Title
An efficient wire routing and wire sizing algorithm for weight minimization of automotive systems

ISBN
9781479930173

Authors
Lin, CW
Rao, L
Giusto, P
et al.

Publication Date
2014

DOI
10.1145/2593069.2593088

Peer reviewed

An Efficient Wire Routing and Wire Sizing Algorithm for Weight Minimization of Automotive Systems

Chung-Wei Lin[1], Lei Rao[2], Paolo Giusto[2], Joseph D'Ambrosio[3], Alberto Sangiovanni-Vincentelli[1]
[1]EECS Department, University of California, Berkeley, Berkeley, CA
[2]Research & Development, General Motors Company, Palo Alto, CA
[3]Research & Development, General Motors Company, Warren, MI
E-Mails: {cwlin, alberto}@eecs.berkeley.edu, {lei.rao, paolo.giusto, joseph.dambrosio}@gm.com

ABSTRACT

As the complexities of automotive systems increase, designing a system is a difficult task that cannot be done manually. In this paper, we propose an algorithm for weight minimization of wires used for connecting electronic devices in a system. The wire routing problem is formulated as a Steiner tree problem with capacity constraints, and the location of a Steiner vertex is selected for adding a splice connecting more than two wires. Besides wire routing, wire sizing is also done to satisfy resistance constraints and minimize the total wiring weight. Experimental results show the effectiveness and efficiency of our algorithm.

1. INTRODUCTION

As the complexities of automotive systems increase, designing a system is a difficult task that cannot be done manually. It has been shown that there are hundreds to thousands of devices, such as Electronic Control Units (ECUs), actuators, and sensors, distributed in an automotive system, and this number is still increasing dramatically, which brings several concerns. First, the numbers of devices and their connections are so many that it is difficult or even impossible for designers to complete their designs manually. Next, without a systematic approach, the design space is not explored effectively and efficiently, which leads to a suboptimal solution or just a feasible solution without being optimized. Last but not least, the length of a design cycle is much increased when dealing with new designs or new constraints. To remedy these problems, automated design tools must be developed.

Due to the increase of the number of connections, the wiring cost also becomes an issue. Although some devices are connected through wireless communications, most of them are still connected through physical wires. It is reported that the wiring weight of an automotive system can be up to 30 kilograms [9], and the wiring cables, along with their harnesses, become the third heaviest and costliest component in an automotive system (after the chassis and the engine) [10]. Many automotive companies and suppliers are trying to minimize the wiring weight by developing lightweight wires [1, 9]. Therefore, when we solve the wiring problem to make sure that devices are correctly connected, it is also very important to reduce the wiring weight to further enhance the performance and fuel efficiency of an automotive system and build a user-friendly vehicle—for example, the weight of a door is a concern from users' perspective.

There are several existing tools [2, 3] for the electrical and wiring harness design. The Capital Suite [2] provides, among other features, a design flow from logical connectivity designs to manufacturing-ready harness designs for transportation platforms such as aircraft and vehicles. Especially, the Capital Integrator can automatically synthesize wiring objects and generate fully-detailed wiring designs. The software also provides interfaces for cost optimization and design validation. E3.series [3] is another tool for creating wiring designs. Besides these commercial tools, a preliminary tool [8] focuses on estimating the cost for connecting two devices. If there are more than two devices to be connected, designers will first manually assign locations of splices which are used for merging more than two wires and then use the tool to estimate the cost—this design flow should be improved so that the locations of splices can be decided automatically, and the total length and the total weight can be optimized.

In this paper, we focus on wire routing and wire sizing for weight minimization and present an algorithm to connect electronic devices in automotive systems. The wire routing problem is formulated as a Steiner tree problem with capacity constraints, and the location of a Steiner vertex is selected to add a splice. The wire routing problem and the Steiner tree problem are well-studied in the field of electronic design automation [11], but the capacity constraints are new challenges in automotive systems. We modify the KMB algorithm [7] to efficiently construct Steiner trees. We also propose a Mixed Integer Linear Programming formulation (MILP) to relocate Steiner vertices and satisfy capacity constraints. The MILP formulation is relaxed to a Linear Programming (LP) formulation which has the same optimal objective and can be solved much more efficiently. Besides wire routing, wire sizing is also done to satisfy resistance constraints and minimize the total wiring weight. To the best of our knowledge, this is the first work in the literature to address the automotive routing problem as a minimal Steiner tree problem with capacity constraints. A real industrial case study shows the effectiveness and efficiency

