CHAPTER III

PROBLEM DECOMPOSITION

Motivation and Literature

After the early work by Melzak [8], many people began to work on the Steiner Minimal Tree problem. The first major effort was to find some kind of geometric bound for the problem. In 1968 Gilbert and Pollak [6] showed that the SMT for a set of points, $\mathcal{S}$, must lie within the Convex Hull of $\mathcal{S}$. This bound has since served as the starting point for every bounds enhancement for SMT's.

As a brief review, the Convex Hull is defined as follows: Given a set of points $\mathcal{S}$ in the plane, the Convex Hull is the convex polygon of smallest area containing all the points of $\mathcal{S}$. A polygon is defined to be convex if a line segment connecting any two points inside the polygon lies entirely within the polygon. An example of the Convex Hull for a set of 100 randomly generated points is shown in Figure 8.

Shamos in his PhD thesis [26] proposed a Divide and Conquer Algorithm which has served as the basis for many parallel algorithms calculating the Convex Hull. One of the first such approaches appeared in the PhD thesis by Chow [27]. This approach was refined and made to run in optimal $\mathcal{O}(log\ n)$ time by Aggarwal et.al. [28], and Attalah and Goodrich [29].

The next major work on the SMT problem was in the area of problem decomposition. As with any non-polynomial algorithm, the most important theorems are those that say "If property $\mathcal{P}$ exists, then the problem may be split into the following sub-problems." For the Steiner Minimal Tree Problem property $\mathcal{P}$ will probably be geometric in nature. Unfortunately, decomposition theorems have been few and far between for the SMT problem. In fact, at this writing there have been only three such theorems.

The Double Wedge Theorem

The first decomposition theorem, known as the Double Wedge Theorem, was proposed by Gilbert and Pollak [6]. This is illustrated in Figure 9 and can be summarized quite

