$$

It reflects the density of vertices near p at the time p is inserted, weighted by the local feature size.

Lemma 3. *If* p_i *is a vertex of the mesh produced by a Delaunay refinement algorithm which chooses points within selection disks of skinny triangles and* C *is the constant specified by Lemma 2, then the following inequality holds:*

$$D(p_i) \leq 1 + \frac{D(\hat{p}_i)}{C}. \tag{7}$$

Proof. If p_i is chosen within the selection disk of a skinny triangle, then $R(p_i) > \sqrt{2}\|p_i - \hat{p}_i\|$ by construction. If p_i is a segment midpoint and $\hat{p}_i$ is a rejected encroaching Steiner point within a selection disk, $R(p_i) > \|p_i - \hat{p}_i\|$ because $\hat{p}_i$ is inside the diametral circle of the segment. If p_i is a segment midpoint and $\hat{p}_i$ is an encroaching vertex which lies on another segment, then by the definition of the insertion radius $R(p_i) = \|p_i - \hat{p}_i\|$ by the definition of the insertion radius. In all cases,

$$R(p_i) \geq \|p_i - \hat{p}_i\|. \tag{8}$$

Then

$$\begin{aligned}
\mathrm{lfs}(p_i) &\leq \mathrm{lfs}(\hat{p}_i) + \|p_i - \hat{p}_i\| && \text{(from Lemma 1)}\\
&\leq \mathrm{lfs}(\hat{p}_i) + R(p_i) && \text{(from (8))}\\
&= D(\hat{p}_i)\,R(\hat{p}_i) + R(p_i) && \text{(from (6))}\\
&\leq D(\hat{p}_i)\,\tfrac{R(p_i)}{C} + R(p_i) && \text{(from Lemma 2)}
\end{aligned}$$

The result follows from dividing both sides by $R(p_i)$. □

Lemma 4 (Extension of Lemma 7 in Shewchuk [23] and Lemma 2 in Ruppert [20]). *Suppose that $\bar{\rho} > \sqrt{2}$ and the smallest angle in the input PSLG is strictly greater than $60°$. There exist fixed constants C_T and C_S such that, for any vertex p_i inserted (or considered for insertion and rejected) within the selection disk of a skinny triangle, $D(p_i) \le C_T$, and for any vertex p_i inserted at the midpoint of an encroached subsegment, $D(p_i) \le C_S$. Hence, the insertion radius of a vertex has a lower bound proportional to its local feature size.*

The proof of this Lemma in [23] relies only on Lemmata 2 and 3 here which have been proven to hold for the Steiner points chosen within selection disks of skinny triangles. Hence, the Lemma holds in this context, too.

Theorem 2 (Theorem 8 in Shewchuk [23], Theorem 1 in Ruppert [20]). *For any vertex p_i of the output mesh, the distance to its nearest neighbor is at least $\frac{\mathrm{lfs}(p_i)}{C_S+1}$.*

The proof in [23] relies only on Lemmata 1 and 4 here and, therefore, holds for the arbitrary points chosen within selection disks of skinny triangles.

Theorem 2 is the precondition of the following theorem:

Theorem 3 (Theorem 10 in Shewchuk [23], Theorem 14 in Mitchell [18], Theorem 3 in Ruppert [20]). *Let $\mathrm{lfs}_{\mathcal{M}}(p_i)$ be the local feature size at p_i with respect to a mesh $\mathcal{M}$ (treating $\mathcal{M}$ as a PSLG), whereas $\mathrm{lfs}(p_i)$ remains the local feature size at p_i with respect to the input PSLG. Suppose a mesh $\mathcal{M}$ with smallest angle θ has the property that there is some constant $k_1 \ge 1$, such that for every point p_i, $k_1 \mathrm{lfs}_{\mathcal{M}}(p_i) \ge \mathrm{lfs}(p_i)$. Then the cardinality of $\mathcal{M}$ is less than k_2 times the cardinality of any other mesh of the input PSLG with smallest angle θ, where $k_2 = \mathcal{O}\left(k_1^2/\theta\right)$.*