DAC '14, June 01–05 2014, San Francisco, CA, USA

http://dx.doi.org/10.1145/2593069.2593088

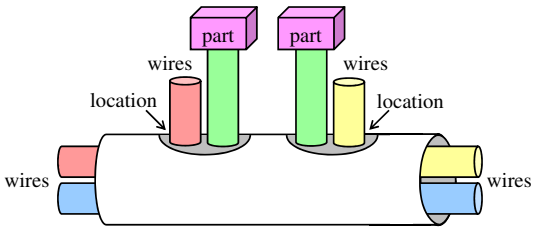

Figure 1: A harness model with its locations [8]. Two parts are connected by a wire, and the wire goes through the harness.

of our algorithm which provides an efficient, flexible, and scalable approach for the design optimization of automotive systems.

The rest of this paper is organized as follows. Section 2 formulates the problem of wire routing and wire sizing, and Section 3 presents our algorithm. Section 4 reports the experimental results, and we conclude our work in Section 5.

2. FORMULATION

A ***harness*** is a complete wire assembly, and a ***location*** of a harness is a place where wires can get in or out of the harness. The end points of a harness are usually also locations, and a harness can have several branches. A ***part*** is an ECU, a sensor, or an actuator. An example is shown in Figure 1 [8] where two parts are connected by a wire, and the wire goes through the harness. A ***splice*** is used for connecting more than two wires, and it is allocated to a location. An ***inline*** is used for connecting two harnesses. Taking Figure 2(a) as an example, v_6 is an inline and v_4 is a part, so a wire can connect the two harnesses only through v_6, not through v_4.

DEFINITION 1. *A **vertex** v is a location, a part, or an inline. It is associated with a capacity C_v ($C_v \geq 0$), defining how many splices can be allocated to the vertex.*

A splice cannot be allocated to a part or an inline, so $C_v = 0$ if v is a part or an inline. An ***l-vertex***, a ***p-vertex***, and an ***i-vertex*** are vertices corresponding to a location, a part, and an inline, respectively.

DEFINITION 2. *An **edge** e connects from an l-vertex to another l-vertex, a p-vertex, or an i-vertex. It is associated with a length L_e ($L_e \geq 0$).*

By the definition, at least one end vertex of an edge is an l-vertex.

DEFINITION 3. *A **routing graph** G includes a set of vertices V_G and a set of edges E_G.*

DEFINITION 4. *A **netlist** N is a set of p-vertices. It is associated with a resistance R_N ($R_N > 0$), defining the upper bound of its allowed resistance.*

Given a routing graph and a set of netlists, we want to find a Steiner tree for each netlist so that all p-vertices in the netlist are connected and all constraints are satisfied. Before introducing the formulation and the constraints, there are some more definitions.

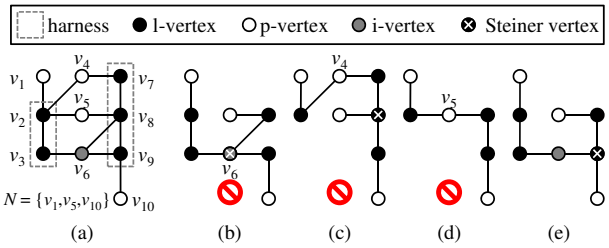

Figure 2: (a) Given a routing graph and a netlist $N = \{v_1, v_5, v_{10}\}$, a Steiner tree must be a subgraph of the routing graph. (b) The Steiner tree is not a feasible solution because v_6 is not an l-vertex (violating Constraint 2). (c) The Steiner tree is not a feasible solution because the degree of v_4 is 2 (violating Constraint 3). (d) Similarly, the Steiner tree is not a feasible solution because the degree of v_5 is 2 (violating Constraint 3). (e) The Steiner tree is a feasible solution.

