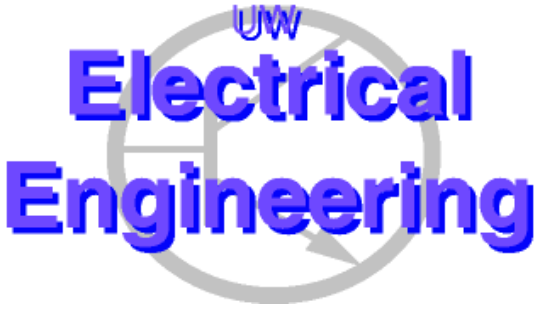
Track Placement: Orchestrating Routing Structures to Maximize Routability

Katherine Compton
kati@ece.northwestern.edu
Dept of ECE, Northwestern University
Evanston IL, 60208-3118

Scott Hauck
hauck@ee.washington.edu
Dept of EE, University of Washington
Seattle WA, 98195-2500



UWEE Technical Report
Number UWEETR-2002-0013
October 24, 2002

Department of Electrical Engineering
University of Washington
Box 352500
Seattle, Washington 98195-2500
PHN: (206) 543-2150
FAX: (206) 543-3842
URL: http://www.ee.washington.edu

Track Placement: Orchestrating Routing Structures to Maximize Routability

Katherine Compton
kati@ece.northwestern.edu
Northwestern University
Evanston IL, 60208-3118

Scott Hauck
hauck@ee.washington.edu
Dept of EE, University of Washington at Seattle
Seattle WA, 98195-2500

University of Washington, Dept. of EE, UWEETR-2002-0013
October 24, 2002

Abstract

The design of a routing channel for an FPGA is a complex process, requiring the careful balance of flexibility with silicon efficiency. With the growing move towards embedding FPGAs into SoC designs, and the opportunity to automatically generate FPGA architectures, this problem becomes even more critical. The design of a routing channel requires determining the number of routing tracks, the length of the wires in those tracks, and the positioning of the breaks on the tracks. This paper focuses on the last of these problems, the placement of breaks in tracks to maximize overall flexibility. We have developed both an optimal algorithm and a number of heuristics to solve the track placement problem. The optimal algorithm finds a best solution provided the problem meets a number of restrictions. Most of the heuristics are without restrictions, and the most promising of these find solutions on average within 1.13% of optimal.

1 Introduction

The design of an FPGA interconnect structure has usually been a hand-tuning process. How many wires should there be, and how long should they be? Where should the breaks in the wires be placed? A human designer, with the aid of benchmark suites and trial-and-error, develops an interconnect structure that attempts to balance flexibility with silicon efficiency. Often, the concentration is on picking the number and length of tracks – long tracks give global communication but with high silicon and delay costs, while short wires can be very efficient only if signals go a relatively short distance.

An area that can sometimes be ignored is the placement of the breaks in these interconnect wires. If we have a symmetric interconnect, where there are N length-N wires, we simply break one wire at each cell and have an even interconnect structure. However, for more irregular interconnects, it is more difficult to determine the best positioning of these breaks. Although the break positioning can have a significant impact on the routability of an interconnect structure, there is little insight into how to quantify this effect, and optimize the break positioning. This becomes even more critical when we consider the automatic generation of FPGA architectures, particularly as subsystems for SoC designs [Compton02].

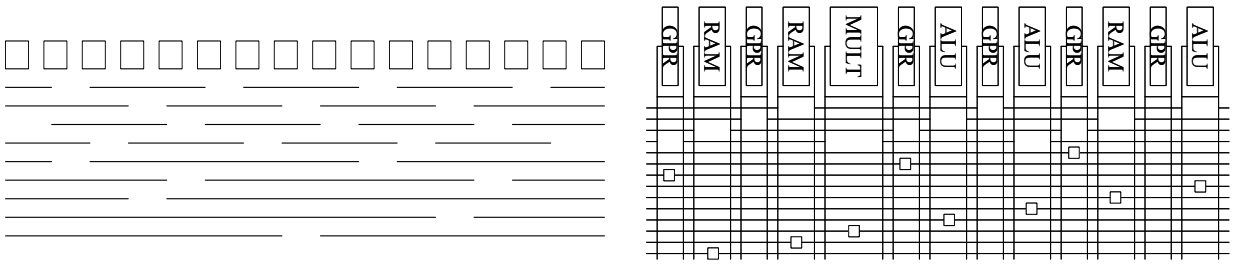


Figure 1: Two examples of reconfigurable architectures with segmented channels. At left, Garp [Hauser97]. On the right, RaPiD [Cronquist99].

In this paper, we address the issue of routing architecture design for reconfigurable architectures with segmented channel routing, such as RaPiD [Cronquist99], Chimaera [Hauck97], Garp [Hauser97] and Triptych [Borriello95]. We formalize the track placement problem, define an optimization metric, and introduce multiple algorithms, including a proven optimal algorithm for a subset of these problems.

Achieving the best track placement requires a careful balance the breaks on multiple, different-length routing tracks. For example, in the right half of Figure 2, the breaks between wires in the routing tracks are staggered. This helps to provide similar routing options regardless of location in the array. If instead for the same tracks all breaks were lined up under a single unit, as in the left part of Figure 2, a signal might become very difficult to route if its source was on one side of the breaks and at least one sink on the other. When large numbers of tracks or tracks of different wire lengths are involved, it can become difficult to find a solution where the breaks are evenly distributed through the array.

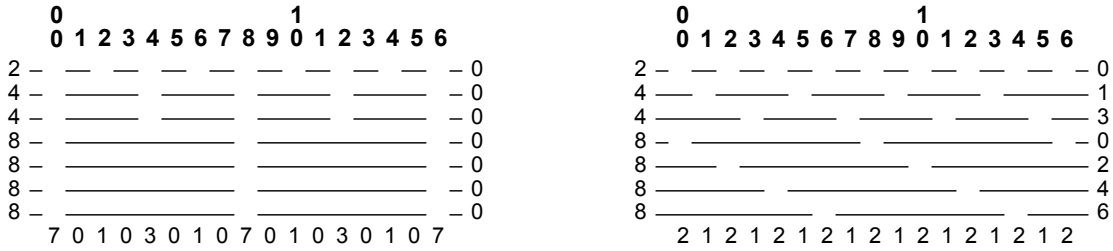


Figure 2: Two different track placements for the same type of tracks, a very poor one (left) and a very good one (right). In each, the numbers on top indicate position in the architecture, and the numbers on the bottom indicate the number of breaks at the given position. Each track has its associated wire length at left, and offset at right.

The issue of determining the positioning (or offset) of a track within an architecture is referred to as track placement. The track placement problem is to pick the offset that should be used for each track in order to maximize the routability, given a predetermined set of tracks with fixed-length wires. For simplicity, each track is restricted to contain wires of only one length, which is referred to as the track's S value, or sometimes track length. The actual determination of the number and S values of tracks is discussed elsewhere [Compton02].

2 Problem Description

Finding a good track placement is a complex task. Intuitively, we wish to space tracks with the same S value evenly, such as by placing the length-8 tracks in the right half of Figure 2 at offsets 0, 2, 4, and 6. However, a placement of tracks of one S value can affect the best placement for tracks of another S value. For example, Figure 2 right shows that the length-4 tracks are placed at offsets 1 and 3 in order to avoid having the breaks of those tracks fall at the same locations as the breaks from the length-2 and length-8 tracks. If we place the length-2 track instead at position 1, either the length-4 tracks should be shifted to offsets 0 and 2 or the length-8 tracks should be shifted to offsets 1, 3, 5, and 7 in order to maintain evenness of the breaks. This effect is called "correlation" between the affected tracks. Correlations occur between tracks when their S values contain at least one common prime factor, which is explained in more depth later. It is these correlations that make track placement difficult.

We have developed a number of algorithms, of varying complexity, to solve the track placement problem. To determine the quality of these algorithms, we must somehow measure whether a particular placement of a set of tracks is superior to another placement of the same set of tracks. Therefore, we require a quantitative value for comparison. We would like to use a measurement based on the number of different tracks onto which a particular signal may be routed. Because, however, we wish to determine overall routability of a given architecture rather than routability for a particular signal, we use a slightly more abstract approach.

Our goal in track placement is to assure that signals of all lengths have the most possible routes, or conversely, that the fewest bottlenecks exist. To quantitatively measure this routability, we examine each possible signal length, and find the region of the interconnect that gives the fewest possible routes for this length signal. Summing this number across all possible signal lengths gives the "diversity score" of a track placement, as shown in Equation 1. The fewer and smaller the routing bottlenecks, the more routing choices are available throughout the architecture, and the higher the diversity score. We have also determined a bound on this value, as described in Theorem 1. It should be noted that comparing diversity scores is only valid across different solutions to the same track placement problem. Determining the actual tracks that should be used is a separate issue [Compton02].

Equation 1: The diversity score for a particular track placement routing problem T and solution set of offsets O can be calculated using the equation:

$$diversity\_score(T,O) = \sum_{L}\left(\min_{all\ \ positions}\left(\sum_{Ti\in T} uncut(T_i, O_i, L, position)\right)\right),$$

for all possible signal lengths L, and all possible positions in the interconnect.

Theorem 1: For all possible offset assignments O to track set T,

$$diversity\_score(T,O) \le \sum_{L} floor\left(|T| - \sum_{Ti\in T}\min(1, L/S_i)\right)$$

We focus on the worst-case (the regions with the fewest possible routes) instead of the average case. This is because the average number of possible routes for a signal is independent of track placement, and only depends on the number of tracks and their wire lengths. Thus, an average case metric cannot tell the difference between the two placements in Figure 2, while the worst-case clearly shows that the spread version at right is superior.

3 Proposed Algorithms

Our track placement algorithms, both optimal and heuristic, are discussed in depth in the next few sections. The first algorithm, Brute Force, simply tests all possible track placements, and returns a solution with the highest diversity score. This algorithm, however, runs extremely slowly, and is impractical for either large problems or large numbers of runs for architecture exploration. The next algorithm, Simple Spread, treats tracks of each S value as separate problems, spacing tracks evenly within each set. While this can provide a fast solution, it does not consider the potential correlations between tracks of different segment lengths. Power2 is another very simple algorithm which does consider inter-set effects, but is restricted to handle tracks with segment lengths that are powers of two. This particular algorithm was used to solve the track placement problem for a 1D reconfigurable routing architecture generator [Compton02].

The final two algorithms are more complex in operation than the other algorithms, yet require significantly less computation time than the Brute Force algorithm. Pseudocode for each of these last two algorithms is given. The Optimal Factor algorithm provably finds an optimal solution provided the problem meets a number of key restrictions. The Relaxed Factor algorithm is an extension of the ideas and techniques from the Optimal Factor algorithm so that it can operate without restrictions. While it is not guaranteed to provide an optimal solution in every case, it provides very high quality results in most cases.

3.1 Brute Force Algorithm

Using a brute-force approach, we can guarantee finding a solution with the highest possible diversity score. However, as with many complex problems, examining the entire solution space is a very slow process. While useful for finding the best possible solution for a final design, this method is inappropriate for situations where a large

number of different routing architectures (each with different types and or quantities of tracks) are being examined—perhaps even within an inner loop [Compton02]. Therefore, for this work the purpose of a brute force approach is to provide a bound on the diversity score. This bound is then used to verify the results of our optimal algorithm as well as provide a frame of reference for the quality of the solutions found by the other algorithms.

3.2 Simple Spread Algorithm

As stated previously, the Simple Spread algorithm is a very straightforward heuristic for performing track placement. In this particular algorithm, we group tracks by segment length. Each track group is considered as a separate problem. Each group first has any "full sets" (where the number of tracks is equal to the segment length) placed, one offset apart, filling all possible offsets for that particular S value. Once the number of tracks in a set is less than the S value for the set, these remaining tracks are spaced evenly throughout the possible offsets 0 through S-1 using the equation O_i=floor(S*i/|G|), where G is the set of remaining tracks of segment length S, i is the index 0…|G|-1 of a particular track in the set, and O_i is the calculated offset of said track.

This algorithm, while simple in its operation and fast in execution, disregards the fact that a placement decision at a given S value can affect the quality of a placement decision at a different S value due to potential correlations. Figure 3 shows a small case in which the Simple Spread returns a track placement significantly worse than the Brute Force optimal placement. The diversity score for the Simple Spread solution is only 6, whereas the Brute Force optimal solution has a diversity score of 9. Looking at the number of breaks occurring at each offset (the bottom row of numbers on each of the diagrams), we see that the solution found by the Brute Force method is significantly smoother than the one found by Simple Spread, which leads directly to the difference in diversity scores.

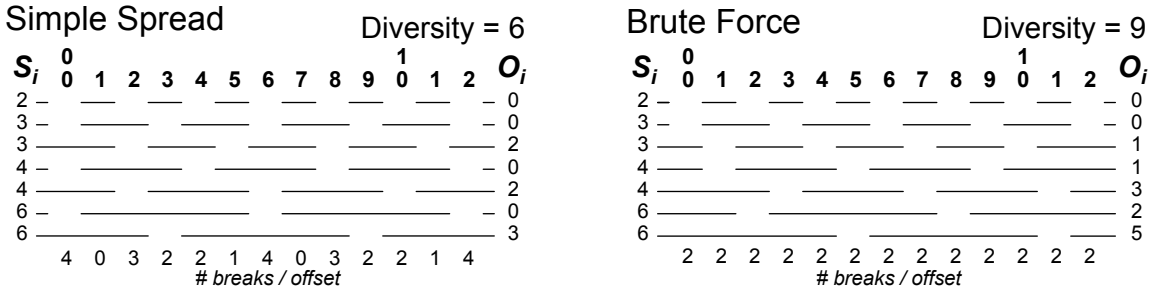


Figure 3: A Simple Spread solution and a Brute Force solution to the same problem. For each diagram, the top row of numbers indicates the position, and the bottom row indicates the number of breaks occurring at the corresponding position. Each track is labeled on its left with its S value and on its right with its assigned offset. The diversity score of each solution is also given in the upper right-hand corner.

3.3 Power2 Algorithm

The Power2 algorithm, unlike Simple Spread, focuses on compensating for the correlation between the S values of tracks more than the even spreading of tracks within a particular S value. This algorithm, however, is restricted to situations where all tracks have S values which are a power of 2. In this case, the offset for a particular track is calculated based on the offsets and S values of all previously placed tracks. While initially this may seem a complex task, we have simplified it such that we only need to remember the placement location of the track immediately previous to the current track being considered.