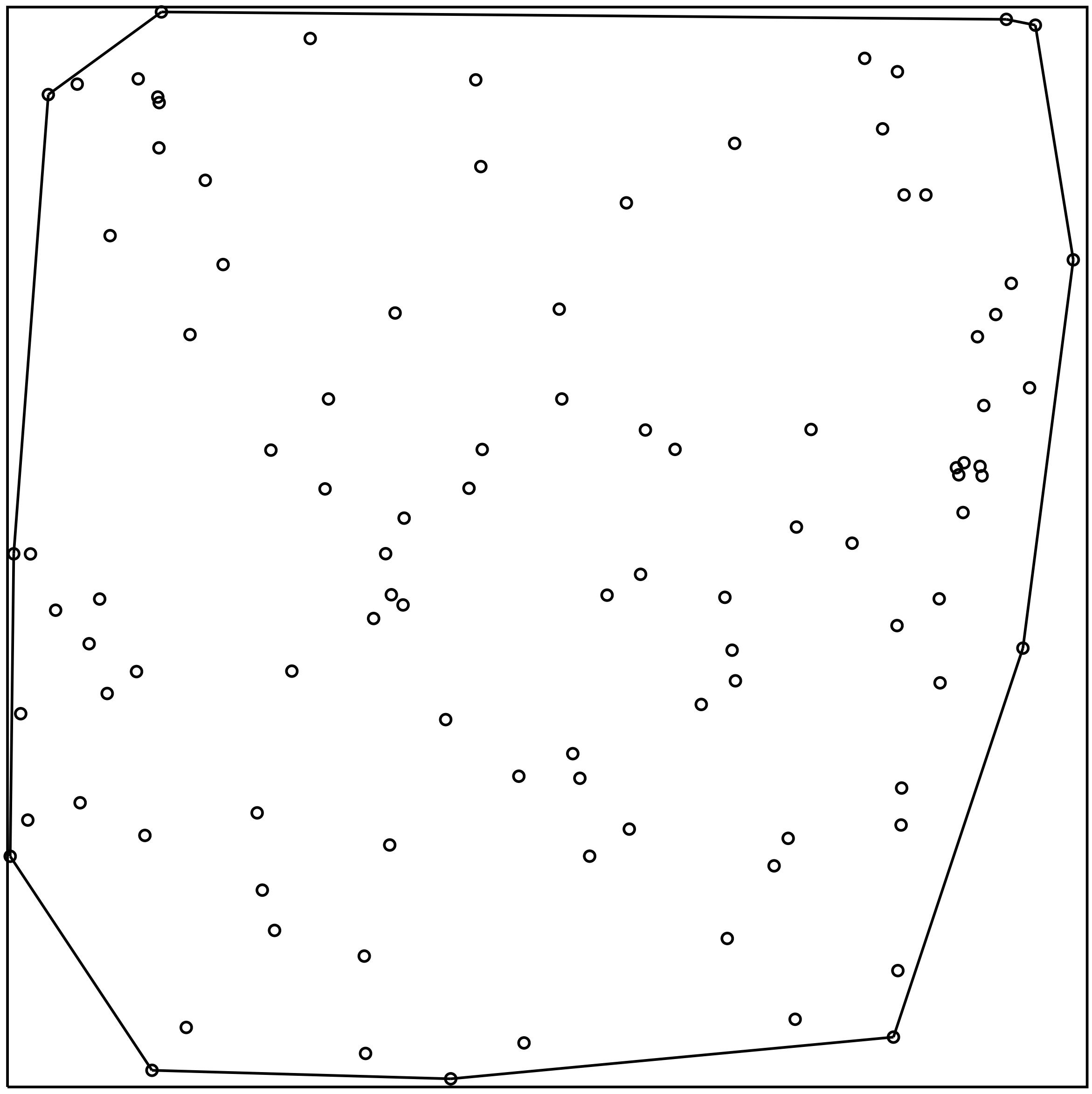

Figure 8. The Convex Hull for a random set of points.

simply as follows: If two lines intersect at point $\mathcal{X}$ and meet at 120°, they split the plane into two 120° wedges and two 60° wedges. If R_1 and R_2 denote the two 60° wedges, and all the points of $\mathcal{S}$ are contained in $R_1 \bigcup R_2$, then the problem can be decomposed. There are two cases to be considered. In case 1 $\mathcal{X}$ is not a point in $\mathcal{S}$; therefore, the Steiner Minimal Tree for $\mathcal{S}$ is the SMT for R_1, the SMT for R_2, and the shortest edge connecting the two trees. In case 2 $\mathcal{X}$ is a point in $\mathcal{S}$; therefore, the Steiner Minimal Tree for $\mathcal{S}$ is the SMT for R_1 and the SMT for R_2. Since $\mathcal{X}$ is in both R_1 and R_2 the two trees are connected.

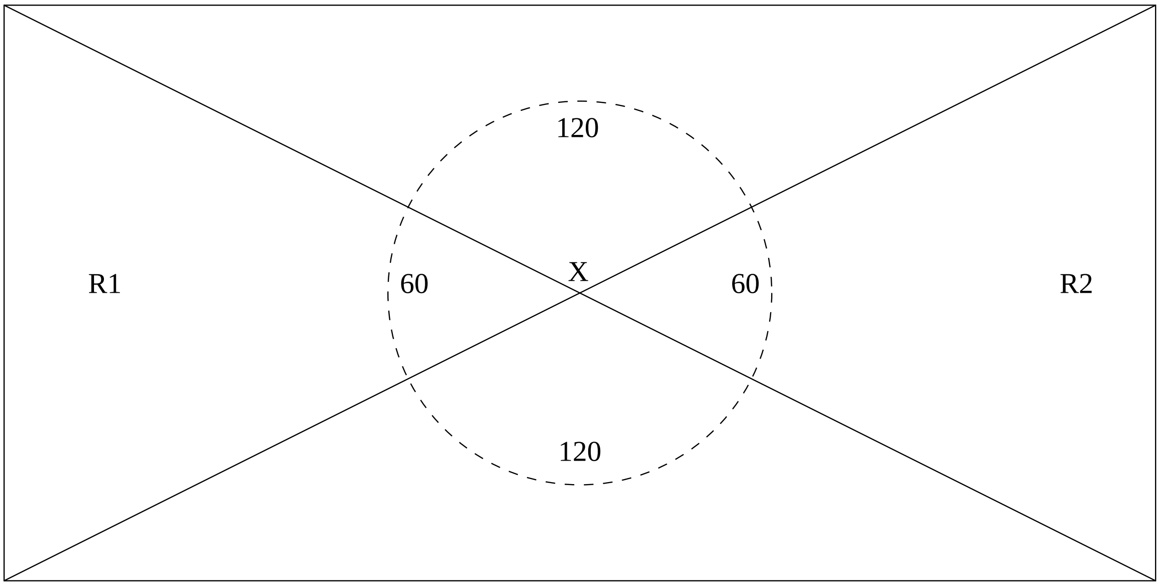

Figure 9. An illustration of the Double Wedge.