DEFINITION 5. *A **segment** S in a Steiner tree is a path where its end vertices are p-vertices or Steiner vertices, and it does not go through any p-vertex or any Steiner vertex between its end vertices (except its end vertices). Its radius r_S is a decision variable.*

DEFINITION 6. *If a segment S consists of a set of edges $\{e_1, e_2, \ldots, e_{n_e}\}$, then the length L_S, the weight W_S, and the resistance R_S of S are*

$$L_S = \sum_{i=1}^{n_e} L_{e_i}; \quad W_S = \alpha \times L_S \times r_S^2; \quad R_S = \beta \times \frac{L_S}{r_S^2},$$

where α and β are constants.

DEFINITION 7. *If a Steiner tree T consists of a set of segments $\{S_1, S_2, \ldots, S_{n_S}\}$, then the length L_T, the weight W_T, and the resistance R_T of T are*

$$L_T = \sum_{i=1}^{n_S} L_{S_i}; \quad W_T = \sum_{i=1}^{n_S} W_{S_i}; \quad R_T = \sum_{i=1}^{n_S} R_{S_i}.$$

To this point, we can define the problem:

- **Problem Formulation**: given a routing graph G and a set of netlists $\{N_1, N_2, \ldots, N_n\}$, find a Steiner tree T_i for each netlist N_i and decide the radius $r_{S_{i,j}}$ for each segment $S_{i,j}$ (the j-the segment of T_i) so that all p-vertices in N_i are connected, all following constraints are satisfied, and the total weight of all Steiner trees $\sum_{i=1}^{n} W_{T_i}$ is minimized.

CONSTRAINT 1. *A Steiner tree T_i must be a subgraph of the given routing graph G.*

CONSTRAINT 2. *A Steiner vertex must be an l-vertex.*

We have this constraint because a splice is only allocated to a location. An example is shown in Figure 2. Given a routing graph in Figure 2(a), the Steiner tree in Figure 2(b) is a subgraph of the routing graph, but it is not feasible because v_6 is not an l-vertex.

CONSTRAINT 3. *The degree of a p-vertex in a Steiner tree must be 1.*

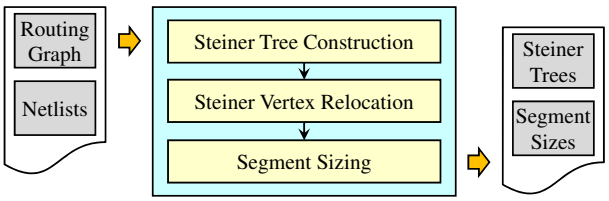

Figure 3: The flow of our algorithm.

We have this constraint because a part cannot be used for connecting wires or harnesses. It implies that a Steiner tree will not have a p-vertex which is not in the given netlist. With this constraint, the Steiner tree in Figure 2(c) is not feasible because the degree of v_4 is 2. Similarly, the Steiner tree in Figure 2(d) is not feasible because the degree of v_5 is 2 (although v_5 is in N), while the Steiner tree in Figure 2(e) is a feasible solution. Besides the constraints above, there are also capacity and resistance constraints as follows.

CONSTRAINT 4. (**Capacity Constraint**) *The number of splices allocated to a vertex v must be smaller than or equal to C_v.*

This constraint reflects that a location may have space limitation so that only a limited number of splices can be allocated to it. As mentioned before, if v is a part or an inline, then $C_v = 0$, which implies that no splice will be allocated to v.

CONSTRAINT 5. (**Resistance Constraint**) *The resistance of a Steiner tree must be smaller than or equal to the upper bound given with its corresponding netlist, i.e., $R_{T_i} \leq R_{N_i}$.*

This constraint guarantees that the delay and the quality of a signal on a wire meet the requirements.

3. ALGORITHM

The flow of our algorithm is shown in Figure 3. There are three steps: Steiner tree construction, Steiner vertex relocation, and segment sizing. The Steiner tree construction focuses on the length minimization, which is beneficial for both of the weight minimization and the resistance minimization. The Steiner vertex relocation moves Steiner vertices so that Constraint 4 (capacity constraint) is satisfied. Last, the segment sizing decides the wire sizes of all segments to meet Constraint 5 (resistance constraint) and minimize the total weight of all Steiner trees. They will be introduced in the following sections.

3.1 Steiner Tree Construction

In this step, we find a Steiner tree for each netlist one by one. The subproblem in this step is defined as:

- **Steiner Tree Construction**: given a routing graph G and a netlist N_i, find a Steiner tree T_i for the netlist N_i so that all p-vertices in N_i are connected, Constraint 1 is satisfied, and the length of T_i is minimized.