First, the tracks in a placement problem are grouped by S values, and these groups are sorted in increasing order. Offsets are then assigned in a particular constant pattern dependent upon the S value (as indicated in the example of Figure 4). The tracks in the group with the smallest S are assigned using their pattern. Afterwards, we move to the next S group, with its pattern, and start with the offset that was next in the previous pattern. Therefore, the algorithm attempts to maintain a somewhat even density of breaks throughout the array. This does mean that the breaks of a particular S group may not be evenly distributed, such as when the number of tracks in the group is not a power of 2, but the goal of the algorithm is to compensate for this by using the correlations between S groups to its advantage.

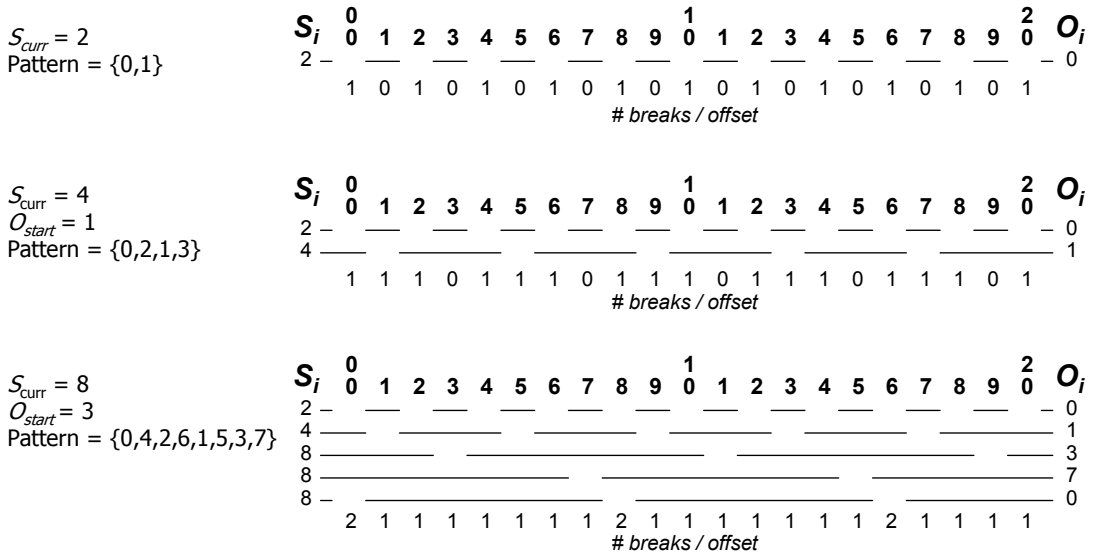


Figure 4: An example of the operation of Power2 at each of three S values for a case with one length-2 track, one length-4 track, and three length-8 tracks. The offset pattern at each S value is given, along with the offset O_{start} to use for the first track in that S group based on the placement of the previous tracks. For each diagram, the top row of numbers indicates the position, and the bottom row indicates the number of breaks occurring at the corresponding position. Each track is labeled on its left with its S value and on its right with its assigned offset.

3.4 Optimal Factor Algorithm

One of our goals was to develop a fast algorithm that, with some restrictions, would provably find the optimal solution. The Optimal Factor algorithm is the culmination of many theorems and proofs. The proofs of these theorems and lemmas appear in the Appendix. We use the same numbering as the appendix for identifying the theorems and lemmas, although they may be stated more simply in this part of the text. This section presents various track placement situations, theorems regarding their solutions, and finally the pseudocode for the Optimal Factor Algorithm itself.

As we stated previously, correlations across S values can occur when those S values share a common prime. Any optimal algorithm must therefore consider the correlations between breaks not only within a particular S group, but also across S groups. The example of Figure 3 illustrates correlation, and the fact that it is transitive. Because the S=2 group and the S=6 group are correlated, there is also a correlation between the S=2 group and the S=3 group, and consequently, between the S=3 group and the S=4 group. But if an additional track group of S=5 were to be introduced into the problem, it would not share a factor with any other track group, and would not be correlated with any other track group in the problem. Regardless of offset choice, the length-5 track would overlap with breaks from each of the other tracks at various positions in the architecture. Therefore, it can be placed independently of the other tracks. This feature is embodied in the following theorem, which is used to divide a track placement problem into smaller independent problems when possible.

> **Theorem 2:** If T can be split into two sets G1⊂T and G2 = T-G1, where the S values of all tracks in G1 are relatively prime with the S values of all tracks in G2, then diversity_score(T,O) = diversity_score(G1, O1) + diversity_score(G2, O2), where O is the combination of sets O1 and O2. Thus, G1 and G2 may be solved independently.

We can also use the fact that correlation is based on common factors to further simplify the track placement problem. Theorem 5 states that whenever one track's S value has a prime factor that is not shared with any other track in the problem (or a track has a higher quantity of a prime factor than any other track in the problem), the prime factor can be effectively removed. After all "extra" prime factors have been removed, the remaining prime factors are all shared by at least two tracks. This can help to make interactions between tracks more obvious. For example, if there are 2 length-6 tracks and one length-18 track, we can remove a 3 from 18, and then in effect we

have 3 length-6 tracks, which can be placed evenly using a later theorem (Theorem 7). Note that we do not actually change the S value of the length-18 track, just the S value we use during the course of track placement. After the offsets of tracks are determined, the original S values are restored if necessary.

> **Theorem 5:** If the S_i of unplaced track T_i contains more of any prime factor than the S_j of each and every other track T_j ($i \neq j$), then for all solutions O, diversity_score(T, O) = diversity_score(T', O), where T' is identical to T, except T'_i has $S'_i = S_i$ with the unshared prime factor removed. This rule can be applied recursively to eliminate all unshared prime factors.

For the rest of the optimal algorithm, the tracks are grouped according to S value (where the S value may have been factored using Theorem 5). These groups are then sorted in decreasing order, such that the largest S value is considered first. Any full sets are removed using Theorem 4a, where a full set is defined as a set of N tracks all with $S_i = N$. In other words, we have enough tracks to fill all potential offset positions 0…N-1. If we can solve the remainder of the track placement problem to meet the bound of Theorem 1, then the overall placement which also meets the bound places one track from the full set at each potential offset position for the S value of that set.

> **Theorem 4a:** Given a set $G \subset T$ of tracks, each with S=|G|. If there exists a solution O' for tracks T'=T-G that meets the bound, then there is also solution O for tracks T that meets the bound (and thus is optimal), where each track in G is placed at a different offset 0…|G|-1.

Note that throughout this algorithm, as well as all of the others, we are only considering as possible offsets the range 0 through $|S_i|$-1 for a track with $S=S_i$. While it is perfectly valid to place a track at an offset greater than S_i-1, the fact that all wires within the track are of length S_i causes a break to be located every S_i locations. Therefore, there will always be a break on track T_i within the range 0 to S_i-1, which can be found as dictated in Lemma 1.

> **Lemma 1:** If a track T_i has a break at position P, it will also have a break at position $P \% S_i$. Therefore, all possible offset locations for track T_i can be represented within the range 0 through S_i-1.

The basic idea of the remainder of the algorithm is to space the tracks of the largest S value (S_{max}) evenly, using Theorem 4a above and Theorem 7 below, then remove these tracks from further consideration in the problem. However, in order to compensate for the effect that their placement has on the optimal placement of tracks from successive S groups, we add a number of placeholder tracks into the problem, of the next largest S value S_{next}, with pre-assigned offsets. These placeholder tracks together have the same number of breaks at the same positions as the original tracks that we removed, but in terms of $S=S_{next}$ instead of S_{max}. This enables us to accurately determine the best offsets for the real tracks with $S=S_{next}$. This process is repeated within a loop, where S_{next} now becomes S_{max} (since the tracks of the old S_{max} were removed from the problem), and S_{next} becomes the next smaller S below the new S_{max}.

We do set a number of restrictions, however, in order to ensure that at each loop iteration, (1) we can perfectly evenly space the tracks with $S=S_{max}$, and (2) the breaks from the set of tracks with $S=S_{max}$ can be exactly represented by some integer number of tracks with $S=S_{next}$. These restrictions are outlined in the next two theorems.

> **Theorem 7:** Let S_{max} be the maximum S value currently in the track placement problem, M be the set of all tracks with $S = S_{max}$, N = |M|, and S_{next} be the largest $S \neq S_{max}$. If N>1, S_{max} is a proper multiple of N, and $S_{next} \leq S_{max}*(N-1)/N$, then any solution to T that meets the bound must evenly space out the N tracks with $S = S_{max}$. That is, for each track M_i, with a position O_i, there must be another track M_j with a break at $O_i + S_{max}/N$.

> **Theorem 10:** Given a set of tracks G, all of segment length X, where X is evenly divisible by |G|, and a solution O with these tracks evenly distributed. There is another set of tracks G', all of length $Y = |G'| * X / |G|$, with a solution O' where the number of breaks at each position is identical for solution O of G and solution O' of G'. If solution in which G has been replaced with G' meets its bound, the solution with the original G also meets its bound.

Using the theorems we have presented, we can construct an algorithm which is optimal (see Appendix) provided the restrictions in Theorem 7 and Theorem 10 are met. There is one additional restriction that is implied through the combination of the two theorems above. Because the tracks at a given S value must be evenly spread, the offsets assigned to the placeholder tracks added using Theorem 10 in the previous iteration must fall at offsets calculated

using Theorem 7. This is accomplished by requiring that S_{next} also be evenly divisible by the number of pseudotracks of that length added during the track conversion phase. Figure 5 gives the pseudocode for the Optimal Factor Algorithm. This pseudocode is commented to show the relevant theorems for each operation.

```
Optimal_Factor(T) {
    // Theorem 2: Run algorithm independently on relatively prime sets of tracks
    If T can be split into two sets G1⊂T and G2 = T-G1, where the segment length
           of all elements of G1 are relatively prime with the segment length of
           all elements of G2 {
        Optimal_Factor(G1)
        Optimal_Factor(G2)
        Quit
    }

    // Theorem 5: Factor out unshared prime factors from each track's S value
    Initialize Si for each track Ti to the actual segment length of that track.
    While the Si of any track Ti has more of any prime factor P than the Sis of
           all other tracks, individually, in T {
        Set Si = Si/P
    }

    While tracks exist without an assigned Oi {
        // Theorem 4a: Place any full sets
        While there exists a set of G tracks, where Si = |G| {
            For each track in G, assign positions to those without a preassigned
                   Oi, such that there is a track in G at all positions 0≤pos<|G|.
            Eliminate all tracks in G from further consideration.
        }
        If all tracks have their Oi assigned, end.

        // Theorem 7: Smax must be evenly divisible by |M|, and we require
        // that Snext ≤ Smax*(|M|-1)/|M|
        Let Smax = the largest Si amongst unplaced tracks
        Let Snext = the largest Si amongst unplaced tracks such that Snext < Smax
        Let M be the set of all unplaced tracks with Si = Smax
        If no track has an assigned position, assign M1's position O1 to 0.
        Assign all unassigned tracks in M to a position Oi, such that all tracks
               in M are at a position k*Smax/|M|, for all k 0≤k<|M|.
        If all tracks have their Oi assigned, end.

        // Theorem 10: we require Snext = c*Smax/|M| for some integer c≥1
        // Use placeholder tracks of S=Snext to model existing breaks
        // Snext must be evenly divisible by c to make Theorem 7 work
        Add c placeholder tracks, where for j=0..c-1, Sj = Snext, and
               Oj = j*Smax/|M|
        Remove all tracks in M from further consideration
    }
}
```

Figure 5: The pseudocode for the Optimal Factor algorithm. This algorithm is optimal (see Appendix) provided all restrictions listed in the pseudocode are met. The comments list the theorems governing the operations.

3.5 Relaxed Factor Algorithm

While the Optimal Factor Algorithm does generate placements that are optimal in terms of diversity score, there are significant restrictions controlling its use. Because not all architectures may meet the segment length and track quantity restrictions of the optimal algorithm, we have also developed a relaxed version of the algorithm. This relaxed algorithm operates without restrictions on track segment lengths or the quantity of tracks of each length, but may not be optimal.

The general framework of the algorithm remains basically the same as the optimal version, although the placement and track conversion phases differ in operation. Figure 6 shows the highest level of the algorithm. Figure 7 and Figure 11 contain the replacement functions for the placement and track conversion, respectively. Note that there is an additional subfunction that will sometimes be needed for placement, shown in Figure 9.

```
Relaxed_Factor(T) {
    // Run algorithm independently on relatively prime sets of tracks
    If T can be split into two sets G1⊂T and G2 = T-G1, where the segment length
           of all elements of G1 are relatively prime with the segment length of
           all elements of G2 {
        Relaxed_Factor(G1)
        Relazed_Factor(G2)
        Quit
    }

    // Factor out unshared prime factors from each track's segment length
    Initialize Si for each track Ti to the actual segment length of that track.
    While the Si of any track Ti has more of any prime factor P than the Sis of
           all other tracks, individually, in T {
        Set Si = Si/P
    }

    // Place any full sets
    While there exists a set G of tracks where the Si of every track in G equals
           |G| {
        For each track in G, set their Oi = their position in G.
        Remove all tracks in G from further consideration.
    }

    // perform some initializations of structures needed in placement and
    // track conversion
    Initialize tracks[] array to size Smax = largest Si amongst all tracks in T,
           and fill with 0's
    Initialize breaks[] array to size K = LCM of Si's of all tracks in T, and
           fill with 0's.

    // Iteratively assign offsets of unplaced tracks with the longest segment
    // length
    While tracks exist in T without an assigned Oi {
        Let Smax = the largest Si amongst unplaced tracks
        Let Snext = the largest Si amongst unplaced tracks such that Snext < Smax
        Let M be the set of all unplaced tracks with Si = Smax
        Relaxed_Placement(M, tracks[], breaks[])
        If all tracks have their Oi assigned, end.
        Remove all tracks in M from further consideration
        Convert_To_Snext(Snext, tracks[], breaks[])
    }
}
```

Figure 6: The framework of the Relaxed Factor algorithm. Many of the operations are identical to the Optimal Factor algorithm.

As before, we subdivide our problem by considering relatively prime track sets separately. Furthermore, we retain the portion of the optimal algorithm that reduces track segment lengths by removing prime factors not shared with any other track, both to simplify the problem and to make interactions between different segment lengths more apparent. Full sets (where the segment lengths of all tracks in the set is equal to the size of the set) are also handled in the same way they were previously, except the operation is performed once instead of within the loop. This is because while the optimal algorithm adds new placeholder tracks in the loop, and thus may create new full sets, the relaxed algorithm does not; the only full sets that can exist in the relaxed algorithm must be present before the loop starts.

