

**FINAL REPORT OF THE
UC DENVER MATHEMATICS CLINIC
Electrical Box Placement and Wiring for
Launch Vehicles**

Taught by

Stephen C. Billups and Michael S. Jacobson

University of Colorado Denver

Department of Mathematical Sciences

P.O. Box 173364

Denver, CO 80217-3364

Stephen.Billups@ucdenver.edu, Michael.Jacobson@ucdenver.edu

<http://math.ucdenver.edu/~billups/>,

<http://math.ucdenver.edu/~msj/>

Sponsored by

United Launch Alliance

Participating students:

Christopher Aquinto, Deborah Arangno, Deborah Fisher,

Jason Lathrop, Trevor McElhaney, Timothy Morris,

Michelle Rendon, Breeann Tonnsen, Dmitriy Vassilyev

Fall Semester 2009

Contents

List of Figures	ii
List of Tables	iv
Acknowledgements	vi
1 Introduction	1
1.1 Problem Description	2
1.2 Baseline Model	3
1.2.1 Vehicle Topology	3
1.2.2 Box specification	3
1.2.3 Wiring Connections	5
1.2.4 Constraints	5
1.3 Overview of Algorithms	5
1.3.1 Team 1: Local Search	6
1.3.2 Team 2 – Finding Good Starting Points	7
2 Local Search Algorithm for Electrical Wiring and Box Placement	9
2.1 Introduction	9
2.2 Center of Mass and Wire Routing Algorithms	10
2.3 Local Search Algorithm	13
2.4 Results	15
2.5 Direction for Further Study	17
3 Some Best First Guesses	19
3.1 Introduction	19
3.2 Main Results	20

3.2.1	Description of Algorithm:	20
3.3	Results	28
3.4	Conclusions	30
4	A Graph-Combining Scatter Search Algorithm for Avionics	
	Box Placement	33
4.1	Introduction	33
4.2	Background: Scatter Search	34
4.2.1	Choice of Scatter Search and Graph Combining	35
4.3	Scatter Search and Graph Combining Algorithm	36
4.3.1	Solution Representation	36
4.3.2	Combining Solutions: Graph Combining	38
4.3.3	Phase II Box Placement	40
4.3.4	Selection of the Reference Set	42
4.3.5	Creation of New Populations	42
4.3.6	Stopping Criteria and Selection of the Best Solution	43
4.3.7	Initialization	44
4.4	Results	44
4.5	Conclusion	45
4.6	Future Work	51
5	Conclusions and Future Work	53
5.1	Conclusions	53
5.2	Avenues for Further Study	54
5.2.1	Local Search	54
5.2.2	Heuristics	56
5.2.3	Scatter Search and Graph Combining	57
	Glossary	58
	Bibliography	58

List of Figures

1.1	Diagram of vehicle layout, with some boxes wired	4
2.1	An example of a graph	11
2.2	Dijkstra's algorithm.	12
2.3	Moving the top box to the right is a small improvement, but moving to the left, after rewiring is a significant improvement.	14
2.4	One panel of an arrangement before local search was applied. .	16
2.5	The same panel after local search was applied.	16
3.1	An example of a circular list of 12 boxes	21
3.2	A possible box configuration based on the above circular list .	21
3.3	An example of an input matrix that gives the mass of the connections per grid unit for seven boxes	21
3.4	The progression of the first three box placements in the example	22
3.5	Three possibilities for box size versus mass interval width, which are presented as parentheses	24
3.6	Example of the placement on the grid of three boxes, whose mass interval widths are all greater than their size	25
3.7	Example of the placement on the grid of three boxes. The first two have mass interval width less than the size of the box, and the third is greater	26
3.8	A flow chart of the algorithm that places boxes onto grid . . .	31
3.9	A flow chart of the algorithm that places boxes onto grid . . .	32
3.10	Given one circular list and placement on the grid, these are two possible panel divisions including the movement of boxes to make them fit	32
4.1	Distance Matrix	37
4.2	Solution Representation	37

4.3	Combination of Distance Matrices	38
4.4	Solution Representation	39
4.5	Combination Solution	40
4.6	Box Placement	41
4.7	Pareto Frontier	43
4.8	Scatter Search Results Test Case I	45
4.9	Scatter Search Results Test Case II	46
4.10	Scatter Search Results Test Case III	47
4.11	Scatter Search vs. Local Search	48
4.12	Scatter Search vs. Local Search	49
4.13	Box Placement	50

List of Tables

4.1	Data Set I	50
4.2	Data Set II	50

Acknowledgements

The success of the Mathematics Clinic program is critically dependent on finding an important and mathematically interesting problem that will fuel the creative energies of our students. We are, therefore, deeply indebted to Zach Richards at United Launch Alliance for providing us with this very interesting and challenging problem. It has been great fun to work on! We are also grateful to United Launch Alliance for their financial support and for hosting a very interesting tour of their vehicle assembly facilities.

Finally, we would like to thank the students who participated. Their enthusiasm, hard work, and creative energy made this clinic a great experience. They are

Christopher Aquinto,	Timothy Morris,
Deborah Arangno,	Michelle Rendon,
Debbie Fisher,	Breeann Tonnsen, and
Jason Lathrop,	Dmitriy Vassilyev (Teaching Assistant).
Trevor McElhaney,	

Stephen C. Billups and Michael S. Jacobson
Math Clinic instructors

Chapter 1

Introduction

This report summarizes the results of the Fall 2009 Mathematics Clinic, which was sponsored by United Launch Alliance. The Mathematics Clinic is a project-based course offered by the University of Colorado Denver Department of Mathematical and Statistical Sciences. In the Mathematics Clinic, graduate and advanced undergraduate students work in teams to address a problem of interest to a sponsoring organization.

The project focused on the *Electrical Box Placement and Wiring Problem* (EBPWP). A detailed description of this problem is given in Section 1.1; but briefly, the problem is concerned with determining where to place electrical boxes on the panels or decks of a space vehicle in order to minimize the total mass of interconnecting wires, while also ensuring that the center of mass of the boxes is close to the central axis of the vehicle. The center of mass constraint ensures that the boxes don't throw the vehicle out of balance. Depending on the particular vehicle being designed, any number of other design constraints may also need to be satisfied. Our task was to develop a computer algorithm that could generate optimal or nearly optimal box-placement solutions.

The EBPWP problem has several characteristics that influenced our algorithmic approach. First, some of the potential design constraints may be very difficult to evaluate (perhaps requiring running simulations to determine whether or not the constraints are satisfied). Because of this, we restricted our approach to so-called “black-box” or “derivative-free” optimization algorithms.

Secondly, even with simple constraints, the EBPWP is a very challenging combinatorial optimization problem. In fact, the problem can be shown to

be NP-hard. From a practical perspective, this effectively means that it is impossible to find a polynomial-time algorithm that will be guaranteed to find the optimal solution. In other words, the computational time required to find the optimal solution will grow exponentially with the size of the problem. (A precise definition of NP-hard is given in the glossary). Because of this, finding *the* (global) optimum is intractable for all but very small instances of the problem. Therefore, we focused our attention on developing heuristic methods aimed at finding good solutions in a reasonable amount of time.

In the following sections, we describe the Electric Box Placement and Wiring Problem in detail and then give an overview of the algorithms developed this semester to solve the problem.

1.1 Problem Description

The Electrical Box Placement and Wiring Problem (EBPWP) is to determine where to place various electrical boxes on the panels or decks of a spacecraft. Each box must be connected to some of the other boxes with wires of various types as specified by an interconnection matrix. The design goal is to minimize the total mass of wires used while satisfying a variety of design constraints.

Design Constraints: The design must satisfy a variety of constraints (depending on the particular vehicle). These may include (but are not limited to) the following:

- The center of mass must be within an acceptable distance from the central axis of the vehicle. (So that the vehicle is balanced).
- Some devices may create a significant amount of heat. Such devices need to be spread out to avoid excessive thermal loading.
- There may be constraints on the length of wire connecting two devices.
- Boxes with redundant functions should be physically separated so that a single event will be less likely to damage more than one of the boxes.
- There may be restrictions on the placement of the wires between boxes. For example the wires may have to follow wiring channels laid out

in a rectangular grid. Additionally, wires may not be allowed to run underneath boxes.

- Two boxes cannot occupy the same space.
- The boxes must be placed within a fixed topology. (We assume that the shapes and sizes, and relative positions of the panels and decks of the spacecraft are fixed). Further, there may be restrictions as to where the boxes can be placed within this topology. For example, they may be limited to discrete locations.

1.2 Baseline Model

The ultimate goal of this project is to develop a computer algorithm capable of solving the EBPWP in for a variety of instantiations. As a first step, this Mathematics Clinic focused on the following specific baseline model.

1.2.1 Vehicle Topology

We assume the vehicle consists of four rectangular panels out into a rectangular prism configuration. (See Figure 1.1). The height and width of the rectangles are specified as input (and the dimensions of the four rectangles are identical).

We assume that boxes can be placed only at fixed locations on the panels. These locations are specified by a rectangular lattice of evenly spaced grid points. The distances between grid points are specified as input.

1.2.2 Box specification

We assume that boxes are rectangular prisms. The dimensions of the boxes are specified as input, but the length and width are constrained to be integer multiples of the grid spacing.

For simplicity, we assume that the electrical connections for each box are located on the upper left corner of the box.

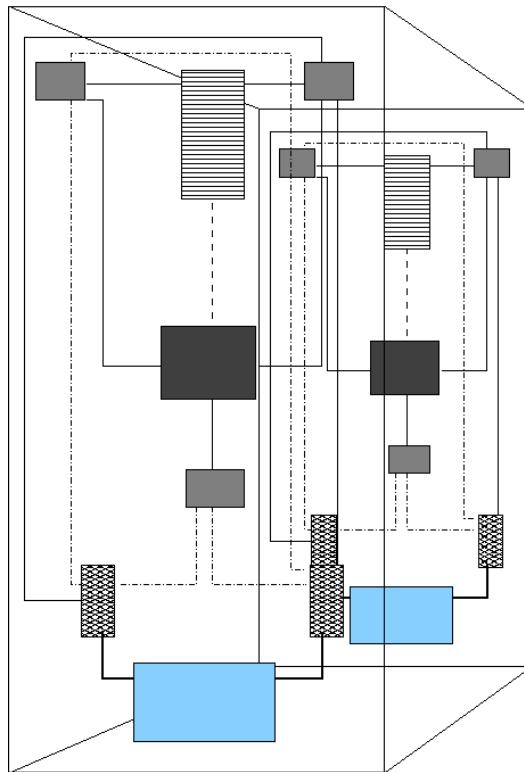


Figure 1.1: Diagram of vehicle layout, with some boxes wired

1.2.3 Wiring Connections

An interconnection matrix specifies which pairs of boxes must be connected together by electrical wires. The interconnection matrix is an $n \times n$ matrix, where n denotes the number of boxes. The (i, j) th entry of this matrix represents the mass per unit length of the wire connecting box i and box j .