Since we assign $C_v = 0$ for a p-vertex or an i-vertex, Constraint 2 is a special case of Constraint 4 which should be considered after the Steiner trees of all netlists are constructed. Therefore, Constraints 2 and 4 are considered in the next step (the Steiner vertex relocation). Constraint 3 is

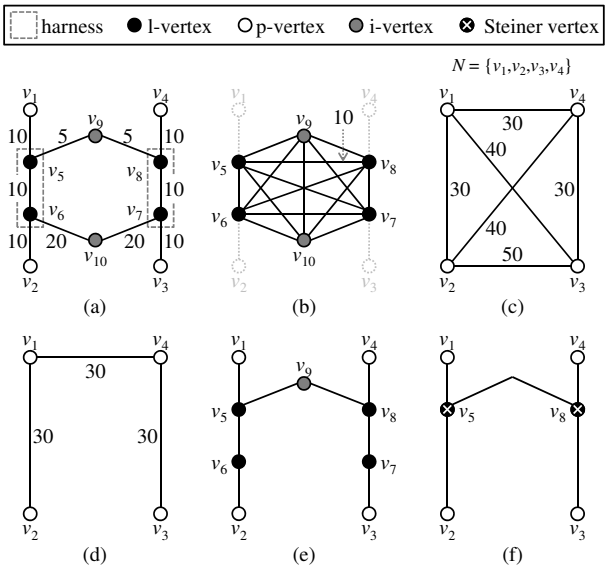

Figure 4: The Steiner tree construction. (a) Given a routing graph, (b) our algorithm computes the distances of all pairs of l-vertices and i-vertices. (c) For each netlist, our algorithm computes the distances of all pairs of p-vertices in the netlist, (d) computes a minimum spanning tree, (e) maps the spanning tree to the edges in the routing graph, and (f) constructs a Steiner tree.

not directly considered in this step, but our approach guarantees that the Steiner tree of a netlist does not include any p-vertex not in the netlist. Last, for a Steiner tree, minimizing the length and minimizing the resistance have a positive correlation, so Constraint 5 is not considered in this step, either. Even without Constraints 2 to 5, the minimum Steiner tree problem is NP-complete [5], so we apply the KMB heuristic algorithm [7] and make some modifications to match our problem. The KMB heuristic is suitable here because the shortest path between two vertices in a routing graph just needs to be computed once, while it may be referred many times for different netlists. In this step, we can also apply other approaches, such as one constructing an optimal Steiner tree [6], which may take more time.

Given a routing graph, we first use the Floyd-Warshall algorithm to compute the shortest paths between all pairs of l-vertices and i-vertices and construct a complete graph of all l-vertices and i-vertices where each edge is associated with the distance (the length of the shortest path) between its two end vertices. For example, in Figure 4(a), the shortest path between v_5 and v_8 is $< v_5, v_9, v_8 >$ with length 10, so the edge between v_5 and v_8 in Figure 4(b) is with length 10. Then, given a netlist, we compute the shortest paths between all pairs of the p-vertices in the netlist by checking the shortest paths between the connected l-vertices of the p-vertices and construct a complete graph of all p-vertices where each edge is associated with the distance (the length of the shortest path) between its two end vertices. For example, v_1 is connected to v_5, and v_4 is connected to v_8, so the shortest path between v_1 and v_4 is the sequence of $< v_1, v_5 >$, $< v_5, v_9, v_8 >$, and $< v_8, v_4 >$, where $< v_5, v_9, v_8 >$ has been recorded in Figure 4(b), and the edge between v_1 and v_4 in Figure 4(c) is with length 30. We can-

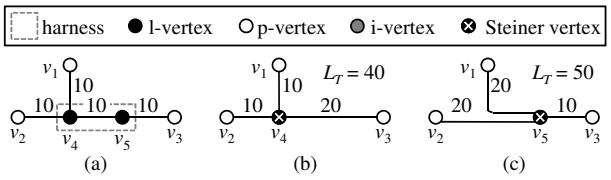

Figure 5: Given (a) a routing graph and (b) a Steiner tree, (c) the relocation cost of the Steiner vertex relocated from v_4 to v_5 is exactly the increased length of the Steiner tree.

not directly use the KMB algorithm because it may find a shortest path going through another p-vertex except its two end vertices, which is not feasible in our formulation. This is also the reason that we compute the shortest paths for two separated sets—the set of l-vertices and i-vertices and the set of p-vertices.