The bulk of the changes between the Optimal Factor and the Relaxed Factor algorithms are in the general placement and placeholder track substitution phases. Where before we used restrictions on track length and quantity to ensure that our methods are optimal, we can no longer rely on these restrictions in the relaxed algorithm. Instead, we use a number of heuristics where the goal is the even spreading of the breaks in tracks. The routing architecture can be considered in terms of a topography, where elevation at a given location is equivalent to the number of breaks at that position. Because "mountains" represent areas with many breaks, and therefore fewer potential routing paths, we attempt to avoid their creation. Instead, we focus on placing our tracks to evenly fill the "plains" in our topography, where the breaks from the additional tracks will have a low effect on the overall routability of the resulting architecture. The top portion of Figure 8 shows the plains and mountains of a sample topography.

The pseudocode for the placement portion of the Relaxed Factor algorithm is shown in Figure 7, with a subfunction in Figure 9. This function uses the *tracks*[] array, which was first initialized in the main body of the algorithm to all 0's. This array is updated to always be the same size as the segment length of the tracks we are currently looking at (again, we examine one segment length at a time). Each position in the array represents a different potential offset that can be chosen for that particular segment length. The actual values in the array essentially represent the number of tracks with breaks at the given potential offset.

Using this information, we attempt to place new tracks at the offsets that have the fewest breaks from the already-placed tracks. The tracks array is updated later in the track transformation function, which will be discussed shortly. For cases where all the values in the *tracks*[] array are identical, we space all remaining unplaced tracks of the current segment length evenly.

```
Relaxed_Placement(M, tracks[], breaks[]) {
   Let unplaced be the number of tracks in M that are unplaced.
   While the number of i's where tracks[i]=min(tracks[]) is ≤ unplaced {
      Place one track at each location with tracks[i]=min(tracks[])
      Update unplaced, tracks[], breaks[]
   }
   If all tracks[i] are the same, space all unplaced tracks in M evenly
   Else if unplaced tracks remain in M {
      Let U be the set of unplaced tracks in M
      Density_Based_Placement(U, unplaced, tracks[])
   }
}
```

Figure 7: The relaxed placement function. The function Density_Based_Placement() appears in Figure 9.

In some cases, we cannot fill the plains in the *tracks*[] array evenly, when there are fewer tracks than positions with the minimum number of breaks. In this case, we use the subfunction in Figure 9 to perform density-based placement. Again, the goal is to distribute the breaks of our tracks as uniformly as possible throughout the architecture. Here we treat the *tracks*[] array (which holds the number of breaks on placed tracks at each position) as circular, with the last position next to position 0. The density-based placement looks at an ever-increasing *region* of this array and places unplaced tracks with the goal of bringing the density of breaks in that *region* close to the *goal_density*. This *goal_density* represents a solution with a completely flat topography. Figure 8 illustrates a sample topography, with plains and mountains labeled, and the corresponding topography if the *goal_density* could be achieved.

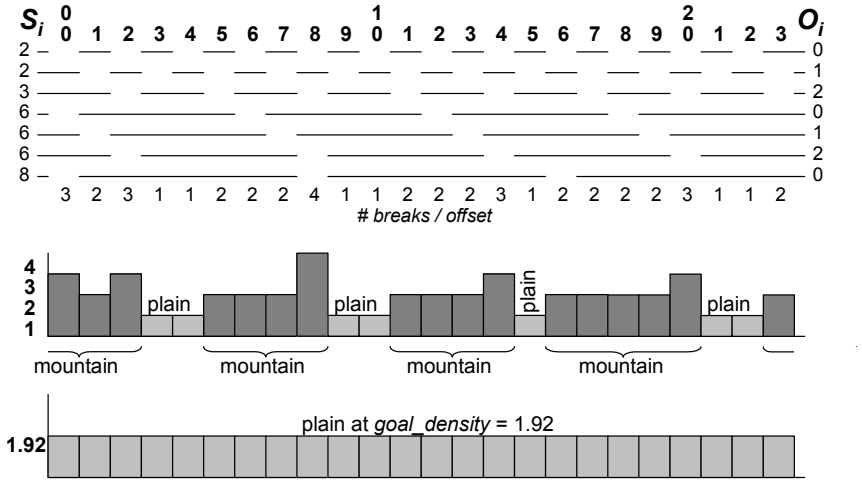


Figure 8: A track arrangement (top), corresponding topography (middle), and the ideal topography for these tracks (bottom). Note that this is an exaggerated example for illustrative purposes. A plain is a series of consecutive values at the lowest elevation, a mountain is a series of consecutive values that are higher than the minimum for the array. The topographies are circular, meaning that in the middle diagram, the value at index 23 is part of the same mountain as indices 0 through 2.

The *region* of consideration will initially be a plain and an adjacent mountain. We calculate the number of tracks to add with breaks positioned in the areas of lowest elevation (a plain). Any given new track can only add at most one break at one position in the *tracks*[] array, as the *tracks*[] array is always the same length as the segment length of the tracks we are placing. The calculated number of tracks is placed such that the breaks are spread evenly through the plain. The *region* is updated to include the next plain and mountain in the circular path around the *tracks*[] array. Again, we calculate the number of tracks to add to the bottom of the new plain in order to bring the density of our entire *region* closer to our ideal density. This pattern continues until we have traveled all the way around the *tracks*[] array back to our beginning position, and the entire *tracks*[] array has become our *region*.

The actual calculation of *num_to_add*, the number of tracks to add to the plain, is somewhat complex. Basically, we would like to minimize abs(potential_density(*region*, *num_to_add*) - *goal_density*), where potential_density(*region*, *num_to_add*) is the density of the current region if *num_to_add* tracks are added with offsets in that region, and *goal_density* is the density if all tracks in the problem could be added such that the topography could remain perfectly flat throughout, as in the bottommost part of
Figure **8:**. Using the two equations,

$$\text{density}(region) = \frac{1}{size(region)}\left(\sum_{i \in region} tracks[i]\right) \qquad goal\_density = \frac{1}{S_{\max}}\left(unplaced + \sum tracks[]\right)$$

(where *unplaced* is the number of tracks that have not yet been placed, and *goal_density* may be fractional), our function to minimize becomes:

$$abs\left(\frac{1}{size(region)}\left(\left(\sum_{i \in region} tracks[i]\right) + num\_to\_add\right) - \frac{1}{S_{\max}}\left(unplaced + \sum tracks[]\right)\right)$$

But to calculate the actual *num_to_add*, we can instead consider the equation to be:

$$\frac{1}{size(region)}\left(\left(\sum_{i \in region} tracks[i]\right) + num\_to\_add\right) - \frac{1}{S_{\max}}\left(unplaced + \sum tracks[]\right) = value$$

where *value* should be as close to 0 as possible.

Then, by fixing *value* at zero, we can rearrange the equation as such:

$$ideal = \frac{size(region)}{S_{\max}}\left(unplaced + \sum tracks\right) - \sum_{i \in region} tracks[i]$$

where the actual *num_to_add* is either the floor() or ceiling() (whichever yields a lower result in the original equation) of the possibly-fractional *ideal*, as we can only ever add integer quantities of tracks. Also, because *ideal* may be fractional, adding integer numbers of tracks can introduce some error, which can accumulate as we progress. Thus, we choose our starting point so that the widest plain will be considered last, as it can tolerate the most error by amortizing it over its wider area.

```
Density_Based_Placement(U, unplaced, tracks[], breaks[]) {
    Find the widest plain in tracks[]
    If tie, choose the one of these with the widest adjacent mountain
    If tie, choose randomly from the tied plains

    Let root_point be the location at the end of this plain run adjacent to the
        wider mountain
    Initialize blockage to the mountain adjacent to root_point
    Initialize next_plain to the plain adjacent to blockage (the plain that does
           not include root_point, unless blockage is the only mountain in the
           architecture)
    Initialize region to next_plain plus blockage

    While (1) {
        num_to_add = Calculate_Num_Tracks(region, S_max, unplaced, tracks[])
        Assign num_to_add unplaced tracks from U to next_plain, spaced evenly
               with respect to each other as well as the boundaries of next_plain
        Update tracks[], breaks[], and unplaced
        If at root_point or all are placed, end
        Set blockage to the mountain on the other side of next_plain from the
               current blockage
        Set next_plain to the plain on the other side of the new blockage from
               the current next_plain
        Update region to include new blockage and new next_plain
    }
}

Calculate_Num_Tracks(region, S_max, unplaced, tracks[]) {
    Let ideal = size(region)/S_max * ( unplaced + sum_{j=0}^{S_max-1} tracks[j] ) - sum_{i∈region} tracks[i]
    Return either floor(ideal) or ceil(ideal) as num_to_add, whichever yields a
           smaller
        abs( 1/size(region) * ( ( sum_{i∈region} tracks[i] ) + num_to_add ) - 1/S_max * ( unplaced + sum_{j=0}^{S_max-1} tracks[j] ) )
}
```

Figure 9: The density based placement function and the function to calculate the number of tracks to add to a given region in the newest plain. A plain is defined as consecutive positions in *tracks*[] equal to min{*tracks*[]). A mountain is defined as consecutive positions in *tracks*[] not equal to min(*tracks*[]).

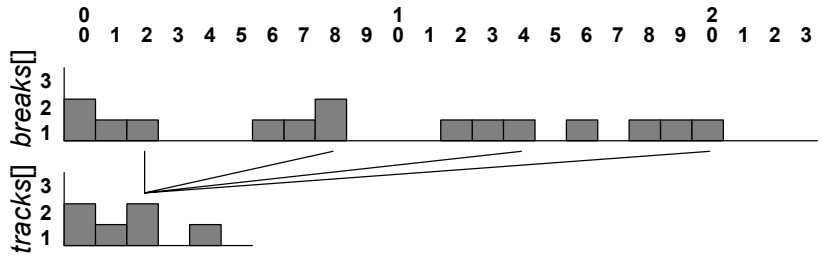


Figure 10: An example of a *breaks*[] array for $K = 24$, and the corresponding *tracks*[] array for $S_{next} = 6$. The lines between the two arrays indicate the locations in the *breaks*[] array used to compute the value of *tracks*[2].

Once we have placed all tracks with S = S_{max}, we need to prepare for the next iteration when we place all tracks with S=S_{next}. We want to take into consideration the effects of all previously placed tracks when placing tracks of length S_{next}. The contents of the *tracks*[] array are therefore in truth more complex than described previously, when we stated that they were the number of tracks with breaks at each of the potential offset positions. We actually use the *tracks*[] array not just to indicate actual break locations, but also to simulate the conversion of all previous track lengths to length S_{next}. This operation is shown in Figure 11. Because the *breaks*[] array can have many more positions than potential offset values for tracks with S = S_{next}, the *tracks*[] array as illustrated by Figure 10, provides somewhat of a "global view" of the *breaks*[] array within the potential offset values 0 to S_{next}-1. It basically records the peaks in the breaks array coinciding with each potential offset value that could be assigned to the S_{next}-length tracks. This translates all of the longer tracks into approximate numbers of S_{next}-length tracks at each offset from 0 to S_{next}-1 so that we can consider the effects of these tracks when placing tracks of length S_{next}.

```
Convert_to_Snext(Snext, tracks[], breaks[]) {
   Replace current tracks[] array with tracks[] array of size Snext
   For i = 0 to i = Snext-1
   tracks[i] = max(breaks[i+Snext*y]) for all y such that i+Snext*y falls within
          the bounds of the breaks[] array
}
```

Figure 11: This function does not create actual placeholder tracks to represent tracks with S > S_{next}, but it does fill the *tracks*[] array in such a way as to simulate all previously placed tracks being converted to segment length S_{next}.

4 Results

We have tested our algorithms, comparing diversity scores of the resulting routing architectures, in a variety of ways. We use a number of terms to describe the area of the search space that we have covered in each of these tests. The value numTracks refers to the total number of tracks in a particular track placement problem, numS refers to the number of discrete S values present in the problem, maxS is the largest S value in the problem, and maxTS is the maximum number of tracks at any one S value in the problem. For these comparisons we have reduced our very large search space by only considering problems where the number of tracks at each S value is less than the S value itself. Figure 4a strongly implies that cases with S or more tracks of a particular S value will yield similar results by placing full sets evenly and reducing them to cases with no more than S-1 tracks at each S value. Cases with just one track, or all track lengths less than 3, are trivial and thus ignored (any solution to these problems is equal to any other). The three terms, numTracks, numS, and maxS, along with the testing policies described above, define the track placement problems we consider in our comparisons.

Our first test was to verify that the Optimal Factor Algorithm yields a best possible diversity score in practice as well as theory. The results of this algorithm were compared to those of the Brute Force Algorithm for all cases with 2≤numTracks≤8, 1≤numS≤4, and 3≤maxS≤9, which represents 5236 different routing problems. Note that even with these restrictions the runtime of brute-force is roughly a week. For example, a single routing problem, with two tracks each of lengths 6, 7, 8, and 9, requires 6 minutes on our fastest machine, a Pentium IV 2GHz. A problem which looks only slightly more difficult, with two tracks each of lengths 7, 8, 9, and 10, requires 1.25 hours on that same machine. In all cases where a solution could be found using the Optimal Factor Algorithm, the resulting diversity score was identical to the Brute Force method. Furthermore, we compared the Relaxed Factor Algorithm to the Optimal Factor Algorithm for this same range and found that Relaxed Factor produces optimal results for all cases that meet the Optimal Factor restrictions within that range.

Next, we compared the performance of the unrestricted heuristics (Relaxed and Simple Spread) to the results of the Brute Force method for the same search space as above to determine the quality of our algorithms. The results for the heuristics, normalized to Brute Force, are shown categorized by numTracks, numS, and maxTS in Figure 12. In this figure, the Brute Force result is a constant value of 1. Figure 12 left indicates that both heuristics achieve optimal results in some cases, with the average of the Relaxed algorithm nearly optimal across the entire range. Simple Spread improves with the increasing number of tracks, and both algorithms degrade with an increase in the number of different S values, though only slightly for Relaxed. Note that the upswing of both algorithms' minimums towards the higher values of numTracks may be an artifact of our benchmark suite – since we only allow at most 4 unique S values, once we have more than 4 tracks we must have at least two tracks with the same S value. This clearly helps Simple Spread's worst case, since in these tests it is optimal for a single S value (Figure 12 center), and thus can take advantage of this

shared S value. We suspect Relaxed may have the same feature, though why Relaxed's min improves only above 6 tracks is unclear.