1.2.4 Constraints

The following constraints must be satisfied:

- The center of mass must be within a given distance from the center of the vehicle. This distance is specified as input.
- Wires must be placed in wiring channels, which run horizontally and vertically. The wiring channels are assumed to be located along the grid lines specified in the vehicle topology.
- Wires may not underneath boxes.
- Two boxes cannot occupy the same space. And in fact, boxes must be separated by at least one grid unit.

1.3 Overview of Algorithms

To tackle the problem, the students in the Mathematics Clinic were divided into three teams, each with a different focus. The first team developed a local search algorithm, which, given a starting box placement, makes incremental improvements until a locally optimal solution is obtained.

The remaining two teams of students explored two very different approaches to finding good starting points from which to run the local search algorithm. The motivation for this lies in the fact that the EBPWP problem is characterized by a very large number of local optima, most of which are not very good solutions. Different starting points generally lead to different local optima. Thus, simply running the local search algorithm from a random starting point is not likely to produce a satisfactory solution. Instead, some way of finding good starting configurations is needed.

One obvious strategy would be to run the local search algorithm from a large number of random starting points. However, experience on related combinatorial optimization problems suggests that this approach is not likely to be very satisfactory. Something better is needed. Therefore, the remaining two teams of students developed two very different approaches for generating good starting points.

1.3.1 Team 1: Local Search

Team 1, which consisted of Deborah Arangno, Trevor McElhany, and Tim Morris, developed a local search algorithm to minimize a penalized objective function, which measures the quality of the solution. The penalized objective function can be written in the form

$$\Theta = w + \rho d,$$

where w denotes the total wire mass, d measures how badly the center of mass of the boxes violates the mass balance constraint, and ρ is a penalty parameter specified by the user.

It is important to note that the term *locally optimal* is meaningful only with respect to a prescribed *neighborhood structure*, which specifies which solutions are considered neighbors. Once the neighborhood structure is defined, then a locally optimal solution is one that has no neighbor that is strictly better. As an example, in the current implementation of the algorithm, we say that two solutions are *neighbors* if one solution can be obtained from the other by moving a single box either one unit horizontally or one unit vertically.

As part of developing the local search algorithm, this team also developed algorithms to calculate the optimal routing of wires between boxes, and also to calculate the center of mass of the boxes.

Additionally, this team augmented the local search algorithm with a rudimentary version of simulated annealing in order to escape poor local minima in pursuit of better solutions.

A complete description of these algorithms is given in Chapter 2, which was written by this team of students.

1.3.2 Team 2 – Finding Good Starting Points

Team 2 (Debbie Fisher, Jason Lathrop and Breeann Tonnsen) developed a heuristic method for creating good starting points based on observations about the structure of the problem. Specifically, the method is based on the intuition that highly interconnected subsets of boxes should be placed close together in the final configuration.

To exploit this idea, the team developed a technique that analyzed the interconnection matrix in order to produce a circular list of boxes. This list is generated in such a way that highly interconnected groups of boxes are placed close together on this list. After the circular list is constructed, it is used to create a layout of boxes, where boxes that are close together on the circular list are close together on the layout. This layout is constructed so that the mass of the boxes is relatively evenly distributed around the perimeter of vehicle, and are also spread out vertically. While this almost certainly is not an optimal layout, it provides the local search algorithm with the maximum amount of flexibility, so provides an excellent starting point for local search.

Preliminary test results indicate that this heuristic strategy does significantly better than a random starting point.

This heuristic method is described in Chapter 3, which was written by this team.

Team 3 – Scatter Search with Graph Combining Team 3 (Christopher Aquinto, Michelle Rendon, and Dmitriy Vassilyev) was responsible for developing a metaheuristic strategy for finding a globally optimal solution. A metaheuristic is a high-level algorithmic strategy that guides other heuristics (such as local search) in searching for a globally optimal solution. After studying several different metaheuristics, this team decided to implement a Scatter Search method [2].

Scatter Search is an evolutionary algorithm which maintains an ever improving population of solutions. To implement this strategy, this team developed a novel idea for combining two solutions to generate new solutions that have a higher likelihood of improvement than randomly generated solutions. This technique is called *graph combining*.

Test results demonstrate that this method generates solutions that are significantly better than the best solution found by running local search repeatedly from a large number of randomly generated starting points.

The method is described in detail in Chapter 4, which was written by this team of students.

Chapter 2

Local Search Algorithm for Electrical Wiring and Box Placement

Deborah Arangno
Trevor McElhaney
Timothy Morris

2.1 Introduction

Any vehicle designed for flight must take into account weight and balance. With this in mind, we consider the problem of minimizing the mass of the wires necessary to connect electrical components together, subject to the restriction that those components must be placed in the vehicle so that they do not throw the center of mass off by more than a predetermined tolerance. In particular, we consider the problem of placing these components on four panels that make up the vertical sides of a cube (IE the sides of a cube except the top and bottom). Each of these panels has a grid of rails on which the boxes holding the electrical components are mounted. Each box is sized so that every edge is along one of the rails and any two boxes must be separated by a space. The boxes are connected by wires which must run along the rails as well. It is assumed that no wire can run along a rail that runs underneath a box. Wires can, however, run along rails that are at the edge of a box.

Once all the boxes are located on the panels, it is frequently possible

to make small movements in the location of some of the boxes to decrease the weight of the wire needed and/or make improvements in the location of the center of mass. The evaluation of these relatively small modifications is referred to as a *local search*. In order to carry out this local search it is necessary to be able to determine both the center of mass of the boxes as well as the weight of the wire necessary to wire the boxes together. With this requirement in mind, there are really three distinct algorithms necessary to carry out the local search. These algorithms are: an algorithm to determine the center of mass of the boxes, an algorithm to determine an optimal wire routing and the mass of the wires used in that wire routing, and the local search algorithm itself.

2.2 Center of Mass and Wire Routing Algorithms

To find the center of mass, we assume that we need only consider two dimensions. So the center of mass algorithm ignores the z -component of the box location. Also, the center of mass calculation only takes the boxes into account and ignores the weight of the wire. The center of mass algorithm, then, consists of simply calculating the weighted average of the center of mass of each of the boxes. Thus, the center of mass algorithm consists of finding:

$$\bar{x} = \frac{\sum_{i=1}^k m_i x_i}{\sum_{i=1}^k m_i} \quad (2.1)$$

and

$$\bar{y} = \frac{\sum_{i=1}^k m_i y_i}{\sum_{i=1}^k m_i} \quad (2.2)$$

where we have k boxes and box i has mass m_i and center of mass (x_i, y_i) . We then find the distance from the center of mass, (\bar{x}, \bar{y}) , to the actual center. This number is the center of mass offset.

In order to build the wiring algorithm, we first introduce the concept of a graph. A *graph* is a set of vertices together with a set of edges where the edges are unordered pairs of vertices. Frequently, we think of a graph pictorially as a collection of dots (vertices) with lines (edges) connecting some of the vertices together. For example, a graph like the one in Figure 2.1 will be used to represent the rails on the vehicle. In a graph two vertices are said to be

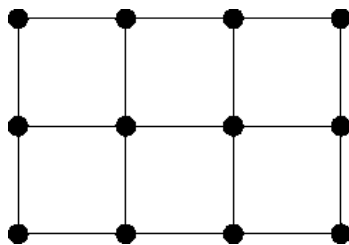


Figure 2.1: An example of a graph

neighbors if there is an edge between those two vertices. A *path* is a sequence v_1, v_2, \dots, v_k of vertices such that $v_i \neq v_j$ if $i \neq j$ and v_{i+1} is a neighbor of v_i for each $i = 1, 2, \dots, v_{k-1}$.

An efficient algorithm for finding shortest paths was created by Dijkstra [1]. Dijkstra's algorithm works on a *weighted graph* which is just a graph where there is a *weight* or number representing, in our case, the distance between vertices associated with each edge. The algorithm is as follows.

1. Start with a vertex v . Assign the value 0 to v and ∞ to every other vertex. Associate with each neighbor u_i of v , both the vertex v (as the predecessor) and the weight of the edge between v and u_i (replacing ∞). Mark the vertex v as visited.
2. Of the vertices that have not been marked visited, select a vertex v' with the lowest number t associated with it. Consider every neighbor u'_i of v' that has not been marked visited. If the number associated with u'_i is greater than $t + k$ where k is the weight of the edge between v' and u'_i , associate the vertex v' as the predecessor of u'_i and associate $t + k$ with u'_i . Mark v' as visited.
3. Repeat step 2 until all vertices of the graph are marked as visited.

For example, in Figure 2.2, step 1 assigns the value of 0 to v and ∞ to u_1, \dots, u_3 . It then assigns v as the predecessor of u_1 , u_2 , and u_3 and associates the number 2 with u_1 , 3 with u_2 , and 5 with u_3 . It then marks v as visited. Step 2 then starts with u_1 and would assign 6 to u_3 , but u_3 already has a lower number, so step 2 simply marks u_1 as visited. Step 3 starts with u_2 and assigns the value 4 to u_3 and puts the predecessor of u_3

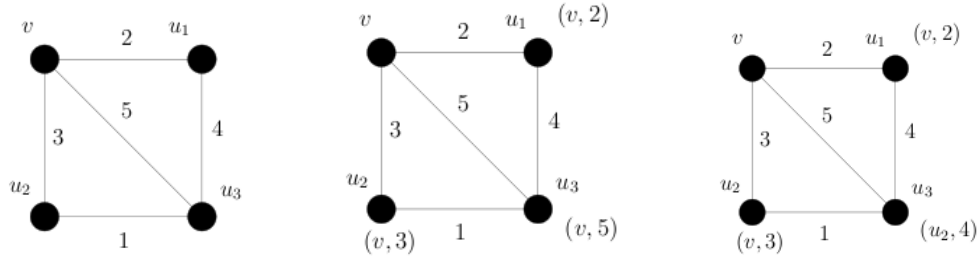


Figure 2.2: Dijkstra's algorithm.

as u_2 then marks u_2 as visited. Step 4 simply marks u_3 as visited. Then all vertices have been visited, so the algorithm terminates.

Now, the number associated with each vertex u in the graph is the total length of the path from v to u . We reconstruct the path backwards: start with u and move to u_1 the predecessor of u . Continue in this fashion always moving from u_i to the predecessor of u_i until you reach the vertex v .

For simplicity, the algorithm will connect the upper left hand corner of boxes that are to be connected together. Now we are ready to describe the algorithm used to produce the wire routing.