The following operations are the same as the KMB algorithm, as shown in Figure 4. After constructing the complete graph in Figure 4(c), a minimum spanning tree is computed in Figure 4(d). The edges of the minimum spanning tree are mapped to the edges of the routing graph in Figure 4(e), and we can decide the segments and the Steiner vertices to construct a Steiner tree in Figure 4(f). Our approach guarantees that the Steiner tree of a netlist does not include any p-vertex not in the netlist.

3.2 Steiner Vertex Relocation

After the Steiner tree of each netlist is constructed, we consider all Steiner trees to compute A_v, the number of Steiner vertices (splices) allocated to vertex v, for each vertex, and then relocate Steiner vertices to satisfy Constraints 1 to 4. The relocation cost of a Steiner vertex is first defined here.

DEFINITION 8. *If a Steiner vertex is relocated along an edge e, its* ***relocation cost*** *is L_e.*

The relocation cost is an estimation for the length increase of a Steiner tree, but it is accurate in most cases. This is because the degree of a Steiner vertex in a Steiner tree is usually 3, and the degree of a vertex in the routing graph of an automotive system is usually not very large. An example is in Figure 5. Given a routing graph in Figure 5(a) and a Steiner tree in Figure 5(b), the relocation cost of the Steiner vertex relocated from v_4 to v_5 is 10, which is exactly the increased length of the Steiner tree in Figure 5(c). We can observe that the degree of the Steiner vertex is 3, and the relocation is along an edge in the Steiner tree. As a result, the lengths of two segments (from v_1 and v_2 to the Steiner vertex) are increased by 10, and the length of the other segment (from v_3 to the Steiner vertex) is decreased by 10, making the increased length of the Steiner tree equal to 10. An alternative to assign relocation costs is to consider the degrees and the connected vertices of Steiner vertices, so two Steiner vertices relocated along the same edge e may have different relocation costs. We do not use this alternative because it induces a lot of decision variables in our Mixed Integer Linear Programming (MILP) formulation.

It is possible that the degree of a p-vertex in a Steiner tree is more than 1 after the first step (the Steiner tree construction). In this special case, we assign a Steiner vertex allocated to the p-vertex so that it will be relocated later (since the capacity of a p-vertex is 0). This assignment makes sure that Constraint 3 is satisfied after this step. To this point, the subproblem in this step is defined as:

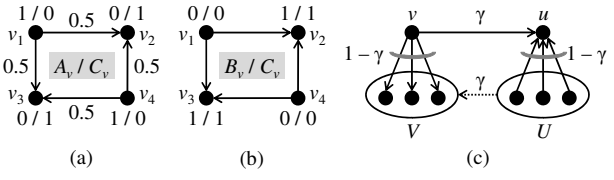

Figure 6: (a) A non-integer solution of the LP formulation leads to (b) a relocation where B_v is an integer for each vertex v, and (c) its generalized view.

- **Steiner Vertex Relocation**: given a routing graph G and the value A_v of each vertex, relocate Steiner vertices so that Constraints 1 to 4 are satisfied, and the total relocation cost of all Steiner vertices are minimized.

We do not directly consider Constraint 5 here, but, when a Steiner vertex needs to be relocated, we will select one from a Steiner tree whose resistance constraint is looser. The subproblem can be formulated as an MILP formulation. Assuming $s(e)$ is the first vertex of e, $t(e)$ is the second vertex of e, x_e is the decision variable representing the number of Steiner vertices relocated from $s(e)$ to $t(e)$, and y_e is the decision variable representing the number of Steiner vertices relocated from $t(e)$ to $s(e)$, the MILP formulation is defined as:

$$\text{minimize} \qquad \sum_e (L_e x_e + L_e y_e), \tag{1}$$

subject to

$$A_v - \sum_{e,s(e)=v} (x_e - y_e) + \sum_{e,t(e)=v} (x_e - y_e) \leq C_v; \tag{2}$$

$$A_v - \sum_{e,s(e)=v} (x_e - y_e) + \sum_{e,t(e)=v} (x_e - y_e) \geq 0, \tag{3}$$

where x_e and y_e are non-negative integers. Since A_v and C_v are integers, the MILP formulation can be relaxed to a Linear Programming (LP) formulation (where x_e and y_e are non-negative real numbers) which can be solved much more efficiently, and their optimal objectives of are the same.