Both of these observations are confirmed by Figure 12 right, which shows that as the number of tracks per S value increases , the quality of all algorithms improves. The only exception is for the relaxed algorithm when maxTS=1; when there is only one track per S value the relaxed algorithm is always optimal. Throughout the benchmark suite the Relaxed algorithm is clearly superior, in both average and min, to the Simple Spread algorithm. This indicates the critical importance of correlation between S values when performing track placement. Figure 12 center also demonstrates how the greater numS in a particular problem, the more difficult it is to solve well, which we expected to be the case. Note that the results for both heuristics are optimal for the case when there is only one S value, as in this case there are no correlations to contend with, and only an even spreading is required.

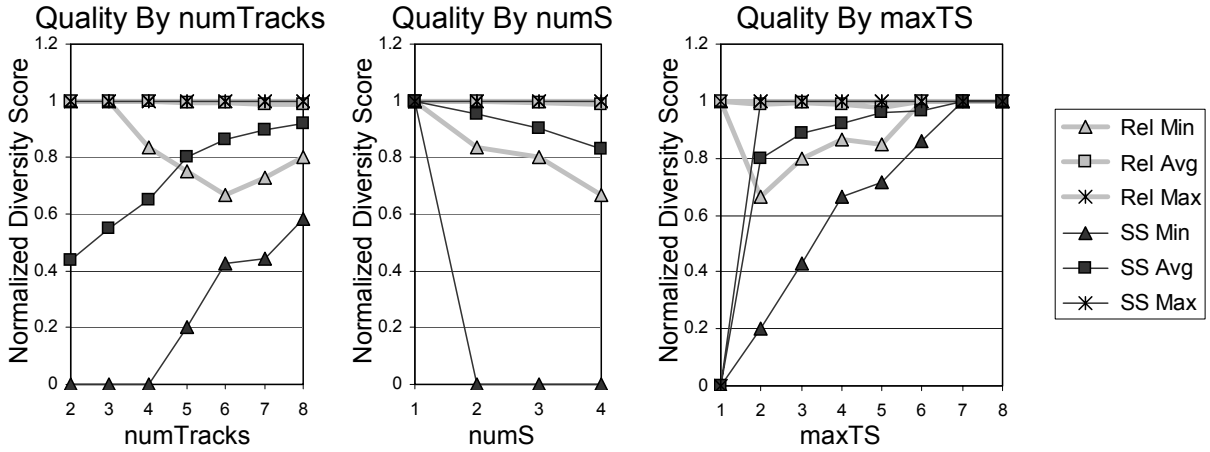


Figure 12: A comparison of the Relaxed and Simple Spread heuristics to the Brute Force diversity scores, with respect to numTracks (left), numS (center), and maxTS (right). The diversity scores for each case were first normalized to the Brute Force result, which is represented by a horizontal line at the value 1.

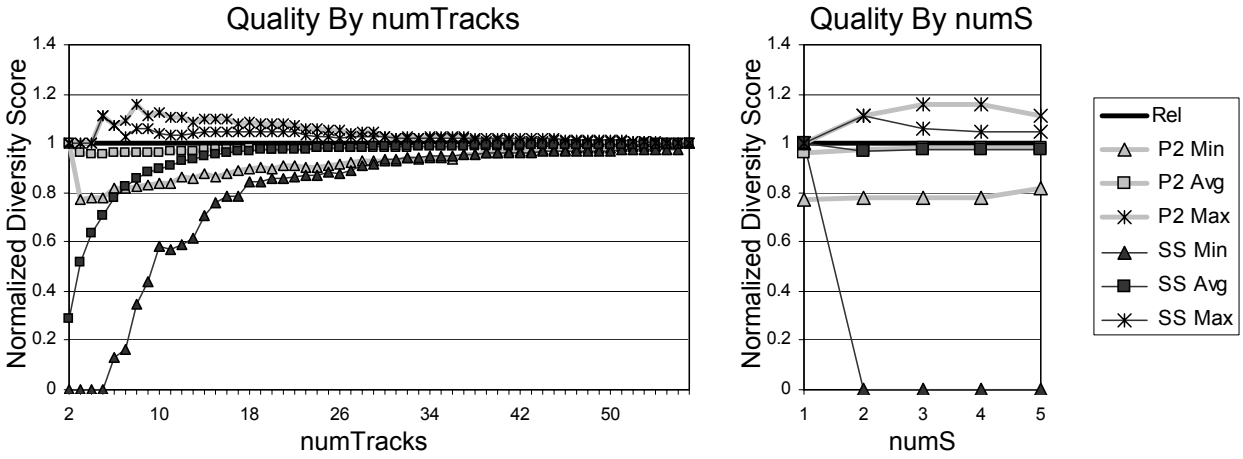


Figure 13: Relaxed Factor, Power2, and Simple Spread comparison for cases with only power-of-two S values. The values for Power2 and Simple Spread were normalized to the Relaxed Factor value.

In order to also compare our Power2 Algorithm to the other heuristics, we ran tests on track sets where all S values were a power of two, up to a maxS of 32. The number of tracks in the tests were not restricted beyond the requirement that for each S value, the number of tracks must be less than that S value. Figure 13 left graphs the minimum, average, and maximum diversity scores (normalized to the Relaxed Factor result) at each number of tracks. We can see that the results of the different algorithms become more similar as the number of tracks increases. The performance of the Power2 algorithm is similar to the Relaxed Factor algorithm, with Power2 performing better than Relaxed Factor in

some cases, and worse in others. Simple Spread performs worse for these tests than the previous because by restricted all S values to powers of two, we have ensured that all tracks are correlated.

Figure 13 right graphs the same data as Figure 13 left, but according to numS. In general, performance is relatively consistent regardless of the number of S values, with the performance of Power2 again close to that of Relaxed Factor. The exception is when there is only one S value, where correlations do not matter. Unlike Relaxed Factor and Simple Spread, however, Power2 is not always optimal in this case because its focus is on properly handling correlations more than even spreading within one S value.

Finally, we have tested our track placement algorithms on a number of existing architectures to determine if their track placements could potentially be improved, given the concepts we have presented. The architectures of RaPiD [Cronquist99], Triptych [Borriello95], Garp [Hauser97], and Chimaera [Hauck97] are evaluated in Figure 14. The quantity of tracks and their lengths are given, followed by the original routing architecture's diversity score, and the results of our algorithms on the architecture's set of tracks. All diversity scores have been normalized to the Brute Force value for each architecture. Note that RaPiD is a special case because it has tracks that do not have uniform track lengths. The diversity score is given for the actual architecture as-is, while the track placement algorithms were executed on two different versions of RaPiD, one with length-3 wires in those tracks, and one with length-4. Garp has multiple entries because the routing resources in the architecture depend on the number of rows of logic units in that architecture. Values are given for Garp channels that span 8, 16, and 32 units.

	Normalized Diversity Score					
	Actual	Brute Force	Optimal	Relaxed	Power2	Simple Spread
Rapid-3 3:4 13:10	1.053	1	N/A	1	N/A	1
Rapid-4 4:4 13:10	1	1	N/A	1	N/A	1
Triptych 4:2 8:2 16:2	0.778	1	N/A	0.944	1	0.778
Garp-8 4:4 8:2 16:1	0.923	1	N/A	1	0.923	0.769
Garp-16 4:4 8:2 16:2 32:1	0.963	1	N/A	0.889	0.963	0.667
Garp-32 4:4 8:2 16:2 32:2 64:1	0.982	1	N/A	0.782	0.982	0.618
Chimaera 3:3 7:7	1	1	1	1	N/A	1

Figure 14: A chart of a number of segmented-channel architectures, their diversity scores, and the diversity scores after track placement using each of our algorithms. All scores have been normalized to the Brute Force result. N/A means the architecture's tracks do not meet the restrictions of a particular algorithm. The quantity and S values of the tracks in each problem are given as S-value:quantity beneath the name of the architecture.

Figure 14 demonstrates that even attempting to determine a good track placement by hand is difficult. The Triptych design was intended to be very regular, but does not account for the correlations between S values, and is essentially the same result as using Simple Spread. Taking correlations into account can significantly improve the diversity score of this architecture. RaPiD's track placement is optimal compared to a near-RaPiD architecture with the local routing tracks only containing wires of length 4, instead of both 3 and 4, and better than the optimal placement of a near-RaPiD architecture where the local tracks are of length 3. All algorithms perform equivalently here, as the two different S values in the problem are relatively prime. All solutions to Chimaera, both calculated and the actual architecture design, are optimal for the same reason. Garp's specified track placement, in all three cases, is very close to optimal.

5 Conclusions

As we have shown, the track placement problem involved fairly subtle choices, including balancing requirements between tracks of the same length, and between tracks of different, but not relatively prime, lengths. We introduced a quality metric, the diversity score, that captures the impact of track placement. Multiple algorithms were presented to solve the track placement problem. One of these algorithms is provably optimal for some situations,

though it is complex and works for only a relatively restricted set of cases. We also developed a relaxed version of the optimal algorithm, which appears to be optimal in all cases meeting the restrictions of the optimal algorithm, and whose average appears near optimal overall. While the other simple heuristics presented here do have problems in some cases, they provide simple methods to obtain good quality results in many cases (particularly the Power2 algorithm).

We envision two situations where the results of this paper can be applied. First, we believe that there is a growing need for automatic generation of FPGA architectures for System-on-a-Chip. By producing domain-specific FPGAs much better area, performance, and power can be achieved than with standard FPGAs. Furthermore, because the FPGA subsystem will be within a custom fabricated SoC, the advantage of pre-made silicon of commodity FPGAs is irrelevant. Second, we have shown that even in hand designed architectures, the track placements achieved by the designers can be improved by carefully considering correlations between track lengths. Some of these techniques can be applied by the FPGA designer to their work to potentially improve overall quality of the resulting architecture.

References

[Borriello95] G. Borriello, C. Ebeling, S. Hauck, S. Burns, "The Triptych FPGA Architecture", IEEE *Transactions on VLSI Systems*, Vol. 3, No. 4, pp. 491-501, December 1995.

[Compton02] K. Compton, S. Hauck, "Flexible Routing Architecture Generation for Domain-Specific Reconfigurable Subsystems", *International Conference on Field-Programmable Logic and Applications*, pp. 59-68, 2002.

[Cronquist99] D. C. Cronquist, P. Franklin, C. Fisher, M. Figueroa, C. Ebeling, "Architecture Design of Reconfigurable Pipelined Datapaths", *Twentieth Anniversary Conference on Advanced Research in VLSI*, 1999.

[Hauck97] S. Hauck, T. W. Fry, M. M. Hosler, J. P. Kao, "The Chimaera Reconfigurable Functional Unit", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 87-96, 1997.

[Hauser97] J. R. Hauser, "The Garp Architecture", *University of California at Berkeley Technical Report*, 1997.

Appendix

Everything here is in **integers** unless stated otherwise.

We assume we have the non-empty set T of tracks, and each track T_i has a segment length $S_i >= 1$ for all wires in that track.

Our job is to find the offset O_i for each track T_i which represents the position of the first break between wires within the track. We also assume we are working with an infinite (in both directions) channel, so that we can ignore edge effects (for now).

Given the above, the breaks on a given track occur at locations $S_i*x + O_i$ forall integers x ($-\infty \ldots \infty$).

Cost Equation Details

The function *uncut*(P,L,S_i,O_i) tells us if a track T_i with segment length S_i and offset O_i has no breaks in the window of length L starting a position P:

$$uncut(P,L,S_i,O_i) = \begin{cases} 0 & \exists x \mid P \le S_i * x + O_i < P + L \\ 1 & otherwise \end{cases}, \text{ where L>0}$$

Extending this to a set G of tracks for a window and position, we get:

$set\_uncut(P,L,G,O) = \sum_{i \in G} Uncut(P,L,G_i,O_i)$, where $G \subseteq T$, and L>0

We define the following equation as the minimum value of $set\_uncut(P,L,G,O)$ for a given L across all possible values of P:

$$min\_set\_uncut(L,G,O) = \min_{P=-\infty}^{\infty}(Set\_uncut(P,L,G,O)) = \min_{P=-\infty}^{\infty}\left(\sum_{i \in G} Uncut(P,L,G_i,O_i)\right),$$

where L>0 and $G \subseteq S$

Our goal is to find O_i that maximizes the number of tracks that can be used to route a signal between any two points in the array. We will assume that each source and sink is equally likely. Thus, we are picking the O_i values to maximize

$$Average_{L=1}^{\infty}\left(min\_set\_uncut\left(L,T,O\right)\right)$$

It is equivalent to maximize:

$$\sum_{L=1}^{\infty} min\_set\_uncut(L,T,O)$$

Since this value will be 0 whenever L is greater than or equal to the maximum segment length S_{max} in set T (as will be proven later), this equation is equal to:

$$\sum_{L=1}^{S\max-1} min\_set\_uncut(L,T,O),$$

which is calculation for the *density_score* function used in the body of this paper.

Theorems and Proofs

The theorems from the preceding paper, together with additional theorems necessary for the proofs, are presented in this section. The theorems in this appendix are worded with more detail than appears in the preceding paper, but the general idea of correspondingly-numbered theorems remains the same. A proof is also given for each theorem.

Theorem 1: For all assignments of O_i, $min\_set\_uncut(L,T,O) \le floor\left(|T| - \sum_{i=1}^{|T|} \min(1, L/S_i)\right)$

Proof:

Let $k = \prod_{i=1}^{|T|} S_i$. If we consider the range of positions 1 to k, we know that it has the same pattern as k+1 to 2k, etc. In other words, if there is a track with a break at position Z, 1<=Z<=k, then there is another break on that track at position Z+k. This is because $Z = S_i*x+O_i$, and $Z+k = S_i*x + O_i + k = S_i*(x + k/S_i) + O_i$, and because we know k/S_i is an integer. This also means that we can essentially create a wrap-around "ring channel", where the position just beyond k is 1, and *uncut*, *set_uncut*, and *min_set_uncut* will have the same values in this ring.