An Example of a Point Selection Strategy

Let us consider Figure 2(right). We can see that the off-center o of the skinny triangle $\triangle(p_k p_l p_m)$ is not the only location for a Steiner point p_i that will lead to the creation of the new triangle incident to the edge $e(p_l p_m)$ with circumradius-to-shortest edge ratio equal to exactly $\bar{\rho}$. The arc shown in bold in the Figure is the intersection of the circle passing through p_l, p_m, and o with the selection disk of $\triangle(p_k p_l p_m)$. Let us denote this arc as $\varGamma$. The thin arcs show parts of the circumcircles of other triangles in the mesh. We can observe that the cavity $\mathcal{C}(p_i)$ will vary depending on the location of p_i, according to the set of triangle circumdisks that include p_i. Let us also represent the penalty for deleting triangle t as $P(t)$:

$$P(t) = \begin{cases} -1, & \text{if } (\rho(t) > \bar{\rho}) \vee (\varDelta(t) > \bar{\varDelta}), \\ 1, & \text{otherwise.} \end{cases}$$

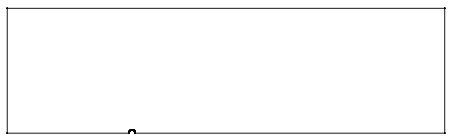
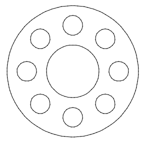

Fig. 3. **(Left)** Un upper part of a model of cylinder flow. **(Right)** Pipe cross-section model.

Table 1. The comparison of the number of triangles generated with the use of three different point insertion strategies, for different models and minimal angle bounds. No area bound is used.

Point position	$\theta = 10°$		$\theta = 20°$		$\theta = 30°$	
	flow	pipe	flow	pipe	flow	pipe
Circumcenter	2173	3033	3153	4651	8758	10655
Off-center	1906	2941	2942	4411	6175	8585
Our optimization-based method	1805	2841	2932	4359	6319	8581

Then our objective is to minimize the profit function associated with the insertion of point p_i as the sum of the penalties for deleting all triangles in the cavity $\mathcal{C}(p_i)$:

$$\min_{p_i \in \Gamma} F(p_i), \quad F(p_i) = \sum_{t \in \mathcal{C}(p_i)} P(t)$$

In other words, we try to minimize the number of deleted good quality triangles and at the same time to maximize the number of deleted poor quality triangles. The results of our experiments with the cylinder flow (Fig. 3(left)) and the pipe cross-section (Fig. 3(right)) models using Triangle version 1.6 [22] are summarized in Tables 1 and 2. We do not list the running times since our experimental implementation is built on top of the Triangle and does not take advantage of its intermediate calculations as do the circumcenter and off-center insertion methods. From Table 1 we can see that our optimization-based method tends to produce up to 20% fewer triangles than the circumcenter method and up to 5% fewer triangles than the off-center method for small values of the minimal angle bound and no area bound, and the improvement diminishes as the angle bounds increase. Table 2 lists the results of the similar experiments, with an additional area bound constraint. We observe that the introduction of an area bound effectively voids the difference among the presented point insertion strategies.

3 Generalized Parallel Delaunay Refinement

In [6] we described the construction of a quadtree which overlays the mesh. If a part of the mesh associated with a leaf *Leaf* of the quadtree is scheduled

Table 2. The comparison of the number of triangles generated with the use of three different point insertion strategies. For the cylinder flow model, the area bound is set to $\bar{\Delta} = 0.01$, and for the pipe cross-section model $\bar{\Delta} = 1.0$.

Point position	$\theta = 10°$		$\theta = 20°$		$\theta = 30°$	
	flow	pipe	flow	pipe	flow	pipe
Circumcenter	219914	290063	220509	289511	228957	294272
Off-center	219517	290057	220479	289331	226894	295644
Our optimization-based method	219470	289505	220281	289396	226585	294694

```
DelaunayRefinement($\mathcal{X}$, $g$, $\bar{\Delta}(\cdot)$, $\bar{\rho}$, $f(\cdot)$, $\mathcal{M}$, $Leaf$)
Input: $\mathcal{X}$ is a PSLG which defines $\Omega$
       $g$ is the granularity
       $\bar{\Delta}(\cdot)$ is the triangle area grading function
       $\bar{\rho}$ is the upper bound on triangle circumradius-to-shortest edge ratio
       $f(\cdot)$ is a deterministic function which returns a specific position
         within triangle's selection disk
       $\mathcal{M}$ is the current Delaunay mesh
       $Leaf$ is the leaf scheduled for refinement
Output: Locally refined Delaunay mesh $\mathcal{M}$
        Locally refined quadtree node $Leaf$
1  $i_{min} \leftarrow \min_{PoorTriangles_i(Leaf) \neq \emptyset} i$
2  $i_{max} \leftarrow i_{min} + g$
3  for $j = i_{min}, \ldots, i_{max}$
4    while $PoorTriangles_j(Leaf) \neq \emptyset$
5      Let $t \in PoorTriangles_j(Leaf)$
6      $p \leftarrow f(t)$
7      Insert $p$ into $\mathcal{M}$
8      for $L \in \{Leaf\} \cup$ BUF $(Leaf)$
9        Update $PoorTriangles(L)$ and $Counter(L)$
10     endfor
11   endwhile
12 endfor
13 Split $Leaf$ recursively while (9) holds
14 return $\mathcal{M}$, $Leaf$
```