LEMMA 1. *There is an optimal solution $(\vec{x'}, \vec{y'})$ to the LP formulation such that, after the Steiner vertex relocation, the number of Steiner vertices (splices) allocated to each vertex v, B_v, is an integer.*

Due to the limitation of space, we omit the proofs of theorems and lemmas in this paper.

B_v is exactly the left-hand side in Equations (2) and (3). The lemma above only claims that there is an optimal solution such that B_v is an integer for each vertex v. It does not claim that it is an integer solution. For example, a non-integer solution of the LP formulation in Figure 6(a) leads to a relocation where B_v is an integer for each vertex v in Figure 6(b). The next lemma can provide a stronger claim.

LEMMA 2. *There is an integer optimal solution to the LP formulation.*

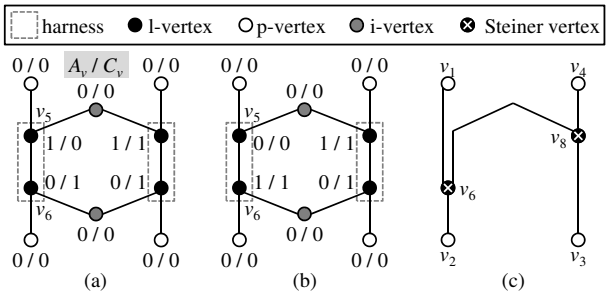

Figure 7: The Steiner vertex relocation. Assuming that there is only one Steiner tree in Figure 4, (a) our algorithm computes A_v, the number of Steiner vertices (splices) allocated to each vertex, (b) calls an LP solver and decides to relocate one Steiner vertex from v_5 to v_6, and (c) relocates a Steiner vertex of the Steiner tree from v_5 to v_6.

The basic concept of Lemma 2 is illustrated in Figure 6(c). Given an optimal solution provided by Lemma 1, if the solution relocates γ $(0 < \gamma < 1)$ Steiner vertex from vertex v along edge e to vertex u, then, by Lemma 1, $1-\gamma$ Steiner vertex must be relocated from v to a set of vertice V, $1-\gamma$ Steiner vertex must be relocated from a set of vertices U to u, and γ Steiner vertex must be relocated from U to V through some paths. As a result, we can just move 1 Steiner vertex from v to u and 1 Steiner vertex from U to V, and this integer solution is also an optimal solution to the LP formulation.

THEOREM 1. *The optimal objective (λ^*_{MILP}) of the MILP formulation is equal to that (λ^*_{LP}) of the LP formulation.*

The concept of Lemma 2 provides an insight to deal with a non-integer solution. We round up the value of an edge, search its corresponding edges, and round down their values. After getting an integer solution, we sort all of the Steiner trees obtained in the first step (the Steiner tree construction) by the tightness of their resistance constraints. We start from the Steiner tree whose resistance constraint is the loosest and greedily relocate Steiner vertices until the relocation matches the integer solution. An example of the Steiner vertex relocation is shown in Figure 7. Assuming that there is only one Steiner tree in Figure 4, our algorithm computes A_v, the number of Steiner vertices (splices) allocated to each vertex as shown in Figure 7(a). Then, it calls an LP solver and decides to relocate one Steiner vertex from v_5 to v_6 as shown in Figure 7(b). Last, it relocates a Steiner vertex of the Steiner tree from v_5 to v_6 as shown in Figure 7(c).

In this step, we need to deal with two special cases. The first special case is that the degree of a p-vertex in a Steiner tree is more than 1 after the first step (the Steiner tree construction). As mentioned before, we assign a Steiner vertex allocated to the p-vertex so that it will be relocated. This relocation may let a segment go through a p-vertex which is not its end vertex, violating Constraint 3 (or violate the definition of a segment). This case does not happen very often since a p-vertex is usually connected to only one l-vertex. If this case happens, the segment will be rerouted by checking the shortest paths between l-vertices, which have been computed in the first step. The second special case is that there is no feasible solution to the LP formulation since the total capacity of all vertices is smaller than the total number of Steiner vertices in Steiner trees. If this case happens, Steiner vertices in the same Steiner tree will be merged to reduce the total number of Steiner vertices. Of course, if the total capacity of all vertices is smaller than the number of netlists with more than two parts, then the problem itself has no feasible solution.