Take the ring channel of positions 1..k. In this structure there are k unique *set_uncut*(P,L,S,O) "windows" that need to be considered for *min_set_uncut*. Consider a given track T_i. It will have k/S_i breaks in the region 1..k. Each break will appear in L windows of length L. If S_i<=L then all *uncut*(P,L,S_i,O_i) are 0 because each track must have at least one break in that window, given that all wire lengths are shorter than or equal to the window size. Otherwise, no two breaks from T_i will appear in the same window because the window is shorter than T_i's wire length. For this case

$$ave_{P=-\infty}^{\infty}\left(uncut\left(P,L,S_i,O_i\right)\right) = 1 - (\#ofWindowsIisIn)/(\#Windows) = 1 - \left(L*k/S_i\right)/\left(k\right) = 1 - L/S_i$$

Given L/S_i >=1 for S_i <= L, we get the unified formula:

$$ave_{P=-\infty}^{\infty}\left(uncut\left(P,L,S_i,O_i\right)\right) = 1 - \min\left(1, L/S_i\right)$$

We can combine the averages across all tracks T:

$$ave_{P=-\infty}^{\infty}\left(set\_uncut\left(P,L,T,O\right)\right) = \sum_{i=1}^{|T|} 1 - \min\left(1, L/S_i\right) = |T| - \sum_{i=1}^{|T|} \min\left(1, L/S_i\right)$$

If this is the average, and *set_uncut*(P,L,T,O) is an integer, then the best-case for

$$min\_set\_uncut(L,T,O) = \min_{P=-\infty}^{\infty}\left(set\_uncut(P,L,T,O)\right)$$

$$= floor\left(ave_{P=-\infty}^{\infty}(set\_uncut(P,L,T,O)\right) = floor\left(|T| - \sum_{i=1}^{|T|} \min\left(1, L/S_i\right)\right)$$

This means that for any assignment of O_i's,

$$min\_set\_uncut(L,T,O) \le floor\left(|T| - \sum_{i=1}^{|T|} \min\left(1, L/S_i\right)\right)$$

Note that this is a loose lower bound, since for S={2,3}, *min_set_uncut*(1,T,O) = 0, while the bounds says it is <=floor(2-(1/2+1/3)) = floor(1+1/6) = 1. This is because 2 & 3 are relatively prime. However, if we combine our bound above with Theorem 2, we get

min_set_uncut(1,{2,3},{0,0}) = *min_set_uncut*(0,{2},{0,0}) + *min_set_uncut*(0,{3},{0,0})

<= floor(1-1/2) + floor(1-1/3) = 0+0 = 0, which is tight for this example.

Theorem 2: If T can be split into two sets G1⊂T and G2 = T-G1, where all elements of G1 are relatively prime with all elements of G2, then *min_set_uncut*(L,T,O) = *min_set_uncut*(L,G1,O1) + *min_set_uncut*(L,G2,O2), where O is the combination of solution sets O1 and O2. This means that you can separately solve the two independent problems. This can also be applied recursively.

Proof:

Similarly to what we did for Theorem 1, let
$k1 = \prod_{i \in G1} S_i$ and $k2 = \prod_{i \in G2} S_i$

Because the segment lengths in G1 are all relatively prime to the values in G2, k1 is relatively prime to k2. This also means that, for any delta D, there are integers Y and Z such that D=k1*Y - k2*Z (the Extended Euclid Algorithm can find Y and Z for 1=k1*Y - k2*Z. Multiply Y and Z by D, and you solve this problem).

As discussed above in Theorem 1, the pattern of breaks for set G1 in the region 1..k1 will repeat in (k1+1)..2*k1, etc. Thus, Forall integers X, P, and L, *set_uncut*(P,L,G1,O1) = *set_uncut*(P+k1*X,L,G1,O1)

Similarly, *set_uncut*(P,L,G2,O2) = *set_uncut*(P+k2*X,L,G2,O2).
Forall L, let $P1_L$ be a value for which $set\_uncut(P1_L, L, G1, O1) = \min_{P=-\infty}^{\infty}(set\_uncut(P, L, G1, O1))$
and $P2_L$ be a value for which $set\_uncut(P2_L, L, G2, O2) = \min_{P=-\infty}^{\infty}(set\_uncut(P, L, G2, O2))$

Find Y and Z such that ($P2_L$-$P1_L$) = (k1*Y - k2*Z). Swapping terms yields ($P2_L$+k2*Z) = (k1*Y+$P1_L$)

We know *set_uncut*($P1_L$,L,G1,O1) = *set_uncut*($P1_L$+K1*Y,L,G1,O1), and *set_uncut*($P2_L$, L, G2,O2) = *set_uncut*($P2_L$+k2*Z,L,G2,O2) = *set_uncut*($P1_L$+k1*Y,L,G2,O2).

Thus, at the point k1*Y + $P1_L$ we hit a point with the minimum of uncut wires for both G1 and G2. Let Q = k1*Y + $P1_L$

The definition of *set_uncut* is additive, such that:
set_uncut(Q,L,T,O) = *set_uncut*(Q,L,G1,O1) + *set_uncut*(Q,L,T-G1,O-O1)
= s*et_uncut*(Q,L,G1,O1)+*set_uncut*(Q,L,G2,O2)

Given that:
$set\_uncut(Q, L, G1, O1) = \min_{P=-\infty}^{\infty}(set\_uncut(P, L, G1, O1)) = min\_set\_uncut(L, G1, O1)$ and
$set\_uncut(Q, L, G2, O2) = \min_{P=-\infty}^{\infty}(set\_uncut(P, L, G2, O2)) = min\_set\_uncut(L, G2, O2)$,

we get:
$$min\_set\_uncut(L, T, O) = \min_{P=-\infty}^{\infty}(set\_uncut(P, L, T, O)) = \min_{P=-\infty}^{\infty}(\sum_{i \in G} uncut(P, L, S_i, O_i))$$
$$\geq \min_{P1=-\infty}^{\infty}(\sum_{i \in G1} uncut(P1, L, S_i, O_i)) + \min_{P2=-\infty}^{\infty}(\sum_{j \in (T-G1)} uncut(P2, L, S_j, O_j))$$
$$= \min_{P1=-\infty}^{\infty}(\sum_{i \in G1} uncut(P1, L, S_i, O_i)) + \min_{P2=-\infty}^{\infty}(\sum_{j \in G2} uncut(P2, L, S_j, O_j))$$
$$= \min_{P1=-\infty}^{\infty}\left(set\_uncut(P1, L, G1, O1)\right) + \min_{P2=-\infty}^{\infty}\left(set\_uncut(P2, L, G2, O2)\right)$$
$$= min\_set\_uncut(L, G1, O1) + min\_set\_uncut(L, G2, O2)$$

and
$$min\_set\_uncut(L, T, O) = \min_{P=-\infty}^{\infty}(set\_uncut(P, L, T, O))$$
$$\leq set\_uncut(Q, L, T, O) = set\_uncut(Q, L, G1, O1) + set\_uncut(Q, L, G2, O2)$$
$$= min\_set\_uncut(L, G1, O1) + min\_set\_uncut(L, G2, O2)$$
Thus, *min_set_uncut*(L,T,O) = *min_set_uncut*(L,G1,O1) + *min_set_uncut*(L,G2,O2).

Theorem 3: If all elements of S are identical, then an optimal solution, meeting the bound of Theorem 1, is Forall i 1..|T|, O_i = floor(S_1*(i-1)/|T|).

Proof:

First, *min_set*_uncut(L,T,O) = 0 for all L >= S_1, since a track of length S_1 always has a cut in a window of length S_1. Thus, all assignments to O_i are identical, and optimal, for lengths where L >= S_1. We now consider situations where L<S_1.

For *set_uncut*(P1,L,T,O), a track will be cut if we can find an integer X such that P1 <= S_i*X + O_i < P1+L

First, we convert the < operator on the right side to a <= operator: P1 <= S_i*X + O_i <= P1+L-1

Substituting in for O_i we get P1 <= S_i*X + floor(S_1*(i-1)/|T|) <= P1+L-1

Since everything except the value computed inside the floor is an integer, this means P1<=S_i*X+S_1*(i-1)/|T|<P1+L

All S_i's are identical, so P1 <= S_1*X + S_1*(i-1)/|T| < P1+L

In this version, the various track breaks are uniformly spaced out by a distance of S_1/|T|. In such a system, if there is a case where there is a break that falls exactly on an integer Z (that is, the real value falls onto an integer Z), then a region with the most breaks will be Z <= S_1*X + S_1*(i-1)/|T| < Z+L

The reason for this is that any other region that also has a break fall on an integer beginning point will be the same (since the pattern repeats), and one that doesn't have a break at its low end will either have the same number of total breaks, or one break less, since the break closest to P1+L may have slid off beyond this region.

For our computation, a break of T_1 falls at location 0.0 in the real space, so a region with the maximum number of breaks starts at 0.

If L>=S_1 then all tracks are cut. Otherwise, the tracks from 1 to i are cut, where i is the largest integer such that S_1*(i-1)/|T|<L. Solving for i:
S_1*(i-1)/|T|<L
(i-1)<L*|T|/S_1
i<1+L*|T|/S_1

Since we want the largest such integer i, we have:
i<ceil(1+L*|T|/S_1)
i=ceil(1+L*|T|/S_1)-1
i=ceil(L*|T|/S_1)

This will be the largest number of elements cut in a region of length L, and thus we get:
min_set_uncut(L,T,O)= {0 if L>=S_1, else |T|-ceil(L*|T|/S_1) }

Theorem 1 states $min\_set\_uncut(L,T,O) \le floor\left(|T| - \sum_{i=1}^{|T|} \min(1, L/S_i)\right)$

For the case where L>=S_1, which means L>=S_i for all i, this formula reduces to *min_set_uncut*(L,T,O) <= 0, and our assignment to O_i's is optimal (as is any). For the case where L<S_1, we need to show that

$$|T| - ceil(L^*|T|/S_1) = floor\left(|T| - \sum_{i=1}^{|T|} \min(1, L/S_i)\right)$$

$$|T| - ceil\left(\sum_{i=1}^{|T|} L/S_1\right) = |T| + floor\left(-\sum_{i=1}^{|T|} \min(1, L/S_i)\right)$$

$$-ceil\left(\sum_{i=1}^{|T|} L/S_1\right) = floor\left(-\sum_{i=1}^{|T|} L/S_i\right)$$

$$Let \quad Y = \sum_{i=1}^{|T|} L/S_1 \quad then \quad -ceil(Y) = floor(-Y)$$

This statement is true, so our formulation for O_i's gives the same results as our lower bound, and is thus optimal.

Theorem 4: Given a set G⊂T where G_j=|G|. If there exists a solution O' for tracks T'=T-G such that forall L $min\_set\_uncut(L,T',O') = floor\left(|T'| - \sum_{i=1}^{|T'|} \min(1, L/T'_i)\right)$, then there is a solution O for tracks T such that $min\_set\_uncut(L,T,O) = floor\left(|T| - \sum_{i=1}^{|T|} \min(1, L/T_i)\right)$. This solution is created by having all elements of T' keep their value O', while for all elements G_j of G, their O is set to j, their position in set G.

This means that if there are N or more tracks of segmentation length N, remove them from the problem and solve the rest optimally. If you succeed, we can then solve the overall problem by adding back the N tracks of length N, each with a unique O from 1..N

Proof:

Renumber set T such that the first |G| elements $T_1 \ldots T_{|G|}$=|G|. We are given that:

$$min\_set\_uncut(L,T',O') = floor\left(|T'| - \sum_{i=1}^{|T'|} \min(1, L/S_i)\right) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T',O')).$$

For elements $G_1 \ldots G_{|G|}$ we are setting their O's to their position in G. We can call this assignment O_G. This means that, for set G, there is exactly one track broken at each position. Thus, we get forall P

$$set\_uncut(P,L,G,O_G)) = \begin{Bmatrix} 0 & L >= |G| \\ |G| - L & otherwise \end{Bmatrix}$$

This can be restated as $set\_uncut(P,L,G,O_G)) = |G| - \min(|G|, L)$

By definition,

$$min\_set\_uncut(L,T,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T,O))$$
$$= \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T',O') + set\_uncut(P,L,G,O_G))$$

As discussed above, *set_uncut* of G is constant for all P for a given L, so

$$min\_set\_uncut(L,T,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T',O')) + set\_uncut(P,L,G,O_G)$$
$$= floor\left(|T'| - \sum_{i=1}^{|T'|} \min(1, L/T_i)\right) + (|G| - \min(|G|, L))$$
$$= floor\left(|T'| + |G| - \left(\sum_{i=1}^{|T'|} \min(1, L/T_i)\right) - \min(|G|, L)\right)$$
$$= floor\left(|T| - \left(\sum_{i=1}^{|T'|} \min(1, L/T_i)\right) - \min(|G|, L)\right)$$
$$= floor\left(|T| - \left(\sum_{i=1}^{|T'|} \min(1, L/T_i)\right) - |G| * \min(1, L/|G|)\right)$$
$$= floor\left(|T| - \left(\sum_{i=1}^{|T'|} \min(1, L/T_i)\right) - \left(\sum_{j=1}^{|G|} \min(1, L/|G|)\right)\right)$$
$$= floor\left(|T| - \left(\sum_{i=1}^{|T'|} \min(1, L/T_i)\right) - \left(\sum_{j=1}^{|G|} \min(1, L/G_j)\right)\right)$$
$$= floor\left(|T| - \left(\sum_{i=1}^{|T|} \min(1, L/T_i)\right)\right)$$

Theorem 4a: Given a set G⊂T where all S_j=|G|. If there exists a solution O' for tracks T'=T-G such that forall L $min\_set\_uncut(L,T',O') - bound(L,T') = delta_L$, where bound(L,T') is the bound from Theorem 1, then there is a solution O for tracks T such that $min\_set\_uncut(L,T,O) - bound(L,T) = delta_L$. This solution is created by having all elements of T' keep their value O', while for all elements G_j of G, their O is set to j, their position in set G.

This means that if there are N or more tracks of segmentation length N, remove the full set of N tracks from the problem and solve the rest as efficiently as possible. We can then solve the overall problem by adding back the N tracks of length N, each with a unique O from 1..N, and the only sub-optimality will be that introduced by the solution to the sub-problem. Note that if you solve the sub-problem optimally, you will also solve the overall problem optimally.

Proof:

Renumber set T such that the first |G| elements $T_1 \ldots T_{|G|}$=|G|. We are given that:
$min\_set\_uncut(L,T',O') = floor\left(|T'| - \sum_{i=1}^{|T'|} \min(1, L/S_i)\right) + delta_L = \min_{P=-\infty}^{\infty}(Set\_uncut(P,L,T',O'))$.