Fig. 4. The algorithm executed by each of the refinement threads.

for refinement by a thread, no other thread can refine the parts of the mesh associated with the buffer zone BUF $(Leaf)$ of this leaf. For each leaf of the quadtree the following relation is maintained:

$$\forall t \in \mathcal{M} \ : \ \bigcirc(t) \cap Leaf \neq \emptyset \implies r(t) < \frac{1}{4} \ell(Leaf), \qquad (9)$$

where $\ell(Leaf)$ is the length of the side of $Leaf$. See [6] for the details.

The algorithm is designed for the execution by one master thread which manages the work pool and multiple refinement threads which refine the mesh and the quadtree. Figure 4 presents the part of the algorithm executed by each of the refinement threads.

When a quadtree leaf $Leaf$ is scheduled for refinement, we remove not only the nodes from the buffer zone BUF $(Leaf)$ of $Leaf$ from the refinement queue, but also the nodes from BUF (L) for each $L \in$ BUF $(Leaf)$. Although this is not required by our theory, there are two implementation considerations for doing so, and both are related to the goal of reducing fine-grain synchronization.[1] First, each leaf has an associated data structure which stores the poor quality triangles whose circumdisks intersect this leaf, so that we can maintain the relation (9). Even though in theory the refinement of the mesh by concurrent threads is not going to cause problems when the threads work within the same quadtree leaf, in practice we would have to introduce synchronization in line 9 of the algorithm in Figure 4 to maintain this data structure. Second, for efficiency considerations, we followed the design of the triangle data element that is used in the Triangle [22]. In particular, each triangle contains pointers to neighboring triangles for easy mesh traversal. However, if two cavities share an edge and are updated by the concurrent threads, which can be done legitimately in certain cases [6], these triangle–neighbor pointers will be invalidated. For these reasons, we chose to completely separate the sets of leaves affected by the mesh refinement performed by multiple threads.

Each of the worker threads performs the refinement of the mesh and the refinement of the quadtree. The poor quality triangles whose split-points selected by a deterministic function $f(\cdot)$ are inside the square of $Leaf$ are stored in the data structure denoted here as $PoorTriangles(Leaf)$. $Leaf$ needs to be scheduled for refinement if the size of this data structure is not empty. In addition, each $Leaf$ has a counter for the triangles with various ratios of the side length of $Leaf$ to their circumradius. If we denote $\sigma(t, Leaf) = \left\lfloor \log_2 \frac{\ell(Leaf)}{r(t)} \right\rfloor$, then $Counter_i(Leaf) = |\{t \in \mathcal{M} \mid (\bigcirc(t) \cap Leaf \neq \emptyset) \wedge (\sigma(t, Leaf) = i)\}|$. When $Counter_i(Leaf) = 0$, $\forall i = 1, 2, 3$, it implies that (9) would hold for each of the children of $Leaf$, and $Leaf$ can be split. In [5] we proved that when a point is inserted into a Delaunay mesh using the B-W algorithm, the circumradii of the new triangles are not going to be larger than the circumradii of the triangles in the cavity of the point or those that are adjacent to the cavity. Therefore, new triangles that would violate (9) are not going to be created. Each leaf of the quadtree has associated with it a bucketing structure which holds poor quality triangles:

$$PoorTriangles_i(Leaf) = \{t \in \mathcal{M} \mid (f(t) \in Leaf) \wedge (\sigma(t, Leaf) = i) \wedge \\ ((\varDelta(t) > \bar{\varDelta}(t)) \vee (\rho(t) > \bar{\rho}))\}.$$

At each mesh refinement step, all triangles in $PoorTriangles_j(Leaf)$ are refined, for all $j = i_{min}, \ldots, i_{min} + granularity$, where

[1]As we have shown in [1], modern SMTs are not suitable for executing fine-grain parallelism in Delaunay mesh refinement.