1. Build a graph where each vertex represents an intersection of two rails and each edge represents the portion of a rail from one intersection to another. Note: this graph is a grid where the ends of the grid wrap around onto each other.
2. Locate each of the boxes on this grid and remove any edges that pass underneath a box.
3. Apply Dijkstra's algorithm using the upper left hand corner of a box B_i as our starting vertex. Store the shortest path from each of these corners to the upper left hand corner of each box that must be connected to box B_i .
4. Repeat step 3 with a new box B_i , until all the boxes have been wired together.
5. Loop through the stored paths multiplying the length of each path by the weight of the wire per unit length of the wire needed to form the

associated connection. Add these numbers together to get the total weight of the wire used.

2.3 Local Search Algorithm

In order to both minimize the weight of the wire and keep the vehicle balanced, we define a *penalized objective function* that we then try to minimize. To define this function we assign a number as the maximum acceptable center of mass offset. To calculate this penalized objective function (for simplicity, the rating), we:

1. Use the center of mass algorithm already built to determine the center of mass offset.
2. Use the wiring algorithm already described to calculate the weight of the wire.
3. Assign a rating to the arrangement as follows:
 - If the center of mass offset is in the acceptable range simply use the weight of the wire as the rating.
 - Otherwise, we add the weight of the wire to the distance the center of mass is beyond the acceptable range multiplied by a scaling factor k which is given as input to the algorithm.

Thus the rating or penalized objective function is given as

$$\Theta = w + \rho d \tag{2.3}$$

where w is the weight of the wire, d is the distance the center of mass is beyond the acceptable center of mass offset, and ρ is the penalty parameter.

This use of the penalized objective function has multiple advantages. First, it allows the local search algorithm to be applied to box arrangements that are not balanced. It also allows the local search algorithm to be used in multiple ways by varying the penalty parameter ρ . For small ρ , the local search algorithm will minimize the wire weight with very little consideration for the center of mass. On the other hand, by increasing ρ the algorithm tries its best to get a ‘balanced’ placement of boxes.

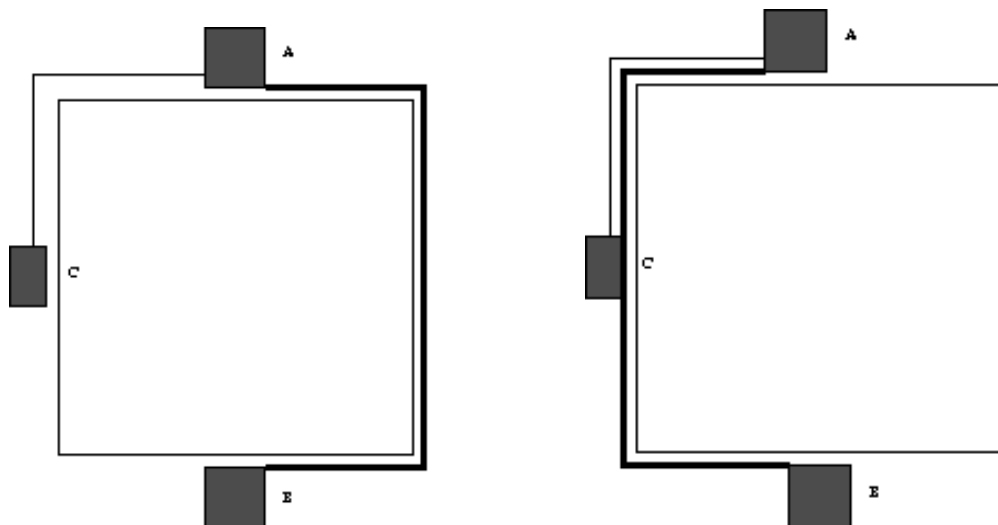


Figure 2.3: Moving the top box to the right is a small improvement, but moving to the left, after rewiring is a significant improvement.

The local search algorithm itself takes as input a placement of boxes and moves each box individually in each of the four cardinal directions to evaluate if such a move will improve the rating. Thus the local search algorithm will check a move to the left even if a move to the right improved the rating. This at first seems counter-intuitive since it seems that if a move in one direction improves things, a move in the other will not; however, this is not necessarily true. To see why, consider the situation where there are two boxes, A and B , directly across from each other that are connected by a very heavy wire w_1 , and another box C on one of the other panels connected to A by a very light wire w_2 . If we move box A away from box C , the heavy wire w_1 is shortened, but w_2 is lengthened by the same amount. Because of the relative masses of the wires, this results in a slight improvement in the total mass of the necessary wire. However, if A is moved toward C , the wiring algorithm will re-route w_1 . Thus both w_1 and w_2 are shortened, so we get a more significant improvement in the wire mass. Note: for w_1 to be improved, it must be rerouted for one of the directions. See Figure 2.3.

We are now ready to describe the local search algorithm.

1. Rate the placement of boxes as given.

2. Select a box. (The selection is made by following the order that the boxes are listed in the placement of boxes as given.)
3. Move the box one square at a time to the right rating the new configuration as above in each step. Continue as long as the new arrangement is improving.
4. Replace the box in its initial position, then move the box one square at a time to the left rating the new configuration as above in each step. Continue as long as the new arrangement is improving.
5. Place the box in the best location found so far, then move the box one square at a time up rating the new configuration as above in each step. Continue as long as the new arrangement is improving.
6. Replace the box to the location it was in before any vertical moves, then move the box one square at a time down rating the new configuration as above in each step. Continue as long as the new arrangement is improving. Note: Boxes cannot move from panel to panel, but must stay on a single panel.
7. Select the best of the arrangements produced in steps 3-6 and place the box there.
8. Select a different box and go back to step 3 until all of the boxes have been moved.
9. Repeat steps 2-8 until one of the following stopping criteria is satisfied. If we have cycled through all the boxes without any improvements or we have reached a maximum number of allowable cycles(given as an input), terminate the algorithm.

2.4 Results

In test cases, the local search algorithm has given significant improvement. As an example, see Figure 2.4. This shows one panel of an arrangement that started out balanced. After local search, the same panel looks like Figure 2.5. In this particular case, both before and after, the arrangement was balanced, but the wire necessary to connect the boxes weighed 146 units before local search and only 115 units after. This was an improvement of 21%.

In an example that started out unbalanced, we started with a center of mass offset of 5.5481 units. Using a penalty parameter of 2, the rating assigned to this example started at 326.0961 before local search and 291.291 after. The wire weight started at 325 units and went to 291 units, and the center of mass offset decreased to 5.1452 which was still unbalanced. However, when we increase the penalty parameter to 10, the rating assigned decreased to 286, the wire weight dropped to 286 units, and the center of mass offset decreased to 4.886 units which was balanced. Interestingly, in this case, when we increased the scaling factor, we not only got into a balanced configuration, but we also improved the wire weight. This means that in this case we escaped a local minimum by increasing the scaling factor.

This result leads to an interesting extension of the local search algorithm. By allowing moves that are harmful to the rating at first and slowly making those moves less likely, we can make it more likely that we will escape a local minimum. This modification is known as *simulated annealing* and is a simplified version of the process used in [4]. The updated algorithm did indeed produce slightly better results, but took significantly longer. Because the updated algorithm was based on the local search algorithm, it required a starting box placement and could not move boxes from one panel to another. Thus, unlike in [4], this algorithm cannot be used on its own.

2.5 Direction for Further Study

The abilities of the local search algorithm may be improved in several ways. These include

- allowing boxes to move from one panel to another,
- allowing boxes to move not only in the cardinal directions, but also diagonally,
- allowing multiple boxes to move simultaneously,
- and/or allowing boxes to rotate.

Allowing boxes to move from panel to panel would increase the likelihood that it is possible to balance an arrangement. It also would increase the amount of flexibility the algorithm would have to decrease wire weight. This change would be a significant improvement if the arrangements that are given

to the local search group are not based on placing boxes on specific panels. If, on the other hand, the arrangements given to the local search group are based primarily on controlling which panel the boxes are placed on, allowing boxes to switch from panel to panel may actually be a negative. We will see later that this is the case for some of the box arrangements given to the local search algorithm.

Allowing the boxes to move diagonally would increase the number of potential arrangements that the algorithm could search. While it is true that a diagonal move is just a horizontal move followed by a vertical move, allowing diagonal moves would still make more arrangements possible. This is because currently if all the boxes are in a row, a vertical move will always increase the amount of wire necessary. A diagonal move, on the other hand, may not. Thus, currently it is extremely difficult for boxes to switch horizontal positions, but allowing diagonal moves would make this much simpler. This change may be a significant improvement if the arrangement of boxes given to the algorithm is based on placing certain boxes on certain panels, but when the box arrangement given to the algorithm is primarily based on putting the boxes in a specific order, this change would not be beneficial.

Since, as we will see, both of these strategies are employed, the ideal improvement to the local search algorithm may actually involve both changes, but in separate algorithms. Thus, it may be ideal to split the local search algorithm into multiple distinct algorithms that can be used as appropriate.

Allowing multiple boxes to move simultaneously would make it easier for the local search algorithm to make improvements in wire weight without affecting the center of mass as much. This is because moving boxes of similar mass on opposing panels in opposite directions will have almost no effect on the location of the center of mass. Thus, this change could make a very significant difference if the center of mass must fall within a small tolerance. This change, however, would have the downside of dramatically increasing the length of time it takes to run the algorithm.

Allowing boxes to rotate has several advantages. One advantage is that if boxes can rotate, there is more flexibility in the way boxes can be placed on a panel. Also, allowing boxes to rotate would be very significant if the wiring algorithm were changed to take into account the location that wires connect to the individual boxes rather than simply connecting to the upper left hand corner.

Chapter 3

Some Best First Guesses

Deborah Fisher
Jason Lathrop
Breeann Tonnsen

3.1 Introduction

The goal of the math clinic is to design a flexible and effective computer algorithm for the Electrical Box Placement and Wiring Problem (EBPWP). The EBPWP is focused on optimizing the placement of various devices, represented as boxes, along with connecting wires on a group of panels within a spacecraft where placement is subject to various design constraints. The solution to the problem minimizes the mass of the wires connecting the boxes while maintaining a balanced configuration for different device combinations or requirements.

The structure of the panels can be conceptualized as the surface of a rectangular prism without the top and bottom. The center of mass needs to be located along the rectangular prism's center axis. The panels are represented by a square grid system with specified dimensions. The devices are represented by rectangular boxes with dimensions defined as integer multiples of the panel grid spacing (i.e. the smallest box is the size of one square of the grid system). The boxes are of various sizes and masses and are connected to other boxes along the grid system by wires of defined mass per unit length. The placement of boxes is subject to the constraints that there must be a distance of at least 1 grid unit between boxes and boxes cannot occupy the

same space. Boxes can be flush against the corner of a panel, as long as there is at least 1 grid unit between it and any box on the next panel.

In order to develop the algorithm, the Math Clinic divided into three teams each with a primary initiative: Team 1 has been developing local search algorithms for finding locally-optimal solutions from a given initial box configuration; Team 2 has been responsible for creating the heuristics to generate good starting configurations, since the quality of the solution from the local search algorithm depends critically on the starting configurations; Team 3 has been building metaheuristics to search for globally optimal solutions. This paper elaborates on the responsibilities of Team 2.