For elements $G_1 \ldots G_{|G|}$ we are setting their O's to their position in G. We can call this assignment O_G. This means that, for set G, there is exactly one track broken at each position. Thus, we get forall P,

$$set\_uncut(P,L,G,O_G)) = \begin{Bmatrix} 0 & L >= |G| \\ |G| - L & otherwise \end{Bmatrix}$$

This can be restated as $set\_uncut(P,L,G,O_G)) = |G| - \min(|G|, L)$

By definition,
$min\_set\_uncut(L,T,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T,O))$
$= \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T',O') + set\_uncut(P,L,G,O_G))$

As discussed above, *set_uncut* of G is constant for all P for a given L, so
$min\_set\_uncut(L,T,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T',O')) + set\_uncut(P,L,G,O_G)$
$= floor\left(|T'| - \sum_{i=1}^{|T'|} \min(1, L/T_i)\right) + delta_L + (|G| - \min(|G|, L))$
$= floor\left(|T'| + |G| - \left(\sum_{i=1}^{|T'|} \min(1, L/T_i)\right) + delta_L - \min(|G|, L)\right)$
$= floor\left(|T| - \left(\sum_{i=1}^{|T'|} \min(1, L/T_i)\right) + delta_L - \min(|G|, L)\right)$
$= floor\left(|T| - \left(\sum_{i=1}^{|T'|} \min(1, L/T_i)\right) + delta_L - |G| * \min(1, L/|G|)\right)$
$= floor\left(|T| - \left(\sum_{i=1}^{|T'|} \min(1, L/T_i)\right) + delta_L - \left(\sum_{j=1}^{|G|} \min(1, L/|G|)\right)\right)$
$= floor\left(|T| - \left(\sum_{i=1}^{|T'|} \min(1, L/T_i)\right) + delta_L - \left(\sum_{j=1}^{|G|} \min(1, L/G_j)\right)\right)$
$= floor\left(|T| - \left(\sum_{i=1}^{|T|} \min(1, L/T_i)\right)\right) + delta_L$

Theorem 5: Convert all S_i to the form $X_i * F^{N(F,i)}$, where F is a prime number, N(F,i)>=0, and X_i is relatively prime to F. Note that many N(F,i) will be 0. If there exists a single track T_j whose N(F,j) is greater than all N(F,i) of all other tracks, i≠j, then forall O $min\_set\_uncut(L,T,O) = min\_set\_uncut(L,T',O)$ where T' is identical to T, except T'_j has $S'_j = S_j/F$.

This means that if there is a track with a unique prime factor, or more of a prime factor than any other, you can simplify things by removing this factor. By iteratively applying this rule you end up with a set of signals such that all remaining prime factors are shared.

Note also that this theorem will help get a tighter lower bound for a given problem, since by removing prime factors you typically reduce the maximum value allowed by Theorem 1.

Proof:

Let $K = \left(\prod_P P^{\max_i(N(P,i))} \right) / F$

for all primes P that are a factor of at least one segment length in set S. K is now a number that is a multiple of all S_i where i≠j, and is a multiple of S_j/F, but is not a multiple of S_j. Also, K contains exactly one less factor of F than S_j contains.

Let P1 be a value where $set\_uncut(P1,L,T',O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T',O))$
Forall i, 1<= i <=|T| and i≠j, *uncut*(P1,L,T_i',O) = *uncut*(P1,L,T_i,O) since $T_i' = T_i$ for i≠j.

If *uncut*(P1, L, T_j',O) = *uncut*(P1, L, T_j,O) then it is clear that:
$\min_{P=-\infty}^{\infty}(set\_uncut(P,L,T',O)) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T,O))$ and
$min\_set\_uncut(L,T,O) = min\_set\_uncut(L,T',O)$.

Since *uncut*(…)∈{0,1}, then if *uncut*(P1,L,T_j',O) ≠ *uncut*(P1,L,T_j,O) then *uncut*(P1,L,T_j',O) = 0 and *uncut*(P1,L,T_j,O) = 1, or *uncut*(P1,L,T_j',O) = 1 and *uncut*(P1,L,T_j,O) = 0

From the definition of *uncut*(…), if *uncut*(P1,L,T_j,O) = 0 then there exists an *x* such that P1 <= S_j*x + O_j < P1+L. This also means that P1 <= (S_j/F) * (x*F) + O_j < P1+L, which is the same as P1<= S'_j*(y) + O_j < P1+L where y = x*F. Thus, *uncut*(P1,L,T_j,O) = 0 implies *uncut*(P1,L,T_j',O) = 0. This eliminates from consideration the case of *uncut*(P1,L,T_j',O) = 1 and *uncut*(P1,L,T_j,O) = 0, since it cannot occur. This also means *min_set_uncut*(L,T,O) >= *min_set_uncut*(L,T',O)

Because of the previous, the only situation we still need to consider is if *uncut*(P1,L,T_j',O) = 0 and *uncut*(P1,L,T_j,O) = 1

Since *uncut*(P1,L,T_j',O) = 0, we know P1<= S'_j*y + O_j < P1+L for some y. Let P2 = S'_j*y + O_j

For an integer z, P2 + K*z = S'_j*y + K*z + O_j= (S_j/F)*y + K*z + O_j

K is a multiple of S_j/F, so let integer K1 = K/(S_j/F).

(S_j/F)*y + K*z + O_j = (S_j/F)*y + (S_j/F)*K1*z + O_j = (S_j/F)*(y + K1*z) + O_j

Recall that K has exactly one less factor of F than S_j has. Thus K1 = K / (S_j/F) = (K*F) / (S_j), and K1 does not have a factor of F. Thus, K1 is relatively prime to F. This means that there exists two integers z1 and z2 such that K1*z1 + y = F*z2. (Extended Euclid Algorithm)

By substitution the formula $P2 + K*z = (S_j/F) * (y + K1*z) + O_j$ becomes $P2 + K*z1 = (S_j/F) * (F*z2) + O_j = S_j*z2 + O_j$

Since $P1 <= P2 < P1+L$, $P1 + K*z1 <= P2 + K*z2 < P1 + K*z2 + L$.

Substituting into this gets $P1+K*z1 <= S_j*z2+O_j < P1 + K*z2 + L$, and thus:
$uncut(P1 + K*z1,L,T_j,O) = 0 = uncut(P1,L,T_j',O)$

Forall $i \neq j$ the truth of the statement $P1 <= S_i*x1_i + O_i < P1+L$ for some integer $x1_i$ is identical to $P1 + K*z2 <= S_i*x1_i + K*z2 + O_i < P1 + K*z2 + L$ which is identical to
$P1 + K*z2 <= S_i*(x1_i + (K/S_i)*z2)+O_i < P1 + K*z2 + L$.

Since K is a multiple of S_i (recall $i \neq j$), and $x1_i$ and z2 are integers, then $x1_i + (K/S_i)*z2$ is an integer. Thus, the truth of the statement $P1 <= S_i*x1_i + O_i < P1+L$ for some integer $x1_i$ is identical to
$P1 + K*z2 <= S_i*(x2_i) + O_i < P1 + K*z2 + L$ for some integer $x2_i$. This means that for all $i \neq j$ $uncut(P1,L,T_i',O) = uncut(P1+K*z2,L,T_i',O) = uncut(P1+K*z2,L,T_i,O)$.

Given all this, we know that

$$min\_set\_uncut(L,T',O) = set\_uncut(P1,L,T',O) = \sum_{i \in T'} uncut(P1,L,T_i,O_i)$$

$$= \sum_{i \in T} uncut(P1 + K*z2,L,T_i,O_i) = set\_uncut(P1 + K*z2,L,T,O)$$

$$min\_set\_uncut(L,T,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T,O)) \leq set\_uncut(P1 + K*z2,L,T,O)$$

$$min\_set\_uncut(L,T,O) \leq min\_set\_uncut(L,T',O)$$

We know from previously that $min\_set\_uncut(L,T,O) >= min\_set\_uncut(L,T',O)$, thus $min\_set\_uncut(L,T,O) = min\_set\_uncut(L,T',O)$

Optimal Algorithm for |T|=2:

If $S_1 \neq S_2$ then there exists a prime factor that appear more times in S_i than S_{3-i}. Based on Theorem 5, replace S_i with S_i/F, and restart the algorithm.

We now have $S_1=S_2$

Set O_i=floor(S_1*(i-1)/|T|), which we know is optimal from Theorem 3.

Theorem 6: If there exists an $S_j = 1$, then *min_set_uncut*(L,T,O) = *min_set_uncut*(L,T-T_j,O-O_j) for all L and O_j.

Some of our algorithms involve removing prime factors from S_i's, which may result in an S_j=1. This theorem thus allows us to eliminate these tracks and simplify the problem.

Proof:

$uncut(P,L,T_j,O_j)$ = {0 if there exists an x such that $P <= S_j*x + O_j < P+L$, 1 otherwise}

When $S_j = 1$, $uncut(P,L,T_j,O_j)$ = {0 if there exists an x such that $P <= x+O_j < P+L$, 1 otherwise}

It is always true that $P <= y+O_j < P+L$ for $y = (P-O_j)$ since $L > 0$, and y is an integer since P and O_j are integers. Thus, for all L and O_j, $uncut(P,L,T_j,O_j) = 0$.

$$min\_set\_uncut(L,T,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T,O)) = \min_{P=-\infty}^{\infty}(\sum_{i\in T} uncut(P,L,T_i,O_i))$$
$$= \min_{P=-\infty}^{\infty}(uncut(P,L,T_j,O_j) + \sum_{i\in T-T_j} uncut(P,L,T_i,O_i)) = \min_{P=-\infty}^{\infty}(\sum_{i\in T-T_j} uncut(P,L,T_i,O_i))$$
$$= min\_set\_uncut(L,T-T_j,O-O_j)$$

Lemma 1: For a given solution O and an integer i, there exists a solution O' with $O_j = O_j'$ for j≠i, and 0 <= O_i'< S_i, such that forall L, *min_set_uncut*(L,S,O) = *min_set_uncut*(L,S,O')

By iteratively applying this rule, we can convert any solution to a solution with O_k' such that 0 <= O_k' < S_k. This means that we only have to consider solutions with 0 <= O_k' < S_k for all k, since all possible solutions have an equivalent solution in this set.

Proof:

Let $O_i' = O_i \bmod S_i$. It is obvious that in this case $0 <= O_i' < S_i$. Also, this means that for $C=O_i-O_i'$, C is divisible by S_i.

By definition we also know that
uncut(P,L,S_i,O_i') = {0 if there exists an *x* such that P <= S_i*x + O_i' < P+L, 1 otherwise}

Let $y = x-C/S_i$, which we know is an integer since C is divisible by S_i. Then $x = y+C/S_i$, and for every integer x there is an integer y and vice-versa.

uncut(P,L,S_i,O_i')={0 if there exists a y such that P <= S_i*(y+C/S_i) + O_i' < P+L, 1 otherwise}
uncut(P,L,S_i,O_i')={0 if there exists a y such that P <= S_i*y+(C + O_i') < P+L, 1 otherwise}
uncut(P,L,S_i,O'_i)={0 if there exists a y such that P <= S_i*y+O_i < P+L, 1 otherwise}
= *uncut*(P,L,S_i,O_i)

By definition we know that
$$min\_set\_uncut(L,S,O') = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,S,O'))$$
$$= \min_{P=-\infty}^{\infty}(\sum_{j\in S} uncut(P,L,S_j,O'_j))$$
$$= \min_{P=-\infty}^{\infty}\left(uncut(P,L,S_i,O'_i) + \sum_{j\in S-S_i} uncut(P,L,S_j,O'_j)\right)$$
$$= \min_{P=-\infty}^{\infty}\left(uncut(P,L,S_i,O'_i) + \sum_{j\in S-S_i} uncut(P,L,S_j,O_j)\right)$$
$$= \min_{P=-\infty}^{\infty}\left(uncut(P,L,S_i,O_i) + \sum_{j\in S-S_i} uncut(P,L,S_j,O_j)\right)$$
$$= \min_{P=-\infty}^{\infty}(\sum_{j\in S} uncut(P,L,S_j,O_j)) = min\_set\_uncut(L,S,O)$$

Lemma 2: For a given solution O, and integers i and c, there exists a solution O' with O_i'= c such that forall L *min_set_uncut*(L,S,O) = *min_set_uncut*(L,S,O')

This means that you can assign the O_i value of any one track to any integer desired and still find an optimal solution, since for every possible solution there is another equally good solution with the given track having an O_i with that value. So, when we are solving a specific example, we can pick the most complex line and assign it to position 1 for example, and then solve the rest of the problem.

Proof:

Let delta = c - O_i. Given a solution O, construct a new solution O', where O_i' = O_j + delta. We know that O_i' = O_i + (c – O_i) = c.

uncut(P,L,S_k,O_k) = {0 if there exists an x such that P<= S_k*x + O_k < P + L, 1 otherwise}
= {0 if there exists an x such that P+delta <= S_k*x + O_k + delta < P + delta + L, 1 otherwise}
= {0 if there exists an x such that P+delta <= S_k*x+O_k' <P+delta+L, 1 otherwise}
= *uncut*(P+delta, L, S_k, O_k').

By definition we know that

$$min\_set\_uncut(L,S,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,S,O))$$
$$= \min_{P=-\infty}^{\infty}(\sum_{j\in T} uncut(P,L,S_j,O_j))$$
$$= \min_{P=-\infty}^{\infty}(\sum_{j\in T} uncut(P+delta,L,S_j,O'_j))$$
$$= \min_{Q-delta=-\infty}^{\infty}(\sum_{j\in T} uncut(Q,L,S_j,O'_j))$$
$$= \min_{Q=-\infty}^{\infty}(\sum_{j\in T} uncut(Q,L,S_j,O'_j))$$
$$= min\_set\_uncut(L,S,O')$$

Lemma 3: A break must appear on track T_i in any region P to P+L-1 inclusive if S_i <= L

Proof:

Let y be the largest value such that S_i*y+O_i < P.

P > S_i*y+O_i
P+S_i > S_i*y+S_i+O_i
P+S_i > S_i*(y+1)+O_i

Since S_i ≤ L, then P+S_i <= P+L

Thus:
S_i*(y+1)+O_i < P+S_i <= P+L
S_i*(y+1)+O_i < P+L
S_i*(y+1)+O_i <= P+L-1

We also know that P <= S_i*(y+1)+O_i, or y wouldn't have been the largest value such that S_i*y+O_i < P.

Thus, P <= S_i*(y+1)+O_i <= P+L-1.

Lemma 4: If there is a track T_i with S_i <= L then *set_uncut*(P,L,S,O) = *set_uncut*(P,L,S-S_i,O-O_i)

Proof:

From Lemma 3 we know that track T_i must have a break in the region P to P+L-1 inclusive, so by definition, *uncut*(P,L,S_i,O_i) = 0.