The Steiner Hull

The next decomposition theorem is due to Cockayne [30] and is based upon what he termed the *Steiner Hull.* The Steiner Hull is defined as follows: let P_1 be the convex Hull. P_{i+1} is constructed from P_i by finding an edge (p, r) of P_i that has a vertex (q) near it such that $\angle pqr \geq 120°$ and there is not a vertex inside the triangle pqr. The final polygon, P_i, that can be created in such a was is called the Steiner Hull. The Algorithm for this construction is shown in Figure 10. The Steiner Hull for the 100 points shown in Figure 8 is given in Figure 11.

```
The initial Steiner Polygon, P_1, is the Convex Hull.
Repeat
     Create Next Steiner Polygon P_{i+1} from P_i by
          1) find a set of points pqr ∈ S such that:
               p and r are adjacent on P_i
               ∠pqr ≥ 120°
               ∄ a point from S in the triangle pqr
          2) remove the edge pr.
          3) add edges pq and qr.
Until(P_i == P_{i+1})
Steiner Hull = P_i
```

Figure 10. The Steiner Hull algorithm.

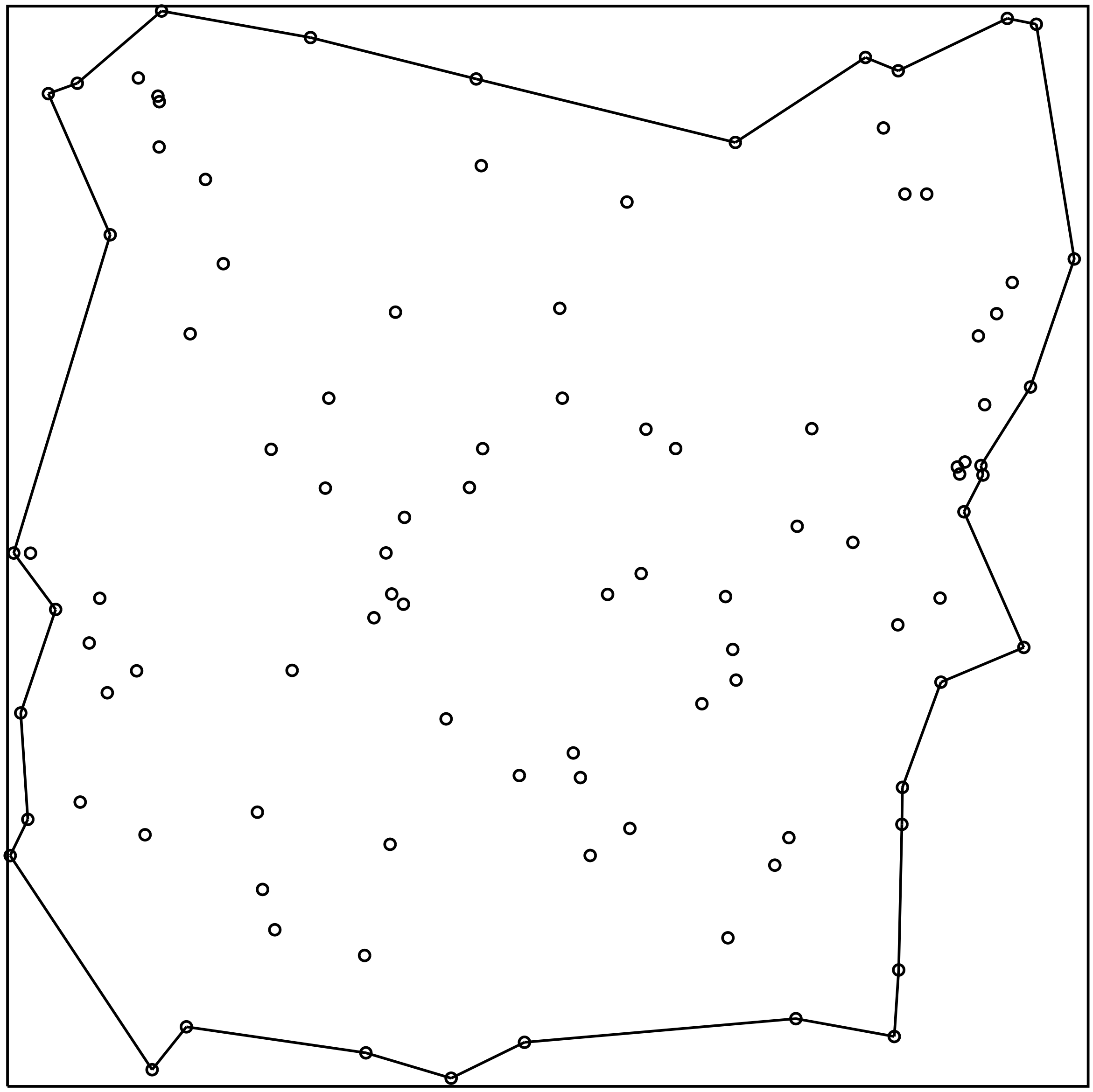

Figure 11. The Steiner Hull for a random set of 100 points.

After defining the Steiner Hull, Cockayne showed that the SMT for $\mathcal{S}$ must lie within the Steiner Hull of $\mathcal{S}$. This presents us with the following decomposition: The Steiner Hull can be thought of as an ordered sequence of points, $\{p_1, p_2, \ldots, p_n\}$, where the hull is defined by the sequence of line segments, $\{p_1p_2, p_2p_3, \ldots, p_np_1\}$. If there exists a point p_i that occurs twice in the Steiner Hull, then the problem can be decomposed at point p_i. If a Steiner Hull contains such a point, then the Steiner Hull is referred to as *degenerate*. This decomposition is accomplished by showing that the Steiner Hull splits $\mathcal{S}$ into two