There has been much discussion on how to solve the problem of finding an initial box configuration. A literature search was undertaken and we did not find any methodologies or processes related to this particular problem. Our team decided to use an "educated guess" for the initial box placement instead of using a random placement. We feel it is important to eliminate many grossly sub-optimal box arrangements that would be generated by random placement in order to save computation time. We created an algorithm that uses the input information, analyzes it, and generates a reasonable initial configuration of the boxes to use as an input into the local search algorithm.

3.2 Main Results

The main idea of our algorithm is to create circular lists of the boxes. Each list is then used to create an initial box layout in such a way that boxes are close together on the list will be close together in the layout. These initial box layouts will then serve as starting configurations for the local search algorithm developed by Team 1. In the following sections we describe our algorithm for creating the circular list and our algorithm for translating each circular list into a box layout that attempts to place the center of mass near the center of the rectangular prism. Following these descriptions, we describe our rationale and give some results of our algorithm. Figure 3.1 is an example of a circular list and Figure 3.2 is a possible corresponding grid placement.

3.2.1 Description of Algorithm:

The input to the algorithm will include a symmetric matrix that gives the mass of the connections between boxes. Label these boxes down the side of

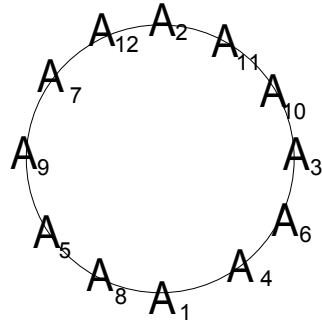


Figure 3.1: An example of a circular list of 12 boxes

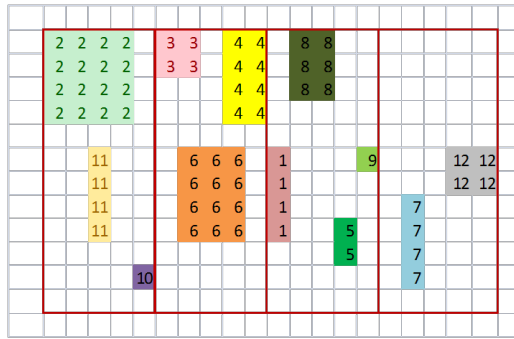


Figure 3.2: A possible box configuration based on the above circular list

	A_1	A_2	A_3	A_4	A_5	A_6	A_7
A_1	0	1	4	1	0	7	3
A_2	1	0	0	2	4	0	0
A_3	4	0	0	2	0	8	2
A_4	1	2	2	0	5	3	0
A_5	0	4	0	5	0	1	1
A_6	7	0	8	3	1	0	2
A_7	3	0	2	0	1	2	0

Figure 3.3: An example of an input matrix that gives the mass of the connections per grid unit for seven boxes

this matrix as A_1, A_2, \dots, A_n . We will call row i the *connection vector* for box A_i , since it contains the mass per unit length of the connections to this box. Figure 3.3 is an example of a connection matrix.

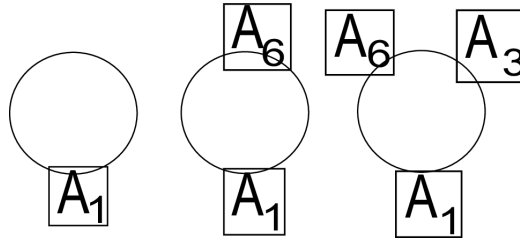


Figure 3.4: The progression of the first three box placements in the example

Step 1 Create a circular list of the boxes based on minimizing wire mass

- 1.1 First choose the box object A_1 , and place it in a circular list.
- 1.2 Next, analyze the weight of the wire connections from A_1 to other boxes by finding the maximum of the terms in the connection vector for A_1 . Choose the box connected to A_1 with the heaviest wire and place it next to A_1 in the circular list. In our example, the connection vector for A_1 is $[0, 1, 4, 1, 0, 7, 3]$. Therefore, A_6 has the heaviest wire connection to A_1 , so it will be placed next. Since there is only one box in the circular list, A_6 is placed next to it.
- 1.3 Our circular list now contains two boxes. Next, determine the box that is connected to the boxes in the list by the heaviest total wire mass. To identify this box, add the connection vectors of the boxes in the list and find the maximum term excluding boxes already placed. For our example we add the connection vectors for A_1 and A_6 : $[0, 1, 4, 1, 0, 7, 3] + [7, 0, 8, 3, 1, 0, 2] = [7, 1, 12, 4, 1, 7, 5]$. Then A_1 and A_6 are replaced by zero in the summed vector: $[0, 1, 12, 4, 1, 0, 5]$. The highest number remaining is for box A_3 . Therefore, A_3 is the third box to be placed, because it has the heaviest total wire mass connecting it to the two previously placed boxes. In the circular list, the only possible placement for the third box is between the first two boxes. Place it there. Figure 3.4 gives the progression through the first three box placements of our example.
- 1.4 Add the connection vectors of box A_3 to the summed vector, zero out all the previously placed boxes, and find the maximum term in the sum of this new summed connection vector. Again, this

maximum will be the box with the highest total wire mass connecting it to the boxes in the circular list. In our example, we add $[0, 1, 12, 4, 1, 0, 5] + [4, 0, 0, 2, 0, 8, 2] = [4, 1, 12, 6, 1, 8, 7]$, zero out the previously placed boxes $[0, 1, 0, 6, 1, 0, 7]$ and find the maximum term excluding the terms for A_1, A_3 and A_6 . This tells us that box A_7 will be placed next.

- 1.5 Determine where to place the box in the circular list. If there are n boxes on the circular list, then there are n possible locations for the new box, each one being between any two adjacent boxes. To find the best location, we check each possible placement to find the minimum wire weight relative to our list. In our example, there are three boxes in the circular list, so there are three places that A_7 could be placed: between A_1 and A_3 , between A_1 and A_6 , or between A_3 and A_6 . We measure the total wire mass in each possible location for A_7 . To do this we find a relative distance between any two boxes. If the boxes are adjacent in the circular list, then their distance is 1. If they have 1 box between them, their relative distance is 2, etc. Since this is a circular list, the relative distance between two boxes could be found in two ways: moving clockwise from the first box to the second or moving counterclockwise from the first box to the second. For example, the distance between A_1 and A_3 in Figure 2 is 2 in the counterclockwise direction and 1 in the clockwise direction, both starting from A_1 . The lesser of these two distances will be chosen. We then multiply this relative distance by the wire mass per unit length of the connection between any two boxes. This gives us the wire mass of a given location. In our example, if we place A_7 between boxes A_1 and A_3 in our circular list, then the distance between box A_1 and A_7 is 2. The wire mass per unit length is 4, so the calculated mass between these boxes is 8 for that location. We continue this exercise for every pair of boxes in the circular list. We calculate a total mass of 32 if A_7 is placed between A_1 and A_3 , 37 between A_6 and A_3 , and 35 between A_6 and A_1 . After totaling the wire mass for each location, we place A_7 in the location with the minimum total wire mass, which in this example is between boxes A_1 and A_3 .

- 1.6 We repeat steps 1.4 and 1.5 until all of the boxes are placed in the

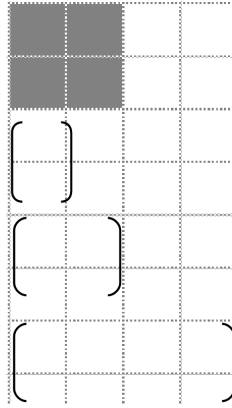


Figure 3.5: Three possibilities for box size versus mass interval width, which are presented as parentheses

circular list.

Step 2 Place boxes on the panels based on the mass of the box

Now that we have a relative ordering of the boxes, we need to place them on the panels. Since we want the center of mass close to the center axis of the panels, we attempt to evenly distribute the mass of the boxes along the panels.

- 2.1 First we want to calculate the mass per horizontal grid unit, λ . To do this, we first sum the mass of all the boxes, $\sum_{i=1}^n m_i = M$, and find the total number of horizontal grid units on all of the panels, call it the circumference C . The mass per horizontal grid unit is the ratio of total mass to total grid units, $\lambda = \frac{M}{C}$.
- 2.2 Next for each box we calculate the mass interval width, which will tell us the number of horizontal grid units each box should have around it to evenly space the mass of the boxes. To find this width we take the mass of box i , m_i , and divide it by λ , $w_i = \frac{m_i}{\lambda}$. The mass interval width may be greater than, equal to, or less than the actual width of the box. (see Figure 3.5)
- 2.3 To start placing boxes on the grid, we first orient each box so its vertical grid displacement is greater than or equal to the horizontal

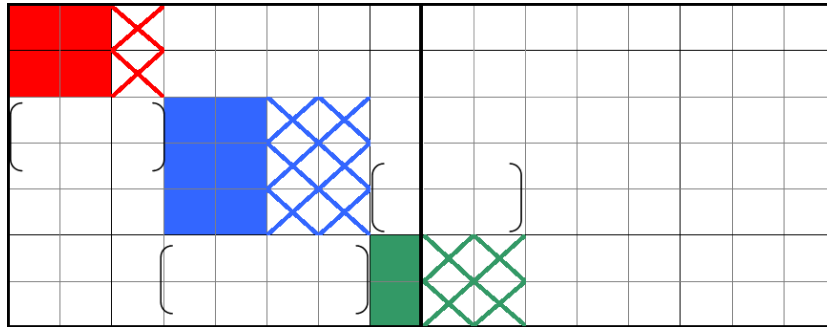


Figure 3.6: Example of the placement on the grid of three boxes, whose mass interval widths are all greater than their size

grid displacement. Next we choose a first box from our circular list to place, A_1 . We place this box in the upper left hand corner of the grid.

- 2.4 Every subsequent box, A_i , is placed with its left edge at the end of the mass interval width for the previous box. The algorithm then scans the column and places the box in the set of rows that will maximize the available space to the right and left. (see Figure 3.6.) If the interval width of the previous box is less than the actual width of the previous box, then box A_i must be placed below or above the previous box if room is available. If there is no room above or below the previous box, the algorithm will advance to the next column and try again. (see Figure 3.7.)
- 2.5 To place the rest of the boxes, we repeat step 2.4 each time for each box. A flowchart describing the algorithm for the initial placement of a set of boxes from a single circular list is represented in Figure 3.8.

Step 3 Divide the grid into four panels

Once the boxes have been placed on the grid, the next step is to divide the grid into fourths, which will represent the panels. Our algorithm finds as many divisions as possible and checks each configuration.