$i_{min} = \min_{PoorTriangles_i(Leaf) \neq \emptyset} i$, and $granularity \geq 1$ is a parameter that controls how much computation is done during a single mesh refinement call. After a mesh refinement call returns, the feasibility of splitting $Leaf$ is evaluated, and it is recursively subdivided if necessary.

3.1 Implementation and Experimental Evaluation

We implemented the algorithm in C++ using Pthreads for thread management. The experiments were conducted on an IBM Power5 node with two dual-core processors running at 1.6 GHz and 8 GBytes of total physical memory. We compared our implementation with the fastest to our knowledge sequential Delaunay mesh generator the Triangle version 1.6 [22]. This is the latest release of the Triangle, which uses the off-center point insertion algorithm [24]. In order to make the results comparable, our GPDR implementation also uses the off-center point insertion [24]. Triangle provides a convenient facility for the generation of meshes respecting user-defined area bounds. The user can write his own `triunsuitable()` function and link it against the Triangle. This function is called to examine each new triangle and to decide whether or not it should be considered big and enqueued for refinement. We encoded our grading function into the `triunsuitable()` function, compiled it into an object file, and linked against both the Triangle and our own GPDR implementation. We ran each of the tests 10 times and used average or median timing measurements as indicated.

Figure 5(left) presents the total running times for several granularity values, as the number of compute threads increases from 1 to 4. One additional thread was used to manage the refinement queue. The mesh was constructed for the pipe cross-section model shown in Figure 3(left), using the grading function

$$\bar{\Delta}(x, y) = c \cdot (\sqrt{(x - 200)^2 + (y - 200)^2} + 1), \tag{10}$$

where $c = 10^{-4}$ and (x, y) is the centroid of a triangle. Thus a triangle is considered big if its area is greater than $\bar{\Delta}(x, y)$. In all tests we used the same $20°$ degrees angle bound. The total number of triangles produced both by the Triangle and GPDR was approximately 17 million.

We can see that the best running time was achieved using 4 compute threads with the granularity value equal to 2, and it constituted 56% of the Triangle's sequential running time. It is also interesting to see the intersection of lines corresponding to granularities 2 and 3, when the number of compute threads was increased from 3 to 4. This intersection reflects one of the basic tradeoffs in parallel computing, between granularity and concurrency: in order to increase the concurrency we have to decrease the granularity, which introduces more overheads.

Figure 5(right) shows the breakdown of the total execution time for each of the threads. The fact that the management thread is idle for 93% of the total time suggests the possibility of high scalability of the code on larger