contained subsets, R_1 and R_2, where R_1 is the set of points contained in the Steiner Hull from the first time p_i appears until the last time p_i appears, and R_2 is the set of points contained in the Steiner Hull from the last time p_i appears until the first time p_i appears. With this decomposition it can be shown that $\mathcal{S} = R_1 \bigcup R_2$, and the SMT for S is the union of the SMT for R_1 and the SMT for R_2. This decomposition is illustrated in Figure 12. Cockayne also proved that the Steiner Hull decomposition includes every decomposition possible with the Double Wedge Theorem.

In their work on 100 point problems, Cockayne and Hewgill [25] mention that approximately 15% of the randomly generated 100 point problems have degenerate Steiner Hull's. The Steiner Hull shown in Figure 11 is not degenerate, while that in Figure 12 is.

The Steiner Hull Extension

The final decomposition belongs to Hwang, Song, Ting and Du [31]. They proposed an extension to the Steiner Hull as defined by Cockayne. Their extension is as follows:
If there exist four points a, b, c, d on a Steiner Hull such that:

1. a, b, c, d form a convex quadrilateral,
2. there does not exist a point from $\mathcal{S}$ in the quadrilateral (a, b, c, d),
3. $\angle a \geq 120°$ and $\angle b \geq 120°$,
4. the two diagonals (ac) and (bd) meet at O, and $\angle bOa \geq \angle a + \angle b - 150°$,

then the SMT for $\mathcal{S}$ is the union of the SMT's for R_1 and R_2 and the edge ab where R_1 is the set of points contained in the Steiner Hull from c to b with the edge bc, and R_2 is the set of points contained in the Steiner Polygon from a to d with the edge ad. This decomposition is illustrated in Figure 13. These three decomposition theorems will be combined in the parallel algorithm for decomposition presented next.