- 3.1 We make the first panel division at the start of the grid. We then check the panel divisions for boxes that straddle corners. If there

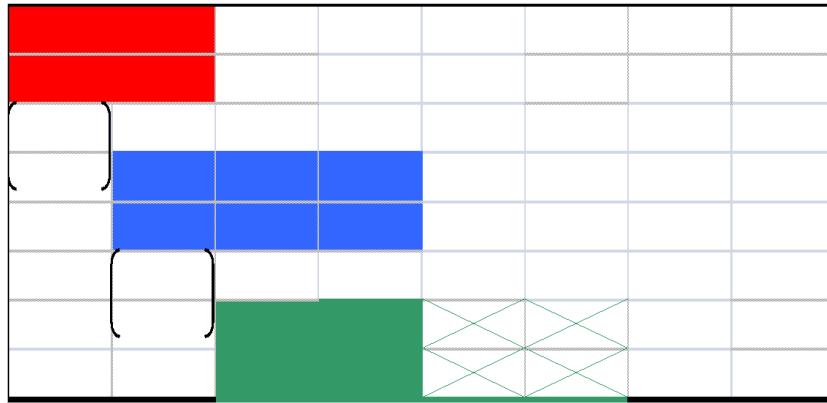


Figure 3.7: Example of the placement on the grid of three boxes. The first two have mass interval width less than the size of the box, and the third is greater

is no box straddling a panel division then we move to the next panel division.

- 3.2 If there is a box straddling the panel division, then we check to see if the conflict can be resolved by moving the box horizontally onto one of the panels. If the box could be moved to both panels, then we check the total mass of opposing panels. We place the box on the panel that gives a smaller difference in the mass of opposing panels.
- 3.3 If the box can only be moved to one panel, then we place it there. Once the box had been moved, we check the next panel division for boxes.
- 3.4 If the box cannot be moved to either panel, then we temporarily leave it in its current location. We then check the next panel division. If there is a problem, we repeat steps 3.1 to 3.3 to attempt to move the box off the panel division. We attempt to move all boxes on panel divisions, and then come back to the boxes that could not be moved. It is possible that by moving other boxes we have created space for a problem box to move, so we try to move it again. If we check every panel division and none of the problem boxes can be moved, then we discard that panel division.
- 3.5 Once we have a feasible panel division, we call the local search

algorithm to improve the configuration.

3.6 Next, we repeat the steps 3.1 to 3.5 for each column from the first to $\frac{C}{4}$. Once we have checked those panel divisions, we have checked them all since they start to repeat. Again, we call the local search algorithm for each feasible configuration. Figure 3.9 describes the algorithm for placing boxes onto the grid. Figure 3.10 represents two possible panel divisions for one circular list, including the movement of boxes to make them fit.

Once we have called the local search algorithm for each feasible panel division, we then repeat this process by going back to step one to create an entirely new circular list. However, we place A_2 in the circular list first instead of A_1 . All three steps will be repeated n times, once for each box being placed in the circular list first.

Rationale of Algorithm:

- Step 1 Step one creates a relative ordering of boxes that is based on minimizing the mass of wire connections at each step. First the decision of which box to place is based on the heaviest sum of the wire mass that connects the box to the circular list. This will prioritize the placement of the boxes with heavier connections to the boxes already in the list, which will lead to boxes with heavier connections being closer together on the grid. Next, we check the wire mass of each placement for the new box. This minimizes the wire mass of placing the new box into the list.
- Step 2 Once we have our circular ordering, we want to distribute the weight as evenly as possible while preserving our ordering. To do this we calculate the approximate amount of box mass that we want on each horizontal grid unit. For placement purposes, we only consider box mass, not wire mass. If the box mass is high, then we give that box a cushion of empty grid spaces around it to average out the mass in that area. If the box mass is small, then we will put multiple boxes in a column to average out the mass in that area. Spreading out the boxes among the rows has two benefits. First, it gives the local search algorithm more room for movement in order to find a local minimum. Second, it ensures that the algorithm that divides the grid into panels will have room to make adjustments, so that boxes do not end up on the corners.

Step 3 Our algorithm checks each possible panel division, so the optimal one won't be missed. For each division, we make small movements in the box placement to facilitate the division. The local search algorithm does not move boxes from one panel to another. Therefore, different panel divisions could have drastically different results in the end.

3.3 Results

We have run many test cases that compare our algorithm for starting configurations to random placements for a starting configuration. In every case after running the local search algorithm, our configuration gives a better rating and often a better center of mass. In most cases, the improvement was significant. For example, we started with a small example consisting of 5 boxes. This example had a connection matrix of

$$\begin{pmatrix} 0 & 0 & 12 & 3 & 8 \\ 0 & 0 & 0 & 0 & 12 \\ 12 & 0 & 0 & 0 & 10 \\ 3 & 0 & 0 & 0 & 6 \\ 8 & 12 & 10 & 6 & 0 \end{pmatrix},$$

and box weights $(329 \ 264 \ 334 \ 286 \ 142)$, and box sizes

$$\begin{pmatrix} 4 & 1 & 3 & 2 & 3 \\ 4 & 4 & 4 & 3 & 1 \end{pmatrix}$$

. We called the local search algorithm with both a random configuration and the best configuration that resulted from our algorithm. With both we used a penalty parameter of 3 and a center of mass error of 3. Recall that calling the local search algorithm with a large penalty parameter will focus it on getting the configuration into balance, and calling it with a low penalty parameter will focus it on minimizing the wire mass. While the wire weight and rating for the random placement was 336, the wire weight and rating of our configuration was 290. Not only did we have better wire weight, but our center of mass was .2633 off center and the random placement was 2.237 off center.

Another test case had a connection matrix of

$$\begin{pmatrix} 0 & 1 & 0 & 2 & 1 & 0 \\ 1 & 0 & 1 & 2 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 2 & 2 & 1 & 0 & 2 & 1 \\ 1 & 0 & 0 & 2 & 0 & 2 \\ 0 & 1 & 1 & 1 & 2 & 0 \end{pmatrix},$$

, and box weights (200 400 200 200 100 200), and box sizes

$$\begin{pmatrix} 2 & 1 & 3 & 2 & 2 & 1 \\ 2 & 1 & 3 & 2 & 2 & 2 \end{pmatrix}.$$

For this test case, we again called the local search with our best configuration and a random configuration. This time we called both configurations with a large penalty parameter of 30 and a center of mass error of 5, and then again with a smaller penalty parameter of 6 and a smaller center of mass error of 2.5. In both cases, our configuration had a better rating than the random placement. In the first case, our rating and wire weight was a 274 and the random rating and wire mass was a 301. Plus, our error was 2.8833 and the error on the random configuration was 4.57 In the second case, we got the same configuration, but now we were not within the center of mass error, so our rating increased to 276.3. The random placement was improved to a wire weight of 295, but the rating increased to 301.34 since the center of mass error was 3.557. We also did larger examples, such as with a connection

matrix of

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 2 & 1.5 & 0 & 1 & 0 & 0 & 2 & 1 & .5 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 2 & 0 & 1 & 1 & 3 & 1 & 0 & 0 & 2 & 0 & 1 \\ 2 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 2 & 1 & 0 & 1 & 2 \\ 1.5 & 1 & 0 & 1 & 1 & 0 & 2 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 3 & 1 & 2 & 0 & 1 & 2 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 2 & 0 & 0 & 2 & 1 & 2 & 1 & 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 1 & 0 & 1 & 2 & 0 \\ 2 & 0 & 1 & 2 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ .5 & 1 & 2 & 1 & 2 & 1 & 0 & 0 & 2 & 0 & 1 & 1 & 0 \end{pmatrix}, \text{ and box weights}$$

(200 250 300 150 400 100 100 300 100 250 200 200 200),
and box sizes

$$\begin{pmatrix} 2 & 1 & 2 & 2 & 3 & 4 & 1 & 3 & 1 & 2 & 2 & 1 & 1 \\ 2 & 2 & 1 & 1 & 3 & 1 & 4 & 2 & 1 & 3 & 1 & 2 & 1 \end{pmatrix}.$$

Again with this larger example, our configuration beat the random configuration, with a difference in rating and wire weight of 665 to 703.5, and a difference in center of mass error of .4198 to .7528.

3.4 Conclusions

First we would like to explain what we consider to be good results. Compared to a random placement of boxes, we would hope our algorithm would generate less wire mass. Additionally, we would hope that our solutions would be better balanced than random solutions. We ran multiple test cases comparing our placement algorithm to a random placement. With each configuration we then called the local search algorithm to make improvements. In every case, our algorithm produced better results than the random placement.

We have some suggestions for improvement to this algorithm. One of these is altering it so it generates more than n circular lists. To do this, we could save the data whenever there is an arbitrary decision to be made. For example, when deciding which box to place next on the circular list, there could be two boxes that have the same wire mass. Each one could be saved to generate more variations of a circular list. Plus, when deciding which location to put a box on the list, there could be a tie between locations. In these instances, we could save the current configuration and then make a choice. After the circular list is complete, we can go back to the saved configuration and continue executing the algorithm with the other choice in the tie. This may result in more circular lists.

To generate more box placements, we could rotate the boxes when they are initially placed on the grid. We could also rotate boxes when moving them off the corners or when shifting them on panels. This would generate more box placements with which to call the local algorithm. We could call the local algorithm every time our algorithm generates a configuration, but this would slow the run time significantly.