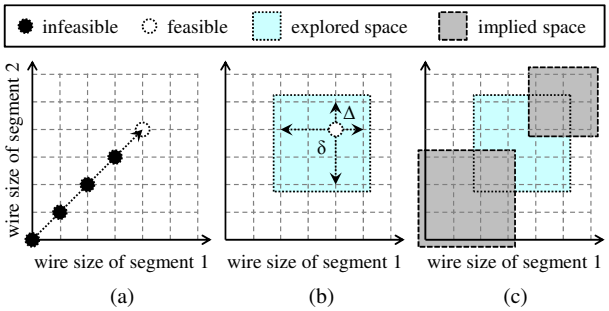

Figure 8: The segment sizing. (a) The heuristic assigns the wire size of each segment as the smallest possible wire size and increases the wire sizes of all segments together until the resistance constraint is satisfied [8]. (b) Assuming that the found value is the i-th wire size, the algorithm explores all combinations of wire sizes with indices from $i-\delta$ to $i+\Delta$. (c) There are two implied design spaces that we do not need to explore.

3.3 Segment Sizing

There is a list of possible wire sizes following the standard American Wire Gauge (AWG) [4]. The subproblem in this step is defined as:

- **Segment Sizing**: given a Steiner tree T_i, decide the radius of each segment in the Steiner tree T_i so that Constraint 5 is satisfied, and the weight of T_i is minimized.

If the number of segments is small (*e.g.*, ≤ 5) in a Steiner tree, then we explore all combinations of their wire sizes. Otherwise, we use a heuristic to decide their wire sizes. As shown in Figure 8(a), the heuristic assigns the wire size of each segment as the smallest possible wire size and increases the wire sizes of all segments together until the resistance constraint is satisfied [8]. Assuming that the found value is the i-th wire size, the heuristic then explores all combinations of wire sizes with indices from $i-\delta$ to $i+\Delta$, where δ and Δ are parameters in experiments, as shown in Figure 8(b). There are two implied design spaces in Figure 8(c) that we do not need to explore. We can guarantee that there is no feasible solution in the first (bottom-left) implied design space, and there is no better solution in the second (top-right) implied design space. If there are m segments and n possible wire sizes, then the total number of solutions is m^n, and the total number of covered solutions is

$$(i-1)^n + (m-i+1)^n + (\delta+\Delta+1)^n - \delta^n - (\Delta+1)^n. \quad (4)$$

In the best case ($i = 1$), the heuristic covers the whole design space. In the worst case ($i \approx \frac{m}{2}$), the heuristic covers about $2^{-(n-1)}$ of the whole design space. Although the covered design space is not large in the worst case, the heuristic tends to assign similar wire sizes to the segments, which prevents the case that a wire size is very small (usually not

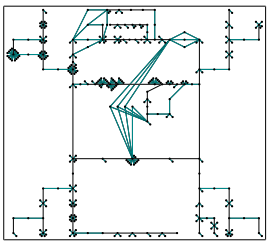

Figure 9: The routing graph of the test case.

feasible) or very large (usually not optimal). Experimental results will show the effectiveness of this heuristic.

After this step, we can get a solution to the problem formulated in Section 2. If a solution is returned, it satisfies all constraints. Furthermore, if a netlist has only two parts, then we can guarantee that its solution is optimal. This property is important since there are many 2-part netlists in a real test case.

4. EXPERIMENTAL RESULTS

We obtained an industrial test case which includes a set of functions, such as object detection, exterior lighting, mirror control, and window control. There are 17 harnesses, 102 locations, 20 inlines, 248 parts, 387 edges, and 100 netlists, and its routing graph is shown in Figure 9. One semi-automated solution based on designers' experience is given with the test case, where the locations of all splices and the sizes of all segments are given. Our algorithm is implemented in C/C++. CPLEX 12.5 is used as the LP/MILP solver. The experiments were run on a 2.5-GHz processor with 4GB RAM. The parameters δ and Δ in the accelerated heuristic mentioned in Section 3.3 are set to 5 and 1, respectively.