By definition:

$$set\_uncut(P,L,S,O) = \sum_{j \in S} uncut(P,L,S_j,O_j)$$
$$= uncut(P,L,S_i,O_i) + \sum_{j \in S-S_i} uncut(P,L,S_j,O_j)$$
$$= 0 + \sum_{j \in S-S_i} uncut(P,L,S_j,O_j)$$
$$= \sum_{j \in S-S_i} uncut(P,L,S_j,O_j)$$
$$= set\_uncut(P,L,S-S_i,O-O_i)$$

Corollary Lemma 4a: *set_uncut*(P,L,S,O) = *set_uncut*(P,L,S',O') where S' and O' are S and O respectively with all tracks T_i eliminated where S_i<=L.

Proof:

Recursive application of Lemma 4 can be used to remove all tracks with S_i<=L.

Corollary Lemma 4b: *min_set_uncut*(L,S,O) = *min_set_uncut*(L,S',O') where S' and O' are S and O respectively with all tracks T_i eliminated where S_i<=L.

Proof:

By definition, and from Lemma 4a, we know that
$$min\_set\_uncut(L,S,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,S,O))$$
$$= \min_{P=-\infty}^{\infty}(set\_uncut(P,L,S',O')) = min\_set\_uncut(L,S',O')$$

Corollary Lemma 4c: *min_set_uncut*(L,S,O) = *min_set_uncut*(L,S',O') where S' and O' are S and O respectively with all tracks T_i eliminated where S_i<=L_{min} and L_{min} <= L.

Proof:

Since S_i<=L_{min}<= L, then S_i<= L.

We can use recursive application of Lemma 4 to show that
set_uncut(P,L,S,O) = *set_uncut*(P,L,S',O').

By definition we know that
$$min\_set\_uncut(L,S,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,S,O))$$
$$= \min_{P=-\infty}^{\infty}(set\_uncut(P,L,S',O')) = min\_set\_uncut(L,S',O')$$

Lemma 5: *min_set_uncut*(L,S,O) = *min_set_uncut*(L,permute(S),permute(O)), where permute(G) = any specific permutation of the elements of G.

Proof:

The definitions of *min_set_uncut*, *set_uncut*, and *uncut* are insensitive to the order of their elements.

Lemma 6: If track S_i has a break at position Y, then *min_set_uncut*(L,S,O) = *min_set_uncut*(L,S,O') where $O'_j = O_j$ for $i \neq j$, and $O'_i = Y$.

Proof:

Since S_i has a break at position Y, we know that $Y = S_i*Z + O_i$ for some integer Z.

We also know that forall x, $S_i*x + Y = S_i*x + S_i*Z + O_i = S_i*(x+Z) + O_i = S_i*w + O_i$, where $w = x+Z$.

Thus, since there exists a w such that $B = S_i*w + O_i$, then there exists an $x=w-Z$ such that $B = S_i*x + Y$.

By definition

$$uncut(P,L,S_i,O_i) = \begin{Bmatrix} 0 & \exists x \mid P \le S_i * w + O_i < P + L \\ 1 & otherwise \end{Bmatrix}$$

$$= \begin{Bmatrix} 0 & \exists x \mid P \le S_i * x + Y < P + L \\ 1 & otherwise \end{Bmatrix}$$

$$= uncut(P,L,S_i,O'_i)$$

By definition

$$min\_set\_uncut(L,S,O) = \min_{P=-\infty}^{\infty}\left(\sum_{j\in S} uncut(P,L,S_j,O_j)\right)$$

$$= \min_{P=-\infty}^{\infty}\left(uncut(P,L,S_i,O_i) + \sum_{j\in S-S_i} uncut(P,L,S_j,O_j)\right)$$

$$= \min_{P=-\infty}^{\infty}\left(uncut(P,L,S_i,O'_i) + \sum_{j\in S-S_i} uncut(P,L,S_j,O'_j)\right)$$

$$= \min_{P=-\infty}^{\infty}\left(\sum_{j\in S} uncut(P,L,S_j,O'_j)\right) = min\_set\_uncut(L,S,O')$$

Theorem 7: Let $S_{max} = max(S)$, M be the set of all tracks with $S_i = S_{max}$, N = |M|, and S_{next} be the largest $S \neq S_{max}$. If N>1 and S_{max} is a proper multiple of N and $S_{next} <= S_{max}*(N-1)/N$, then any solution to T that meets the lower bound from Theorem 1 must evenly space out the N tracks. That is, foreach M_i, with a position O_i, there must be another track M_j with a break at $O_i + S_{max}/N$.

Proof:

We will focus on *min_set_uncut*(L_{min},S,O) for $L_{min}=(S_{max}*(N-1)/N)$. From Lemma 4b we know that *min_set_uncut*(L_{min},S,O) = *min_set_uncut*(L_{min},S',O') where S' = S – all S_i with $S_i <= L_{min}$. Restated, S' = S - all S_i with $S_i <= (S_{max}*(N-1)/N)$. Since $S_{max} > S_{max}*(N-1)/N$, $S_{next} <= S_{max}*(N-1)/N$, and all tracks not in M have an $S_i <= S_{next}$, then for $L >= (S_{max}*(N-1)/N)$ *min_set_uncut*(L,S,O) = *min_set_uncut*(L,M,O').

To have a solution that meets the bound of Theorem 1, it must meet it for all L.

For $L_{min}=(S_{max}*(N-1)/N)$ Theorem 1 states:

$$min\_set\_uncut(L_{min},S,O) \le floor\left(\mid S \mid -\sum_{i=1}^{|S|}\min\left(1, L_{min}/S_i\right)\right)$$

$$min\_set\_uncut(L_{min},M,O') \le floor\left(\mid M \mid -\sum_{i=1}^{|M|}\min\left(1, L_{min}/S_i\right)\right)$$

We need to meet the bound, so

$$min\_set\_uncut(L_{\min}, M, O') = floor\left(|M| - \sum_{i=1}^{|M|} \min(1, L_{\min}/S_i)\right)$$
$$= floor\left(|M| - \sum_{i=1}^{|M|} \min(1, (S_{\max} * (N-1)/N)/S_{\max})\right)$$
$$= floor\left(|M| - \sum_{i=1}^{|M|} \min(1, (N-1)/N)\right) = floor\left(|M| - \sum_{i=1}^{|M|} (N-1)/N\right)$$
$$= floor\left(|M| - |M| * (N-1)/N\right)$$

Recall that N=|M|:

$$min\_set\_uncut(L_{\min}, M, O') = floor\left(N - N * (N-1)/N\right) = floor(N - (N-1)) = floor(1) = 1$$

This means that every region P to P+L_{min}-1 inclusive must contain breaks in at most N-1 tracks. Also, every track must have a break in every region P to P+S_{max}-1 inclusive. This means there must be at least one break outside P to P+L_{min}-1 inclusive but inside P to P+S_{max}-1 inclusive. Thus, there must be at least one break in each region P+L_{min} to P+S_{max}-1 inclusive, which means there must be at least one break in each region P to P+L_x inclusive, where L_x=S_{max}-1-L_{min}=S_{max}-1-(S_{max}*(N-1)/N)=S_{max}*N/N-(S_{max}*(N-1)/N)-1=(S_{max}/N)-1

Each of N tracks has a break in the region P to P+S_{max}-1 inclusive, which is a region of S_{max} locations, or N regions of length S_{max}/N. We also know that every region P1 to P1+(S_{max}/N)-1 must have at least one break, which is a region of length S_{max}/N. Thus, each of the N regions of length S_{max}/N must have exactly one break, and thus all regions of length S_{max}/N must have exactly one break.

Consider track T_i that is in M, which by definition must have a break at position O_i. From above, we know that there cannot be another break in the region O_i to O_i+S_{max}/N-1 inclusive. Also, there must be exactly one break in the region (O_i+1) to (O_i+1)+S_{max}/N-1. Thus, there must be exactly one track in M with a break at position (O_i+1)+S_{max}/N-1 = O_i+S_{max}/N

This break cannot be on track T_i, since the two breaks closest to this region on track T_i is O_i < (O_i+1) and O_i+S_{max}>O_i+S_{max}/N since N>1.

Thus, there must be exactly one other track in M with a break at position O_i+S_{max}/N.

Theorem 8: The solution from Theorem 7 yields the optimal *min_set_uncut*(L,T,O^T) forall L>=S_{next} and any assignment to the O_i where $T_i \notin M$.

Proof:

From the statement of Theorem 7, we know that all signals not in M have an S_i<=S_{next}. Thus, M = T-all T_i with S_i<=S_{next}.

From Lemma 4c, we know that *min_set_uncut*(L,G,O) = *min_set_uncut*(L,G',O') where G'=G-all G_i with S_i<=L_{min} and L_{min} <= L.

This means *min_set_uncut*(L,T,O^T) = *min_set_uncut*(L,M,O) where L>S_{next}.

We want to use Theorem 3, so for M we need to achieve Forall i, O_i=floor(S_{max}*(i-1)/|M|). We can do this with a series of steps.

First, convert the solution O of M to O' of M', where O'$_i$ = O_i – delta, and delta=O_1. This means that O'$_1$ = 0. Note also that from Theorem 7 we know that foreach M_i, with a position O_i, then there must be another track M_j with a break at O_i + S_{max}/N. This means that for M', foreach track M'$_i$ with a position O'$_i$ = O_i-delta, there must be another track M'$_j$ with a break at O'$_i$ + S'$_{max}$/N = O_i-delta + S'$_{max}$/N.

Thus, we have tracks with breaks at $(S'_{max}/N)*x$ for all x. Since there are N=|M| breaks from $(S'_{max}/N)*0=0$ to $(S'_{max}/N)*(N-1)<S'_{max}$, and no track with length S'_{max} can have two breaks in a region of length S'_{max}, each of these breaks is on a unique track.

From Lemma 2 we know *min_set_uncut*(L,M,O) = *min_set_uncut*(L,M',O'). Also, from Lemma 5 we can create M'' = permute (M'), where *min_set_uncut*(L,M'',O'') = *min_set_uncut*(L,M',O') = *min_set_uncut*(L,M,O). In the permutation, order each track by when their break appears in the region $0...(S'_{max}/N)*(N-1)$.

From recursive application of Lemma 6, we can produce track set Q, where the S values in Q are identical to those in M'' ($S^Q_i=S''_i$), and *min_set_uncut*(L,Q,O^Q) = *min_set_uncut*(L,M'',O''), but the O_i of each track in Q is set to that track's break in the region $0...(S'_{max}/N)*(N-1)$. Thus, for set Q, we have $O^Q_i = (S'_{max}/N)*(i-1) = floor(S_{max}*(i-1)/|M|)$.

From Theorem 3, we know that this assignment of locations is optimal for Q. We know that since *min_set_uncut*(L,M,O) = *min_set_uncut*(L,Q,O^Q), the solution O to M is also optimal.

Finally, since *min_set_uncut*(L,T,O^T) = *min_set_uncut*(L,M,O) where $S_{next} <= L$, we know that Theorem 7 yields the optimal *min_set_uncut*(L,T,O^T) for all $L >= S_{next}$.

Theorem 9: Let $S'_i = c*S_i$ for all i and for any integer c. Given a solution O' to T' with segment lengths S', which meets the bounds of Theorem 1 for all L, then the solution O with $O_i = floor(O'_i/c)$ is a solution for T with segment lengths S that meets the bounds of Theorem 1 for all L.

This theorem allows us to fix situations where the length of a set of tracks doesn't have the right factor. For example, we may want the number of tracks N with a length S_i to be a factor of S_i. If it isn't, simply multiply all track lengths by N and solve that problem. Note though that this may remove our ability to apply Theorem 2 or Theorem 3, so these might need to be applied first.

Proof:

We are given $O_i = floor(O'_i/c)$, which means
$(O'_i/c)-1 < O_i <= O'_i/c$
$O'_i-c < c*O_i <= O'_i$
$O'_i-c+1 <= c*O_i <= O'_i$

By definition we know:

$$min\_set\_uncut(L,S,O) = \min_{P=-\infty}^{\infty}\left(set\_uncut(P,L,S,O)\right) = \min_{P=-\infty}^{\infty}\left(\sum_{i\in S} uncut(P,L,S_i,O_i)\right)$$

$$= \min_{P=-\infty}^{\infty}\left(\sum_{i\in S}\begin{cases}0 & \exists x \mid P \le S_i * x + O_i < P + L\\ 1 & otherwise\end{cases}\right)$$

$$= \min_{P=-\infty}^{\infty}\left(\sum_{i\in S}\begin{cases}0 & \exists x \mid P \le S_i * x + O_i \le P + L - 1\\ 1 & otherwise\end{cases}\right)$$

$$= \min_{P=-\infty}^{\infty}\left(\sum_{i\in S}\begin{cases}0 & \exists x \mid c*P \le S_i * c * x + c * O_i \le c*P + c*L - c\\ 1 & otherwise\end{cases}\right)$$

$$= \min_{P=-\infty}^{\infty}\left(\sum_{i\in S}\begin{cases}0 & \exists x \mid \left(c*P \le S_i * c * x + c * O_i\right) \quad and \quad \left(S_i * c * x + c * O_i \le c*P + c*L - c\right)\\ 1 & otherwise\end{cases}\right)$$

$$\ge \min_{P=-\infty}^{\infty}\left(\sum_{i\in S}\begin{cases}0 & \exists x \mid \left(c*P \le S_i * c * x + O'_i\right) \quad and \quad \left(S_i * c * x + O'_i - c + 1 \le c*P + c*L - c\right)\\ 1 & otherwise\end{cases}\right)$$

$$= \min_{P=-\infty}^{\infty}\left(\sum_{i\in S}\begin{cases}0 & \exists x \mid \left(c*P \le S_i * c * x + O'_i\right) \quad and \quad \left(S_i * c * x + O'_i \le c*P + c*L - c + c - 1\right)\\ 1 & otherwise\end{cases}\right)$$

$$= \min_{P=-\infty}^{\infty}\left(\sum_{i\in S}\begin{cases}0 & \exists x \mid c*P \le S_i * c * x + O'_i \le c*P + c*L - 1\\ 1 & otherwise\end{cases}\right)$$

$$= \min_{P=-\infty}^{\infty}\left(\sum_{i\in S'}\begin{cases}0 & \exists x \mid c*P \le S'_i * x + O'_i \le c*P + c*L - 1\\ 1 & otherwise\end{cases}\right)$$

$$= \min_{P1=-\infty}^{\infty}\left(\sum_{i\in S'}\begin{cases}0 & \exists x \mid P1 \le S'_i * x + O'_i \le P1 + c*L - 1\\ 1 & otherwise\end{cases}\right)$$

$$= \min_{P1=-\infty}^{\infty}\left(\sum_{i\in S'}\begin{cases}0 & \exists x \mid P1 \le S'_i * x + O'_i < P1 + c*L\\ 1 & otherwise\end{cases}\right) = min\_set\_uncut(c*L,S',O')$$

Summarizing, *min_set_uncut*(L,S,O) >= *min_set_uncut*(c*L, S',O')

If we have a solution for S' that meets the bounds of Theorem 1, then:

$$min\_set\_uncut(c*L,S',O') = floor\left(|S'| - \sum_{i=1}^{|S'|}\min\left(1,(c*L)/S'_i\right)\right)$$

$$= floor\left(|S'| - \sum_{i=1}^{|S'|}\min\left(1,(c*L)/c*S_i\right)\right)$$

$$= floor\left(|S| - \sum_{i=1}^{|S|}\min\left(1,(c*L)/c*S_i\right)\right)$$

$$= floor\left(|S| - \sum_{i=1}^{|S|}\min\left(1,L/S_i\right)\right)$$

Thus, $min\_set\_uncut(L,S,O) \ge floor\left(|S| - \sum_{i=1}^{|S|}\min\left(1,L/S_i\right)\right)$

We know from Theorem 1 that $min\_set\_uncut(L,S,O) \le floor\left(|S| - \sum_{i=1}^{|S|}\min\left(1,L/S_i\right)\right)$

Putting these together we get $min\_set\_uncut(L,S,O) = floor\left(|S| - \sum_{i=1}^{|S|}\min\left(1,L/S_i\right)\right)$

Thus, given our formulation, the solution for T meets the bounds of Theorem 1 for all L.