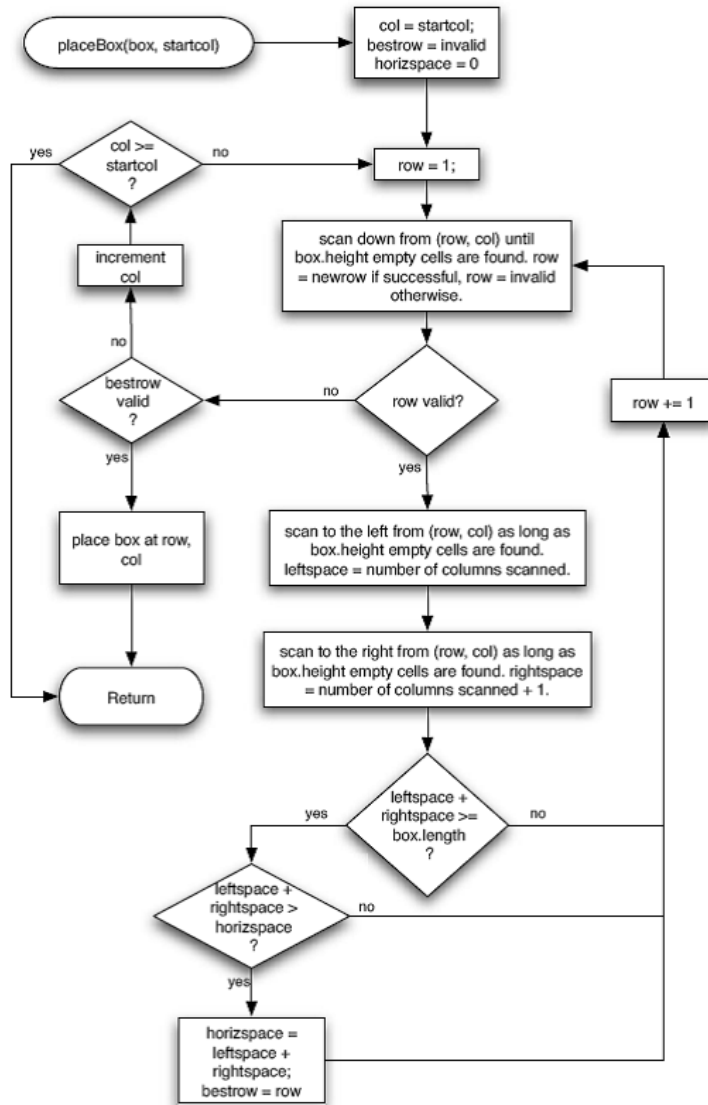


Figure 3.8: A flow chart of the algorithm that places boxes onto grid

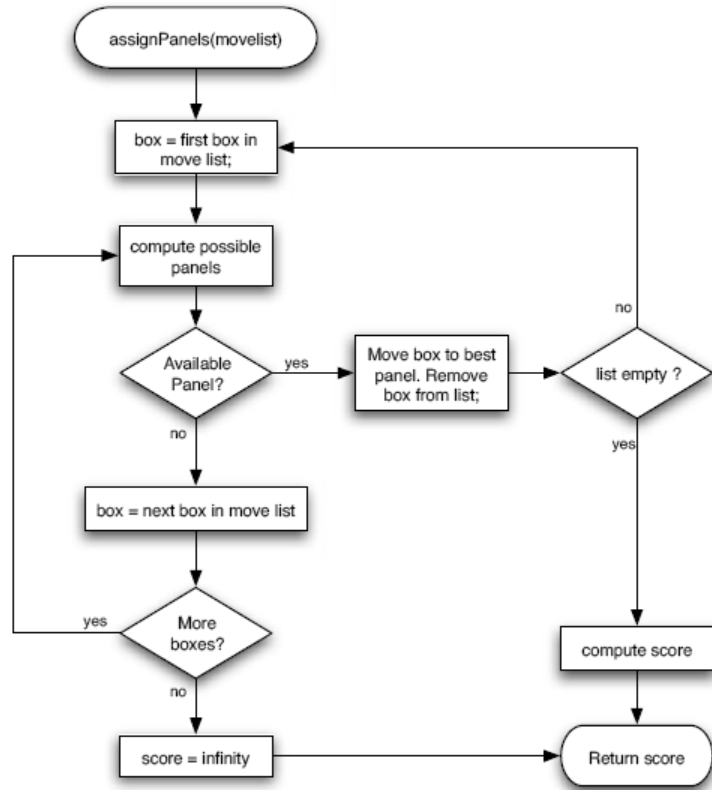


Figure 3.9: A flow chart of the algorithm that places boxes onto grid

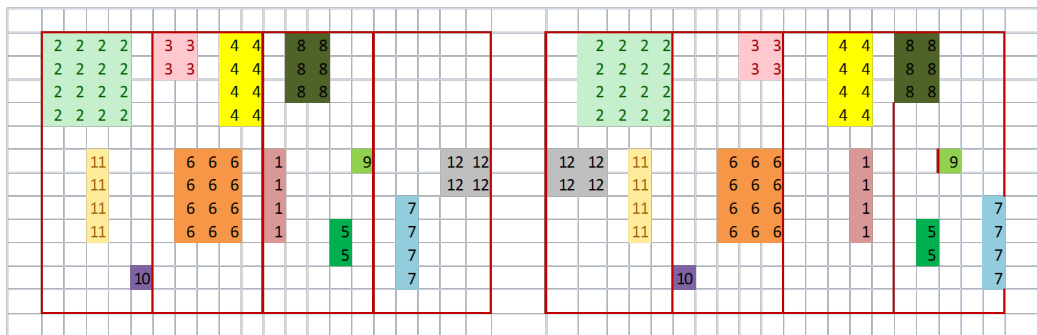


Figure 3.10: Given one circular list and placement on the grid, these are two possible panel divisions including the movement of boxes to make them fit

Chapter 4

A Graph-Combining Scatter Search Algorithm for Avionics Box Placement

Christopher Aquinto
Michelle Rendon
Dmitriy Vassilyev

4.1 Introduction

When sending vehicles into space it is important to keep the mass to a minimum while also keeping the vehicle's center of mass in balance. It may be obvious why one would want to keep the center of mass in check, but the reasoning behind keeping the weight to a minimum is due to the fact that it costs around \$14,000 for each kilogram that is sent into space. The avionics box placement problem is one in which optimal box placement must be considered in order to minimize wire length from box to box, minimize wire mass (considering that some wires are heavier than others), and so that the boxes center of mass are in equilibrium. Our team has been given the task of developing a metaheuristic to search across a large solution space to find solutions that will fit our constraints given to us by our sponsor United Launch Alliance.

From these requirements we have designed and implemented a Scatter Search (SS) and Graph combining algorithm to solve the Avionics Box Place-

ment Problem (ABPP). This algorithm approaches Scatter Search methods in a novel manner, operating on fixed-arc graph structures. In the following section we give a brief introduction to Scatter Search and an explanation as to why we choose this method. We then discuss in Section 4.3 our algorithm, which is based on Scatter Search, and which incorporates a novel solution combination method called Graph Combining. Testing results are given in Section 4.4 and finally conclusions and future work are discussed in Sections 4.5 and 4.6.

4.2 Background: Scatter Search

The SS algorithm maintains a population of solutions called the “reference set”. At each iteration many new solutions are generated based on combining two or more solutions from the reference set and some of these replace members of the previous population. The population discussed previously includes the best solutions found so far as well as diverse solutions. These diverse solutions are selected to be as different as possible from other members of the population. Then new solutions will be created by combining information from small subsets of solutions from the previous population. When creating new solutions, it is important to use a method which will retain common qualities of the solutions being combined. In this way, the new solutions will be more likely to be good than randomly generated solutions. Often new solutions are created by using linear combinations of the subsets of solutions, but this will not be possible for our problem.

A simple outline of a general SS algorithm is as follows:

- 1) Create an initial random population of solutions which typically consists of 20-50 number of populations
- 2) Evaluate and improve upon the population (for example, by using local search or some other heuristic)
- 3) Choose from population to create a small set of solutions called the reference set
 - The reference set contains two groups of solutions, good and diverse
 - * Good solutions are superior to the others based on some measure of quality (typically, the objective function).

- * Diverse solutions are chosen one at a time from the initial population to maximize the distance to the other members of the reference set. Once the solution is chosen, it is added to the reference set, so that the next "diverse" solution must be far away from both the "good" solutions and all previously identified diverse solutions.
- 4) Randomly choose solutions from the reference set and combine them to create new solutions
- 5) Evaluate and improve upon these new solutions and make a new reference set from the combined solutions
- 6) Repeat steps 3-5 until some stopping criteria has been met (such as when a maximum number of iterations is reached)

Scatter Search (SS) is a metaheuristic that is different than many other evolutionary methods because it does not depend as heavily on randomness as other methods. Instead, Scatter Search incorporates diversity explicitly. While SS does use some randomness in its method it is not entirely based around randomization. The point that makes SS so special is its use of diversity. Scatter Search's use of diversity should keep the method from only finding local minimum or maximum and bringing it closer to perhaps finding a global maxima or minima. Scatter search has also proven to be very successful in solving a wide variety of very difficult optimization problems.

4.2.1 Choice of Scatter Search and Graph Combining

The reasoning behind our team's choice of SS is essentially that we found it to be superior to the other evolutionary search methods. There have been many studies in which it has been found that SS works faster and better than other methods such as Simulated Annealing and Genetic Algorithms. Some methods that could have been used are TABU search [3], Simulated Annealing[4] and Genetic Algorithms (GA)[5]. Simulated Annealing and GA, while different from one another are both more dependent on using randomness. TABU is a method in which solutions or moves would be put on "taboo" lists and never be allowed to be used for at least some specified number of iterations. GA is similar to SS in that it mates solutions together

to create new ones, but it also only deals with randomness when creating its initial population and then from there chooses only to mate good solutions.

Also with concern to our combination of solutions, we chose not to use linear combinations since it would not work with our solutions. Instead we implemented Graph Combining to create new solutions. Graph combining is a method which will incorporate preserving good qualities from solutions while also allowing for diversity. Once again the method of Graph Combining will be discussed in much more detail in Section 4.3.2.

4.3 Scatter Search and Graph Combining Algorithm

In order to implement Scatter Search for our problem, we must first discuss some main ideas:

1. How solutions are represented.
2. How solutions are combined.
3. How boxes are placed on each panel.
4. How the reference set is created.
5. How solutions are evaluated.
 - How "best" solutions are chosen.
 - How "diverse" solutions are chosen.
6. How the starting population is generated.

4.3.1 Solution Representation

For this problem our solutions will be represented graphically and will not represent the wire connections between boxes. In Figure 4.2 a simple example solution is shown. Within the computer, the solution is represented by an assignment of each box to a panel, and the grid location of the box within the panel. In order to describe how solutions are combined and how diversity is defined, we need to discuss distance matrices, which provide a method for describing important features of solutions. For each pair of boxes, we define

$$\begin{bmatrix} - & 1 & 2 & 0 & 1 \\ 1 & - & 1 & 1 & 0 \\ 2 & 1 & - & 2 & 1 \\ 0 & 1 & 2 & - & 1 \\ 1 & 0 & 1 & 1 & - \end{bmatrix}$$

Figure 4.1: Distance Matrix

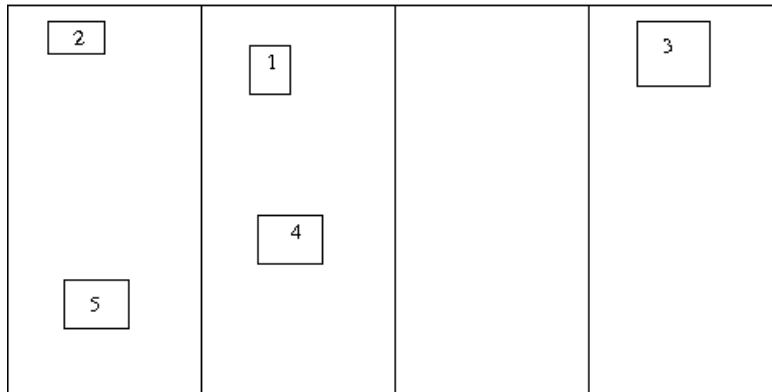


Figure 4.2: Solution Representation

the distance between the boxes based on which panels they lie on. Let $p(i)$ denote the panel that box i lies on. Then the distance between boxes i and j is defined as follows $d(i, j) = |p(i) - p(j)| \bmod 4$. The distance matrix for a solution is the matrix $D = (d(i, j))_{i, j}$ i.e. the matrix whose $(i, j)^{th}$ entry is $d(i, j)$. An example distance matrix is shown in Figure 4.1, where each number in the distance matrix represents the number of panels separating the boxes. A zero signifies that the boxes are on the same panel, a one signifies that the boxes are one panel away from each other and a two signifies that the boxes are two panels away from each other. [Note: The distance matrix does not depend on the actual panel assignments - only the relative panel assignments. So for example, if we moved every box one panel clockwise, the resulting solution would have the same distance matrix.]