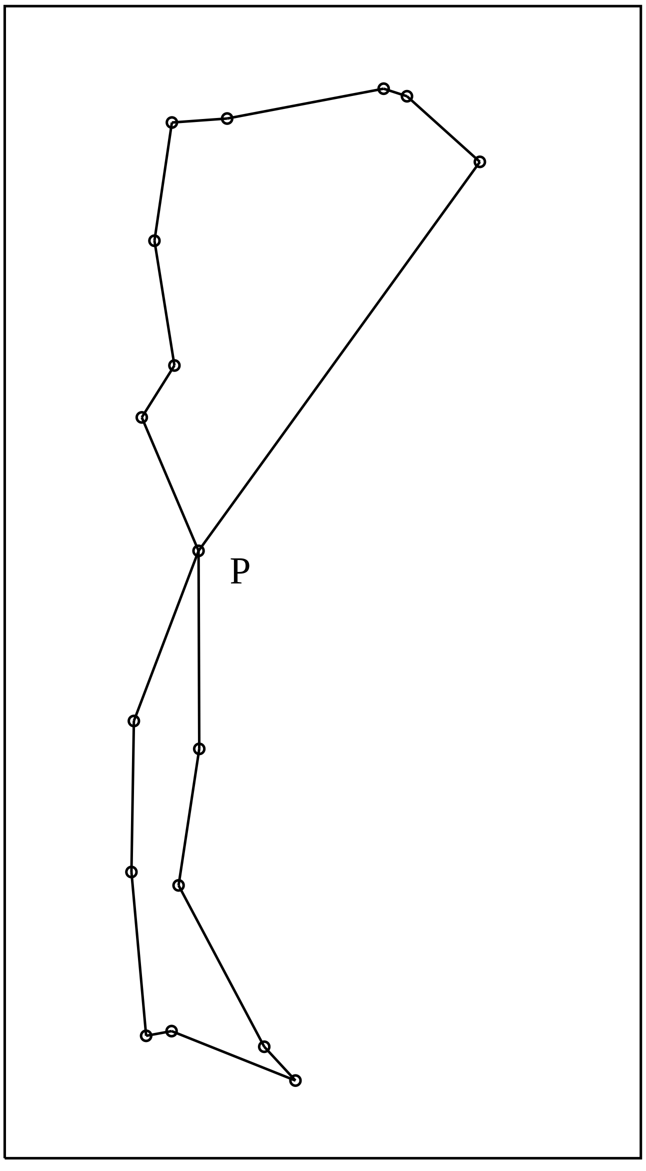

Figure 12. An illustration of the Steiner Hull decomposition.

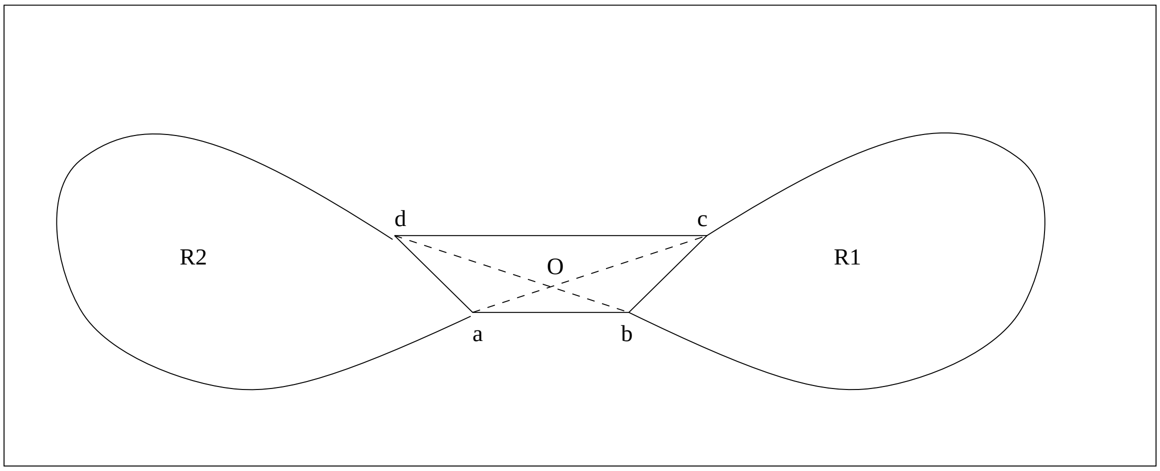

Figure 13. An illustration of the Steiner Hull extension.