The results are listed in Table 1. In Step 1 (Steiner tree construction), our algorithm takes only $0.078s$ to construct the initial Steiner trees for all netlists, but it needs the next step to satisfy capacity constraints. In Step 2 (Steiner vertex relocation), our algorithm gets a solution with the total length $285{,}010mm$ and improves the semi-automated solution by more than 4%. As proved in Section 3.2, the optimal objectives of the LP formulation and the MILP formulation are the same, but using the LP formulation (it directly returns an integer solution) is more efficient ($0.671s$ vs. $0.733s$). In Step 3 (segment sizing), our algorithm gets a solution with the total weight $2.707kg$ and improves the semi-automated solution by more than 10%. In this case, although this property is not guaranteed by a theorem, the accelerated heuristic gets the same solution with a brute-force approach and reduces the runtime $106.595s$ by almost 70X. In fact, if δ and Δ are both set to 1 for this test case, the accelerated heuristic can still get the same solution, indicating that an optimal solution tends to have similar wire sizes assigned to all segments in a Steiner tree, and it is exactly the design space we explore.

If we apply our segment sizing on the Steiner trees of the semi-automated solution, we can get a 2% improvement on the total weight. This is because our algorithm can cover much more solutions (Equation (4)) than a semi-automated approach. Furthermore, our solution still has about 8% less weight than the semi-automated solution with our segment sizing. This is because the total weight and the total length have a quadratic relation—if the length of a segment is doubled, its area also needs to be doubled to maintain the same resistance. As a result, the weight of the segment becomes four times to the original weight. This means that minimizing the total length is very important for the weight minimization, so an effective and efficient algorithm must be applied on it.

Table 1: Comparison between the semi-automated solution and our solution. In Steps 2 and 3, the numbers in the parentheses are the runtime with the MILP formulation (without the LP relaxation) and the runtime with a brute-force approach, respectively.

		Semi-Automated	Our Algorithm
Step 1	Length (mm)	—	281,260
	Runtime (s)	—	0.078
Step 2	Length (mm)	297,310	285,010
	Comparison	1.000	0.959
	Runtime (s)	—	0.671 (0.733)
Step 3	Weight (kg)	3.031	2.707
	Comparison	1.000	0.893
	Runtime (s)	—	1.529 (106.595)

To demonstrate the efficiency of the LP relaxation, we generate a random test case with 500 vertices and tight capacity constraints (90% of the total capacity will be used). In this test case, the LP formulation (it directly returns an integer solution) and the MILP formulation take $1.978s$ and $10.898s$ to find an optimal solution, respectively. This indicates that the LP relaxation is necessary for a large-scale test case.

5. CONCLUSION

In this paper, we proposed a wire routing and wire sizing algorithm for weight minimization of automotive systems. A real industrial case study showed the effectiveness and efficiency of our algorithm which provides an efficient, flexible, and scalable approach for the design optimization of automotive systems, and thus improves the design flow of automotive systems.

6. REFERENCES

[1] *Delphi Aluminum Cable Systems.* http://delphi.com/manufacturers/auto/ee/cables/cable-al/.

[2] *Mentor Graphics Capital.* http://www.mentor.com/products/electrical-design-software/capital/.

[3] *Zuken E3.series.* http://www.zuken.com/en/products/electrical-wire-harness-design/e3-series.

[4] ASTM International. Standard specification for standard nominal diameters and cross-sectional areas of awg sizes of solid round wires used as electrical conductors. *ASTM Standard B258-02*, 2002.

[5] R. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, Mar 1972.

[6] T. Koch and A. Martin. Solving steiner tree problems in graphs to optimality. *Networks*, 32(3):207–232, 1998.

[7] L. T. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta Inf.*, 15:141–145, 1981.

[8] C.-W. Lin, L. Rao, J. D'Ambrosio, and A. Sangiovanni-Vincentelli. Electrical architecture optimization—cost minimization through wire routing & wire sizing. *SAE World Congress & Exhibition*, Apr 2014.

[9] K. Oba. Wiring harnesses for next generation automobiles. *Fujikura Technical Review*, 42:77–80, Mar 2013.

[10] K. Pretz. Fewer wires, lighter cars. *IEEE The Institute*, Apr 2013.

[11] J. Soukup. Circuit layout. *Proceedings of the IEEE*, 69(10):1281–1304, Oct 1981.