Our solution representations will take on two phases. Phase I will give

$$\begin{bmatrix} - & 1 & 2 & 0 & 1 \\ 1 & - & 1 & 1 & 0 \\ 2 & 1 & - & 2 & 1 \\ 0 & 1 & 2 & - & 1 \\ 1 & 0 & 1 & 1 & - \end{bmatrix}
\begin{bmatrix} - & 1 & 1 & 1 & 1 \\ 1 & - & 2 & 2 & 0 \\ 1 & 2 & - & 0 & 2 \\ 1 & 2 & 0 & - & 2 \\ 1 & 0 & 2 & 2 & - \end{bmatrix}
\begin{bmatrix} - & 1 & & & 1 \\ 1 & - & & & 0 \\ & & - & & \\ & & & - & \\ 1 & 0 & & & - \end{bmatrix}$$

Figure 4.3: Combination of Distance Matrices

box assignments to panels, which is described in Section 4.3.2. Phase II will give actual coordinates of the boxes, explained in Section 4.3.3.

4.3.2 Combining Solutions: Graph Combining

To understand the algorithm it is first important to understand the graph combining process. Represented in Figure 4.2 is a grid with four panels and box placements which correspond to the distance matrix given in Figure 4.1. As can be seen, there will never be an instance when there are 3 or more panels separating boxes due to the fact that panels 1 and 4 are actually connected.

Our method for combining two solutions begins by constructing a graph that represents similar features in the solutions. First we must select which solution is to be our “leading” solutions and which is to be our “regular” solutions. A leading solution is defined as having certain requirements that make the box placement more optimal as compared to the “regular” solution, these requirements will be given at the end of this section. Each node of the graph represents a box, and an arc is defined between two nodes if the distance between the corresponding boxes is the same in both solutions. Each arc in this graph represents a distance relationship between two boxes that will remain fixed in the new solution. Figure 4.3 represents two distance matrices and their combination matrix with fixed arcs.

In Figure 4.3 the blank spaces in the combination matrix represent distances that differ between the two matrices. These blank spaces can be filled in with any distances in the resulting solution, since the only distances that matter are the ones that are fixed. To represent the idea behind the combination of two solutions, Figure 4.4 gives graphs of the leading and regular distance matrices for Figure 4.3, in a box layout format and with the fixed

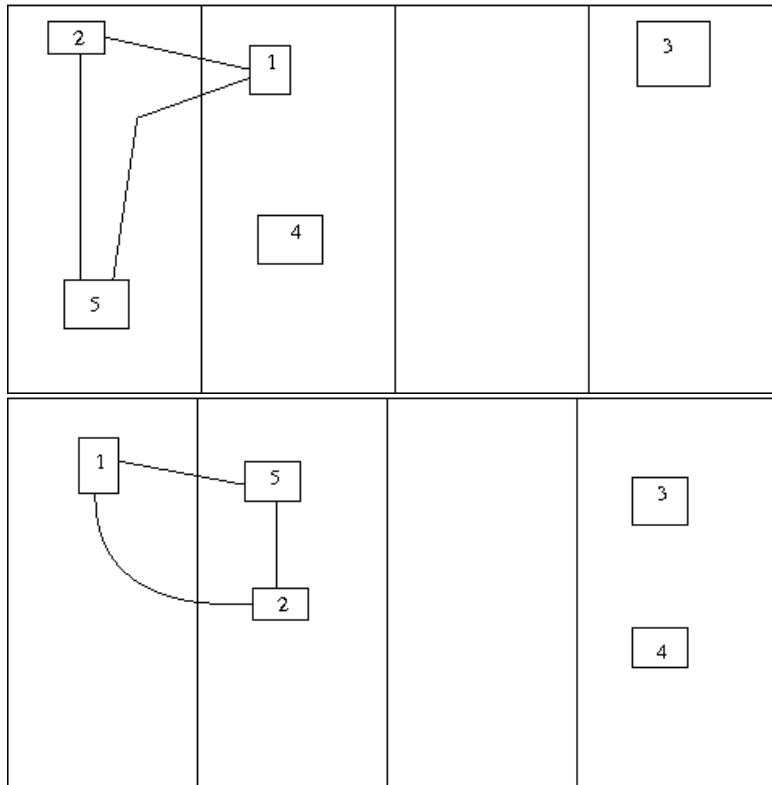


Figure 4.4: Solution Representation

arcs between them. One combination is shown in Figure 4.5, this combination was done by simply moving box three into the third panel which makes the regular solution one step closer to the leading solution.

The combining process begins when the leading and regular solutions are compared to one another. The next step will be to create a "path" of intermediate solutions by moving boxes from the regular solution to different panels, in order to construct more similarities between the "regular" and "leading" solutions. Each time a box is moved, all of the boxes connected by arcs must also be moved in order to guarantee that all fixed distances remain fixed. After the box is moved, the combination graph is updated to represent the similarities between the new solution and the leading solution. These new arcs are then fixed, and the process is repeated. If repeated enough times, this process would create a path of solutions ending at the leading solution. If

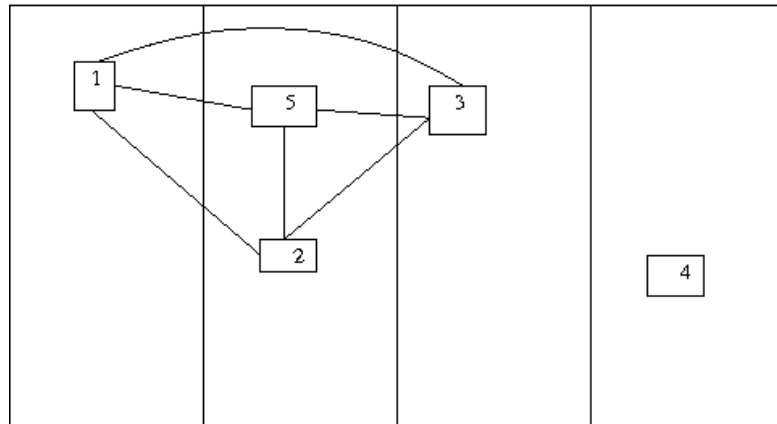


Figure 4.5: Combination Solution

the leading solution is reached before the specified number of iterations then the algorithm will stop there. One of the intermediate solutions along the path from the “regular” solution to the “leading” solution is selected as the new solution. Since there is no need to actually create all of the intermediate solutions on the path from “regular” to “leading”. So, we randomly choose a number of iterations to take and stop after that many iterations have been taken.

4.3.3 Phase II Box Placement

Now that we have panel assignments, we need to create actual coordinates for each box. The Box Placement algorithm takes a “Greedy” approach when placing boxes, this means that once a box is placed it will not be moved.

- **Step 1** Get coordinates from both parents for each box on the panel [Note: Boxes may be on different panels in the parent solutions, but this does not matter because position of box is defined by its height and width on a panel and all panels are the same size.]
- **Step 2** Define a neighborhood for each box
 - The neighborhood is the smallest rectangular area containing the locations of the box on both parent panels (see Figure 4.6)

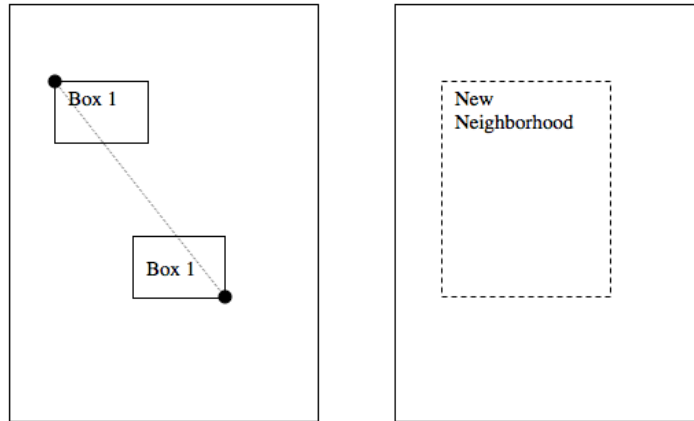


Figure 4.6: Box Placement

- **Step 3** Loop over all neighborhoods and start from the neighborhood with the smallest area
- **Step 4** Place the new box within the neighborhood while ensuring that there is at least one grid space separating it from and previously placed boxes.
 - First try to place the box on one of the original corners
 - If the box cannot be placed go over all grid crossing points within the neighborhood trying to place the box
- **Step 5** If the box cannot be placed anywhere within the neighborhood then the solution is infeasible; discard this solution and start over with a new solution
- **Step 6** If the box can be placed, return to Step 3
- **Step 7** Once all boxes are placed then stop

Figure 4.6 shows a defined neighborhood given two original boxes.

4.3.4 Selection of the Reference Set

The reference set consists of two sets. The two sets are "best" solutions and "diverse" solutions. The creation of the "best" solution is based on the concept of Pareto optimality. Each solution is given two scores calculated by the local search algorithm, weight and distance from the center of mass. Let m_i denote the wire mass of solution i , and let P_i denote the center of mass score. So j dominates i if either $m_j \leq m_i$ and $P_j < P_i$ or $m_j < m_i$ and $P_j \leq P_i$. The set of solutions that are not dominated by any other solutions are said to be "pareto optimal". This idea is illustrated in Figure 4.7. In this figure, the scores of each solutions is represented graphically as a point. Point j dominates point i if point i lies above and not left of point j , or to the right and not below point j . The non-dominated solutions are plotted in solid black. These solid black points make up the "pareto frontier".

Since we wanted to start with populations that are more concerned with wire weight, then we start with a low penalty parameter and increase the parameter with each iteration. This causes us to have more balanced solutions in our final populations. Diversity is measured based on the distance matrix, defined in Section 3.1, we first need to specify how we measure the distance between two solutions. The distance between two solutions is defined by the number of entries in their respective distance matrices that are different. Diverse solutions are classified by comparing distances to the solutions in the current reference set. The solutions that are the least similar to the members of the current reference set will be considered diverse. By least similar we mean that a new member of the diverse set will have the least number of common arcs with the other members of the reference set.

4.3.5 Creation of New Populations

We will have two solutions and we will create new solution which will keep the best qualities of both parent solutions. We will create a path of intermediate solutions from the "regular" solution to the "leading" solution, such that each new solution will have at least one more similar arc than the previous "regular" solution.

- **Step a** Randomly choose two solutions from the initial Reference Set
- **Step b** Decide which is the regular solution and which is the leading solution.