A Parallel Algorithm to Calculate the Steiner Hull

A Introduction to Parallelism

Parallel computation is allowing us to look at problems that have previously been impossible to calculate, as well as allowing us to calculate faster than ever before problems we have looked at for a long time. It is with this in mind that we begin to look at a parallel algorithm for the decomposition of the Steiner Minimal Tree problem.

There have been volumes written on parallel computation and parallel algorithms; therefore, the author will not rehash the material that has already been so excellently covered by many others more knowledgeable on the topic, but will refer the interested readers to various books currently available. For a thorough description of parallel algorithms, and the PRAM Model the reader is referred to the book by Joseph JáJá [32], and for a more practical approach to implementation on a parallel machine the reader is referred to the book by Vipin Kumar et.al. [33] or the book by Justin Smith [34].

The Parallel Algorithm

The algorithm presented here is designed to operate on a CREW PRAM machine, which is a shared memory machine which allows concurrent reads from memory but allows only one processor to write to a specific location at a time. Other models allow concurrent writing to the same location, but this requires an arbitration scheme for what is being written.

This algorithm starts with its input being the Convex Hull for $\mathcal{S}$. The vertices on the Convex Hull are placed in an array called *Edges*. This array is made up of structures, where each structure contains an integer (the vertex number) and two pointers that form a doubly linked list through the ordered entries. Initially the linked list goes straight through the edges of the Convex Hull. Two counters are kept for each round. The first counter is named *Previous_count* and is initialized to 0. This counter keeps track of the count of the beginning vertex of the first edge alive on this round. The second counter is named *Current_count* and it is initialized to the number of edges in the Convex Hull. This counter keeps track of where new edges created this round will begin to be placed. Processor i is assigned to the edge defined by vertices $i, i+1$ and the calculation begins. Each processor determines whether the edge it was assigned is to be replaced via the Steiner Hull definition and it's extension. An integer representing the number of edges that will replace this edge (0 – no replacement, 2 – standard Steiner Hull definition, or 3 – Steiner Hull Extension) is placed into *New_Edges*[i]. The prefix sum is then calculated for this array, which results in *New_Edges*[i] $= \sum_{j=0}^{i}$. This value determines the offset into *Edges* where the new edges are to be placed. Processor i places its new edges into the array beginning at $Current\_count + New\_Edges[i-1]$. Processor i then updates the doubly linked list through *Edges*, starting with the previous link pointing to edge i, updating the list through the new edges, and then updating the next link at i. The processor assigned to edge *Current_count* - 1 then does the end of round updating: $Previous\_count = Current\_count$, $Current\_count = Current\_count + New\_Edges[i]$. For the next round the processors are assigned the edges from *Previous_count* to *Current_count*, and the calculation begins again. This continues until $Previous\_count == Current\_count$.

When analyzing this algorithm, whose code is given in Figure 14, There is a case where only one edge will find a new set of edges to add to the current Steiner Polygon. This makes the algorithm end up with $\mathcal{O}(n)$ rounds of computation. In practice however, for the 100 point point problems we solved here, we never saw the algorithm go past 4 rounds of computation.

```
Parallel_Steiner_Hull()
     Input:  The Convex Hull for S
     Output:  The Steiner Hull for S
{
     int i,k;

     Previous_count = 0;
     Current_count = Convex_Hull.size;
     par_for(i = 0; i < Current_count; i++) {
          Edges[i] = Convex_Hull.edges[i];
     }
     do {
          par_for(i = Previous_count; i < Current_count; i++){
               int j;
               j = i - Previous_count;
               New_Edges[j] = Number_of_new_edges(i);
               P_S[j] = Prefix_Sum(j);
               for(k=0; k < New_Edges[j].count; k++){
                    Edges[Current_count + P_S[j-1] + k] =
                         New_Edges[j].edges[k];
               }
               Update_Links(i,Current_count + P_S[j-1], k);
               if(i == Current_count -1){
                    Previous_count = Current_count;
                    Current_count = Current_count + P_S[j];
               }
          }
     } while(Previous_count != Current_count);
}
```

Figure 14. A parallel Steiner Hull algorithm.