Theorem 10: Given a set of tracks G, all of length X, where X evenly divisible by |G|, and a solution O with these tracks evenly distributed ($O_i = C+(i-1)*X/|G|$). There is another set of tracks G', all of length Y = |G'| * X/|G|, with a solution O' where forall L <= min(X,Y) *set_uncut*(P,L,G,O) = *set_uncut*(P,L,G',O') + |G| - |G'|. That is, the difference in *set_uncut*'s is just the difference in the number of tracks between the two sets. The solution O' is $O'_k = O_1 + (k-1)*X/|G|$ forall 1 <= k <= |G'|.

What this means is that once we've solved the layout for a set of equal length tracks by spacing them evenly (via Theorem 7) we can go to the next size, and replace all these long tracks by a few shorter tracks. If these shorter tracks, and the next length track to solve for, are be the same size this simplifies the placement problem.

Proof:

We can show that for every position with a break in G, there is a break in G'. Assume there is a break in G at position P in track T_i. We know from the definition of breaks that, for some integer Z,
$P = O_i + Z*S_i = O_i + Z*X$
$= C + (i - 1)*X / |G| + Z*X$
$= O_1 + ((i - 1) + Z*|G|) * X/|G|$

Since we have integers |G'| and ((i-1) + Z*|G|), by the definition of integer division we get ((i-1) + Z*|G|) = W*|G'| + r for some integer W and r, with 0<=r<|G'|.

Thus:
$O_1 + ((i - 1) + Z*|G|) * X/|G| = O_1 + (r + W*|G'|) * X/|G|$
$= O_1 + r*X/|G| + W*|G'|*X/|G|$
$= O_1 + ((r + 1) - 1)*X/|G| + W*|G'|*X/|G|$
$= O'_{r+1} + W*Y$

Thus, G' will have a break at P on line G'_{r+1}

We can show that for every position with a break in G', there is a break in G (the reverse of the previous). Assume there is a break in G' at position P in track I. We know from the definition of breaks that, for some integer W,
$P = O'_i + W*Y$
$= O_1 + (I -1)*X/|G| + W*|G'|*X/|G|$
$= O_1 + ((I - 1) + W*|G'|)*X/|G|$

Since we have integers |G| and ((I - 1) + W*|G'|), by the definition of integer division we get ((I - 1) + W*|G'|) = Z*|G| + r for some integer Z and r, with 0<= r <|G|. Thus:
$O_1 + ((I - 1) + W*|G'|)*X/|G| = O_1 + (r + Z*|G|)*X/|G|$
$= O_1 + r*X/|G| + Z*|G|*X/|G| = O_1 + r*X/|G| + Z*X$
$= O_{r+1} + Z*X$

Thus, G will have a break at P on line G_{r+1}

Note that neither G or G' will ever have more than one track with a break at any given position, since all tracks in a given set have the same length, and their offsets differ by less than the track length. Since no two tracks in G or G' have identical breaks, and each break in G has a corresponding break in G', and each break in G' has a corresponding break in G, there is thus a 1-to-1 relationship between breaks in G and G'.

From the definition of *set_uncut*, *set_uncut*(P,L,G,O) = |G| - B, where B is the number of tracks with breaks in the region P to P+L-1. Since L<=X, no track can have more than one break in the region P to P+L-1, thus B also equals the number of breaks in the region P to P+L-1 for G.

Similarly, from the definition of *set_uncut*, *set_uncut*(P,L,G',O') = |G'| - B', where B' is the number of tracks with breaks in the region P to P+L-1. Since L<=Y, no track can have more than one break in the region P to P+L-1, thus B' also equals the number of breaks in the region P to P+L-1 for G'.

Since there is a one-to-one correspondence between breaks in G and in G', therefore B = B'. Thus,

set_uncut(P,L,G,O) = |G| - B = |G'| - |G'| + |G| - B = |G'| - B + |G| - |G'| = *set_uncut*(P,L,G',O') + |G| - |G'|.

Theorem 11: If the G and O of Theorem 10 has, for L <= min(X,Y), *min_set_uncut*(L,G,O) – bound(L,G) = $delta_L$, where bound(L,G) is the bound from Theorem 1, then *min_set_uncut*(L,G',O') – bound(L,G') = $delta_L$.

This means that making the transformation of Theorem 10 does not impact the quality of the possible solutions.

Proof:

From Theorem 1 we have $bound(L,G) = floor\left(|G| - \sum_{i=1}^{|G|} \min(1, L/S_i)\right)$

Since S_i = X for all i, and L<=X, we have $bound(L,G) = floor\left(|G| - \sum_{i=1}^{|G|} L/X\right) = floor\left(|G| - |G| * L/X\right)$

From Theorem 10 we have Y = |G'| * X/|G|, so X = |G|*Y/|G'|. Thus,

$$bound(L,G) = floor\left(|G| - |G| * L/\left(|G| * Y/|G'|\right)\right)$$
$$= floor\left(|G| - |G'| * L/Y\right) = floor\left(|G| + \left(|G'| - |G'|\right) - |G'| * L/Y\right)$$
$$= |G| - |G'| + floor\left(|G'| - |G'| * L/Y\right)$$

Since S'_i = Y for all i, and L<=Y, we have

$$bound(L,G) = |G| - |G'| + floor\left(|G'| - \sum_{i=1}^{|G'|} \min(1, L/S'_i)\right) = |G| - |G'| + bound(L,G')$$

By definition we know that $min\_set\_uncut(L,G,O) = \min_{P=-\infty}^{\infty}\left(set\_uncut(P,L,G,O)\right)$

From Theorem 10 we get

$$min\_set\_uncut(L,G,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,G',O') + |G| - |G'|)$$
$$= \min_{P=-\infty}^{\infty}(Set\_uncut(P,L,G',O')) + |G| - |G'|$$
$$= min\_set\_uncut(L,G',O') + |G| - |G'|$$

We are given that

$$min\_set\_uncut(L,G,O) - bound(L,G) = delta_L$$
$$\left(min\_set\_uncut(L,G',O') + |G| - |G'|\right) - \left(|G| - |G'| + bound(L,G')\right) = delta_L$$
$$min\_set\_uncut(L,G',O') - bound(L,G') = delta_L$$

Theorem 12: If all S are identical, then forall L<S_1, *set_uncut*(P,L,S,O) = |T| - *set_uncut*(P+L,S_1-L,S,O)

Proof:

Foreach track T_J, the track has a break every S_1 positions, based on the formula $S_J*X + O_J$. This means that there is exactly one break in each region of length S_J. For example, there is exactly one break in the region P..P+S_J-1 inclusive. Since L<S_J, this means that there will be one break in the region P..P+L-1 inclusive, or P+L..P+S_J-1 inclusive, but not both.

By definition

$$uncut(P,L,S_J,O_J) = \begin{Bmatrix} 0 & \exists x \mid P \le S_J * x + O_J < P + L \\ 1 & otherwise \end{Bmatrix}$$

$$= \begin{Bmatrix} 0 & otherwise \\ 1 & \exists x \mid P + L \le S_J * x + O_J < P + S_J \end{Bmatrix}$$

$$= \begin{Bmatrix} 1-1 & otherwise \\ 1-0 & \exists x \mid P + L \le S_J * x + O_J < P + S_J \end{Bmatrix}$$

$$= 1 - \begin{Bmatrix} 1 & otherwise \\ 0 & \exists x \mid P + L \le S_J * x + O_J < P + S_J \end{Bmatrix}$$

$$= 1 - uncut(P + L, S_J - L, S_J, O_J)$$

By definition

$$set\_uncut(P,L,T,O) = \sum_{i \in T} uncut(P,L,T_i,O_i)$$

$$= \sum_{i \in T} \left(1 - uncut(P + L, S_i - L, S_i, O_i)\right)$$

$$= |T| - \sum_{i \in T} uncut(P + L, S_i - L, S_i, O_i)$$

$$= |T| - set\_uncut(P + L, S_1 - L, S, O)$$

Corollary Theorem 12a: If all S are identical, then forall L<S_1,

$$min\_set\_uncut(L,T,O) = |T| - \max_{P=-\infty}^{\infty}(set\_uncut(P, S_1 - L, T, O))$$

Proof:

From Theorem 12 we have *set_uncut*(P,L,S,O) = |T| - *set_uncut*(P+L,S_1-L,S,O)

By definition

$$min\_set\_uncut(L,T,O) = \min_{P=-\infty}^{\infty}(set\_uncut(P,L,T,O))$$

$$= \min_{P=-\infty}^{\infty}\left(|T| - set\_uncut(P + L, S_1 - L, T, O)\right)$$

$$= \min_{P=-\infty}^{\infty}\left(|T| - set\_uncut(P, S_1 - L, T, O)\right)$$

$$= |T| + \min_{P=-\infty}^{\infty}\left(- set\_uncut(P, S_1 - L, T, O)\right)$$

$$= |T| - \max_{P=-\infty}^{\infty}\left(set\_uncut(P, S_1 - L, T, O)\right)$$

Optimal Factor Routing Algorithm Pseudocode

```
/* Note that the algorithm is only shown optimal for cases that meet the
restrictions given in the algorithm. */

We assume, without loss of generality, that the tracks are numbered such that
S_i >= S_i+1

/* Theorem 2 */
If T can be split into two sets G1⊂T and G2 = T-G1, where all elements of G1
are relatively prime with all elements of G2, then run the algorithm
independently on G1 and G2.

/* Theorem 5 */
Convert all track lengths to factored form, using only prime factors.
While any track T_i has more of any factor P than any other {
      Set S_i = S_i/P
}

While tracks exist without an assigned O_i {
      /* Theorem 4a.  Note: only optimal if all future steps meet bound
         of Theorem 1 */
      While there exists a set of G tracks, where S_i = |G| {
            For each track in G, assign positions to those without a specific
            O_i, such that there is a track at all positions 0<=pos<|G|.
            Eliminate all tracks in G from the routing problem.
      }
      If all tracks have their O_i assigned, end.

      /* Theorem 7 and Theorem 8 */
      Let S_max = max(S), S_next = largest S_i ≠ S_max
      Let M be the set of all tracks with S_i = S_max.
      /* S_max must be evenly divisible by |M|. */
      /* We require S_next <= S_max*(|M|-1)/|M| */
      If no track has an assigned position, assign M_1's position O_i to 0.
      Assign all unassigned tracks in M to a position O_i, so that all tracks
      in M are at a position k*S_max/|M|, 0<=k<|M|.
      /* From Theorem 8, yields optimal result for lengths L>= S_next */

      If all tracks have their O_i assigned, end.

      /* Theorem 10 and Theorem 11 */
      /* We require S_next = c*S_max/|M| for some integer c>=1 */
      Eliminate all tracks in M
      Add c tracks based on, for j=0..c-1, S_j = S_next, O_j = j*S_max/|M|
      /* From Theorem 11, optimal if the resulting problem can be solved
         with bound(L,G,O). */
      /* We require that the number of tracks at S_next be divisible by c,
         so that the application of Theorem 7 will work (the unassigned
         tracks can be spread evenly */
}
```

Proof of Optimality of Optimal Factor Routing Algorithm

The application of Theorem 2 is proven to convert the combined problem into separate problems with the same set of solutions. Thus, an optimal solution to the subproblems are combined into an optimal solution to the overall problem.

Theorem 5 proves that the quality of solutions is identical after removal of any unique prime factors.

Because of all of these, the "while tracks exist" loop has access to all of the same solutions as the original problem, and thus the optimal solution to the overall problem is still available. As we will show, all rules applied in this loop either produce an optimal solution directly, or produce a partial solution that is optimal if the rest of the problem is solved optimally. Theorem 4a shows that the problem before application of the rule is as close to the bound as is the solution to the remaining problem. Thus, if all rules in the loop produce solutions identical to the bound, this step is optimal.

Theorem 8 proves that the application of Theorem 7 is optimal for all lengths $L>S_{next}$. We know that we can achieve the layout of Theorem 7 (even spacing) since no tracks will be assigned unless it came from a previous application of Theorem 7 followed by Theorem 10. Theorem 10 will produce C tracks, and we require that the number of unassigned tracks at S_{next}, which we will call X, is also a multiple of C. This means that we can place X/C unassigned tracks of length S_{next} between the assigned tracks of length S_{next}.

Note that the application of Theorem 7 involves the placement of a track to location 0 if none are already assigned. Lemma 2 guarantees that we can do this without eliminating the optimal solution.

The application of Theorem 7 guarantees an optimal result for all lengths $L>S_{next}$. We now only have to worry about cases for $L<= S_{next}$. Theorem 11 states that the application of Theorem 10 maintains our ability to find an optimal solution for all $L<=\min(S_{max}, S_{next})$. Since $S_{next}<S_{max}$, we know that we can still find an optimal solution for all $L<=S_{next}$. Also, we already have an optimal solution for all $L>S_{next}$. Thus, we can get an optimal solution for all L.