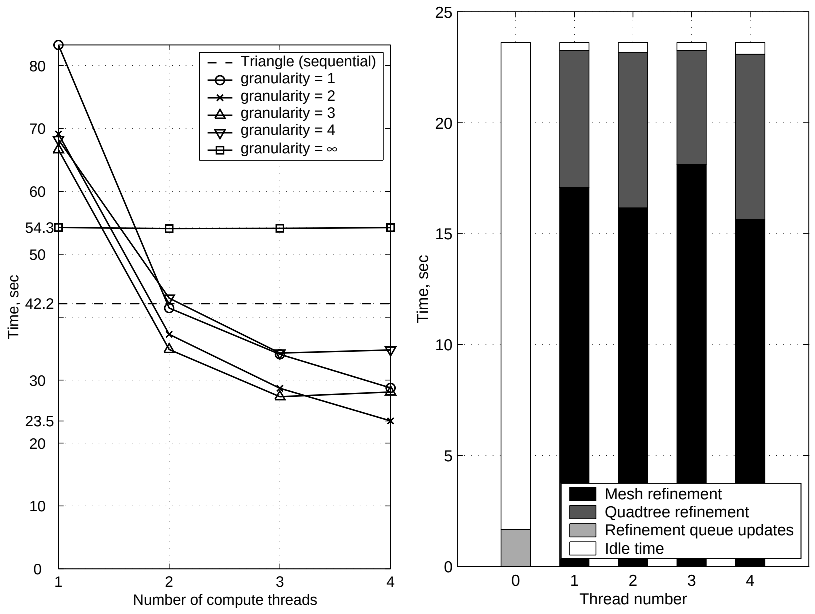

Fig. 5. **(Left)** The total running time of the GPDR code, for different granularity values, as the number of compute threads is increased from 1 to 4, compared to the Triangle [22]. Each point on the graph is the average of 10 measurements. **(Right)** The breakdown of the total GPDR execution time for each of the threads, when the number of compute threads is 4 and granularity is 2. Thread number 0 performs only the management of the refinement queue, and threads 1–4 perform mesh and quadtree refinement. The data correspond to the test with the median total time.

machines, since it can handle many more refinement threads (cores) than the current widely available machines have.

Standard system memory allocators exhibited high latency and poor scalability in our experiments, which lead us to develop a custom memory management class. At initialization, our memory pool class takes the size of the underlying object (triangle, vertex, quadtree node, or quadtree junction point) as a parameter and at runtime it allocates blocks of memory which can fit a large number of objects. When the objects are deleted, they are not deallocated but are kept for later reuse instead. Our qualitative study of the performance of the standard, the custom, and a novel generic multiprocessor allocator appears in [21].

4 Conclusions

We analyzed the existing point insertion methods for guaranteed quality Delaunay refinement and unified them into a framework which allows to develop

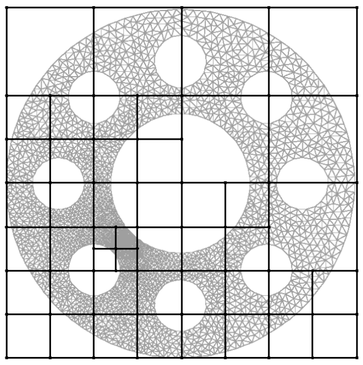

Fig. 6. An example mesh of the pipe model, with the corresponding quadtree. The grading function is given by (10) with $c = 0.3$.

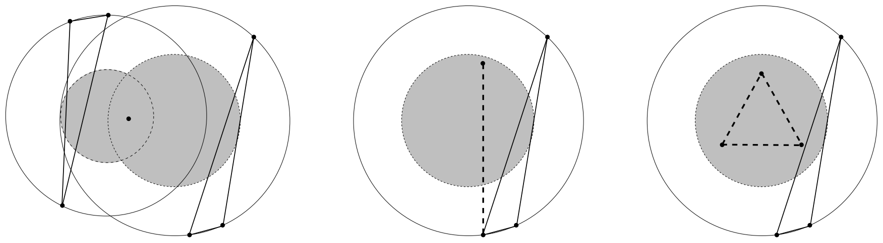

Fig. 7. Examples of the approaches for choosing Steiner points within selection disks of skinny triangles.

customized mesh optimization techniques. The goals of these techniques may include the following:

- minimizing the number of inserted points, see for example [24] and Subsection 2.2 here;
- eliminating slivers, see [9];
- splitting multiple poor quality triangles simultaneously, see Fig. 7(left).
- creating elongated edges in required directions, see Fig. 7(center);
- inserting more than one point, e.g., to create elements with specific shapes, see Fig. 7(right);
- satisfying other application-specific requirements.

In this paper, we extended our theoretical framework for parallel Delaunay refinement presented in [6] to work with custom point placement techniques. We used three different point placement methods: circumcenter, off-center and a new optimization-based method which allows to improve the size of the mesh by up to 20% and up to 5% over the first two methods, respectively.

Our current algorithm is limited to deterministic point selection; incorporating randomized point selection is left to the future research.

We presented the algorithm and the implementation of a parallel 2D graded guaranteed quality Delaunay mesh generator. The experimental results show that our code on a machine with two dual-core processors runs in 56% of the time taken by the fastest sequential code the Triangle [22]. By using a quadtree constructed in a specific way, we eliminated the need to solve the difficult domain decomposition problem. Our ongoing research includes the extension of the theory and of the implementation to three dimensions.

Acknowledgments

This work was supported (in part) by the following grants: EIA-0203974, ACI-0312980, and CNS-0521381. We thank the anonymous reviewers for helpful comments.

References

1. C. D. Antonopoulos, X. Ding, A. N. Chernikov, F. Blagojevic, D. S. Nikolopoulos, and N. P. Chrisochoides. Multigrain parallel Delaunay mesh generation: Challenges and opportunities for multithreaded architectures. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 367–376. ACM Press, 2005.
2. A. Bowyer. Computing Dirichlet tesselations. *Computer Journal*, 24:162–166, 1981.
3. S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Sliver exudation. *J. ACM*, 47(5):883–904, 2000.
4. A. N. Chernikov and N. P. Chrisochoides. Parallel guaranteed quality planar Delaunay mesh generation by concurrent point insertion. In *14th Annual Fall Workshop on Computational Geometry*, pages 55–56. MIT, Nov. 2004.
5. A. N. Chernikov and N. P. Chrisochoides. Practical and efficient point insertion scheduling method for parallel guaranteed quality Delaunay refinement. In *Proceedings of the 18th Annual International Conference on Supercomputing*, pages 48–57. ACM Press, 2004.
6. A. N. Chernikov and N. P. Chrisochoides. Parallel 2D graded guaranteed quality Delaunay mesh refinement. In *Proceedings of the 14th International Meshing Roundtable*, pages 505–517. Springer, Sept. 2005.
7. A. N. Chernikov and N. P. Chrisochoides. Parallel guaranteed quality Delaunay uniform mesh refinement. *SIAM Journal on Scientific Computing*, in print, May 2006.
8. L. P. Chew. Guaranteed quality mesh generation for curved surfaces. In *Annual ACM Symposium on Computational Geometry*, pages 274–280, 1993.
9. L. P. Chew. Guaranteed-quality Delaunay meshing in 3D. In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 391–393, 1997.
10. N. Chrisochoides and D. Nave. Parallel Delaunay mesh generation kernel. *Int. J. Numer. Meth. Engng.*, 58:161–176, 2003.

11. N. P. Chrisochoides. A survey of parallel mesh generation methods. Technical Report BrownSC-2005-09, Brown University, 2005. Also appears as a chapter in Numerical Solution of Partial Differential Equations on Parallel Computers (eds. Are Magnus Bruaset, Petter Bjorstad, Aslak Tveito), Springer Verlag.
12. S. Dong, D. Lucor, and G. E. Karniadakis. Flow past a stationary and moving cylinder: DNS at Re=10,000. In *2004 Users Group Conference (DOD_UGC'04)*, pages 88–95, 2004.
13. P.-L. George and H. Borouchaki. *Delaunay Triangulation and Meshing. Application to Finite Elements*. HERMES, 1998.
14. L. Linardakis and N. Chrisochoides. A static medial axis domain decomposition for 2d geometries. *ACM Transactions on Mathematical Software*, 2005. Submitted.
15. L. Linardakis and N. Chrisochoides. Delaunay decoupling method for parallel guaranteed quality planar mesh refinement. *SIAM Journal on Scientific Computing*, 27(4):1394–1423, 2006.
16. G. L. Miller. A time efficient Delaunay refinement algorithm. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 400–409, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.
17. G. L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. A Delaunay based numerical method for three dimensions: Generation, formulation, and partition. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 683–692. ACM Press, May 1995.
18. S. A. Mitchell. Cardinality bounds for triangulations with bounded minimum angle. In *Proceedings of the 6th Canadian Conference on Computational Geometry*, pages 326–331, Aug. 1994.
19. D. Nave, N. Chrisochoides, and L. P. Chew. Guaranteed–quality parallel Delaunay refinement for restricted polyhedral domains. *Computational Geometry: Theory and Applications*, 28:191–215, 2004.
20. J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.
21. S. Schneider, C. D. Antonopoulos, A. N. Chernikov, D. S. Nikolopoulos, and N. P. Chrisochoides. Designing effective memory allocators for multicore and multithreaded systems: A case study with irregular and adaptive applications. Submitted to the Supercomputing Conference, 2006.
22. J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.
23. J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1–3):21–74, May 2002.
24. A. Üngör. Off-centers: A new type of Steiner points for computing size-optimal guaranteed-quality Delaunay triangulations. In *Proceedings of LATIN*, pages 152–161, Apr. 2004.
25. R. A. Walters. Coastal ocean models: Two useful finite element methods. *Recent Developments in Physical Oceanographic Modelling: Part II*, 25:775–793, 2005.
26. D. F. Watson. Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes. *Computer Journal*, 24:167–172, 1981.