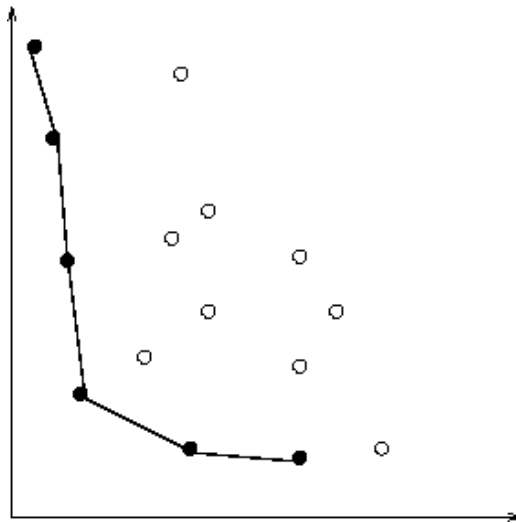


Figure 4.7: Pareto Frontier

- **Step c** Determine the number of steps we would like to take to get from the regular solution to the new solution in the direction of the leading solution
- **Step d** Use the graph combining algorithm described in Section 4.3.2 to define the panel assignments for the boxes for the new solution.
- **Step e** Place the boxes on the assigned panels as described in Section 4.3.3.
- **Step f** Repeat steps a through e until a sufficient amount of new solutions have been generated.
- **Step g** Call the local search algorithm to improve each of the new solutions and to calculate the wire mass and center of mass penalties for each new solution.

4.3.6 Stopping Criteria and Selection of the Best Solution

Our stopping criteria is defined as follows:

- Chosen number of iterations has been met
- Progress is no longer being made (the Pareto frontier has not changed for some given number of iterations)

- Time has ran out

If one of these criteria has been meet then we will choose the best solution, which is balanced and has the least total wire weight.

4.3.7 Initialization

There are a number of user specific parameters that control the execution of the algorithm:

- P = the number of solutions in each population
- R = size of the reference set
- D = number of diverse solutions in reference set
- G = number of good solutions in reference set

Select search parameters: number of solutions in random populations. Now generate P random solutions. These solutions will only give which panel the boxes are on and not there specific placement on the grid. Create initial population of solutions from generated random solutions.

4.4 Results

The algorithm was tested using test cases consisting of 8 boxes, 6 boxes and 12 boxes. For the first test case, consisting of 8 boxes, the data is given in Table 4.2. The second test case, consisting of 6 boxes, had a population size of 20 solutions, with 3 "best" solutions and 3 "diverse" solutions. For the last test case, consisiting of 12 boxes, the data is given in Table 4.1. Figure 4.8 represents the progress made by the first test case, Figure 4.9 represents the second test case and Figure 4.10 represents the third test case. In these figures the dotted green lines and the thick black line represents our pareto frontier at each iteration of the algorithm. The vertical dotted blue line represents the acceptable distance from the center of mass, anything to the left of this line is deemed acceptable. Our algorithm is working, seeing as the pareto frontier is moving closer to the origin i.e. the optimal solution.

Figure 4.11 and Figure 4.12, where Figure 4.11 is using 8 boxes and Figure 4.12 is using 12 boxes, compare our algorithm to using local search

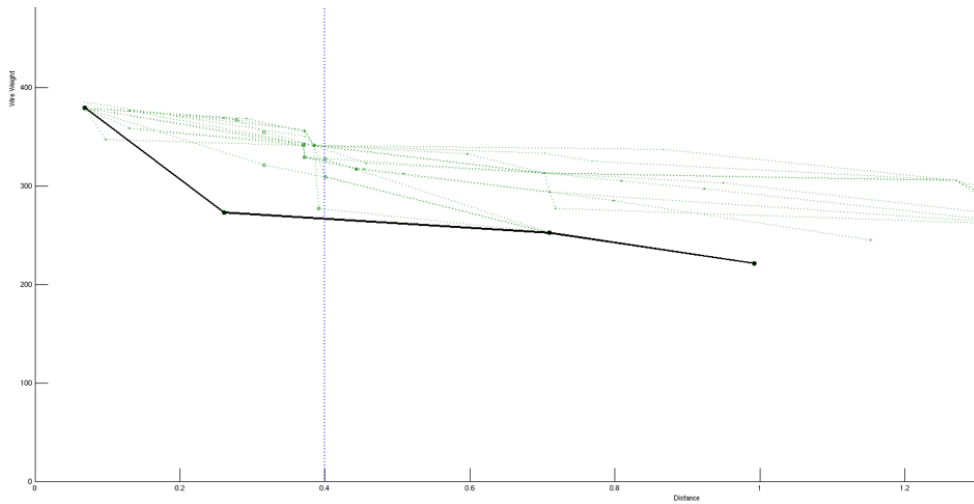


Figure 4.8: Scatter Search Results Test Case I

with random results. In these figures, the circles represent the scores of all the solutions generated in the entire run of the scatter search method. The x's represent the scores of an equal number of solutions generated by running the local search algorithm from a random starting point. The dark line is the Pareto frontier of the final Scatter Search population and the light line is the Pareto Frontier of the solutions generated by the local search algorithm. [Note: The Scatter Search frontier is significantly better than the local search frontier, indicating that Scatter Search performs better on this small example.] When using less than 7 boxes then Scatter Search does not have a clear advantage over Local Search.

Figure 4.13 gives a box placement for Test Case I. We created a program which translates results into this figure. It is very useful to have this program so that we can all of the data visually.

4.5 Conclusion

We believe that the Scatter Search and Graph Combining algorithm is well suited to the avionics electrical box placement problem. We have demon-

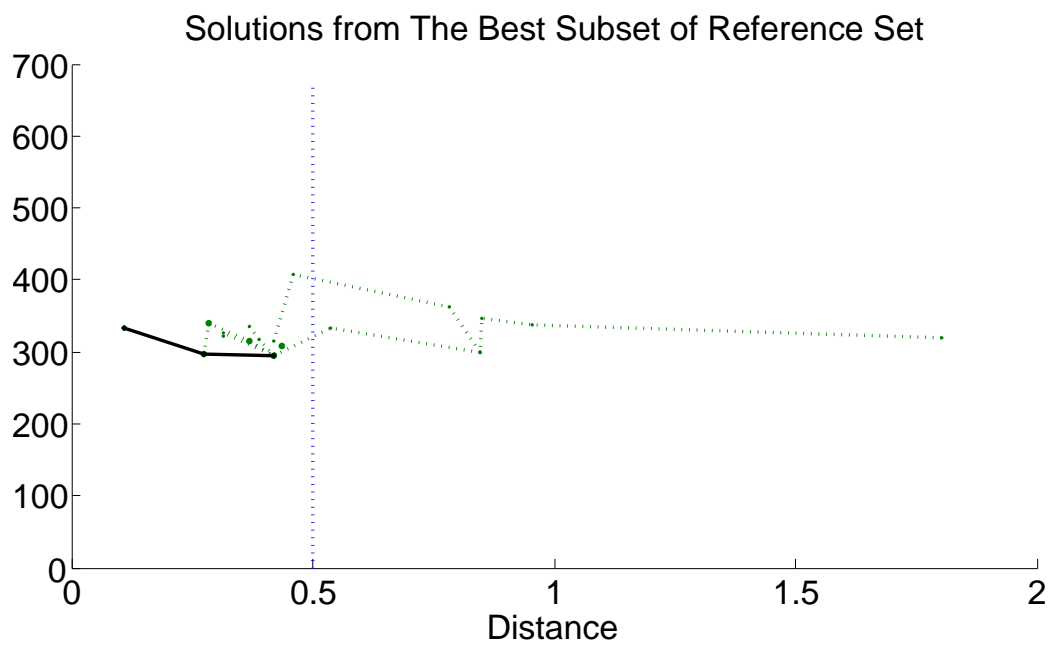


Figure 4.9: Scatter Search Results Test Case II

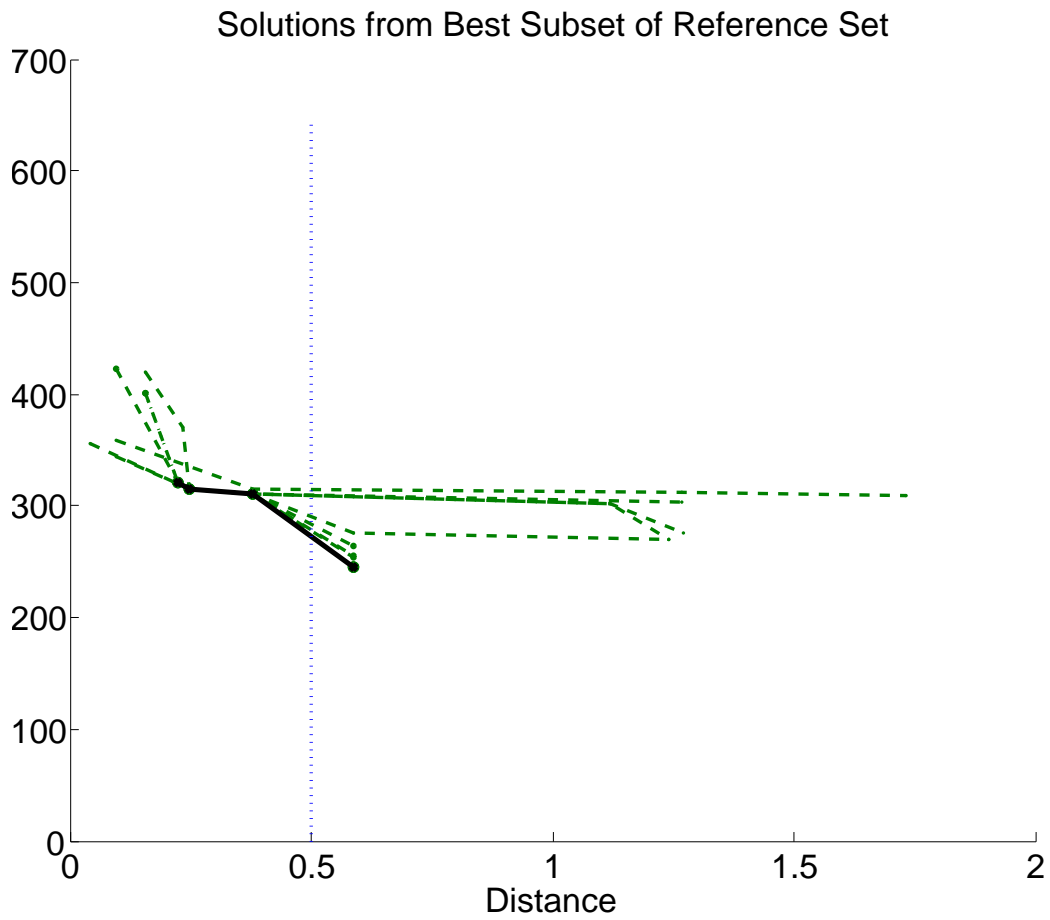


Figure 4.10: Scatter Search Results Test Case III

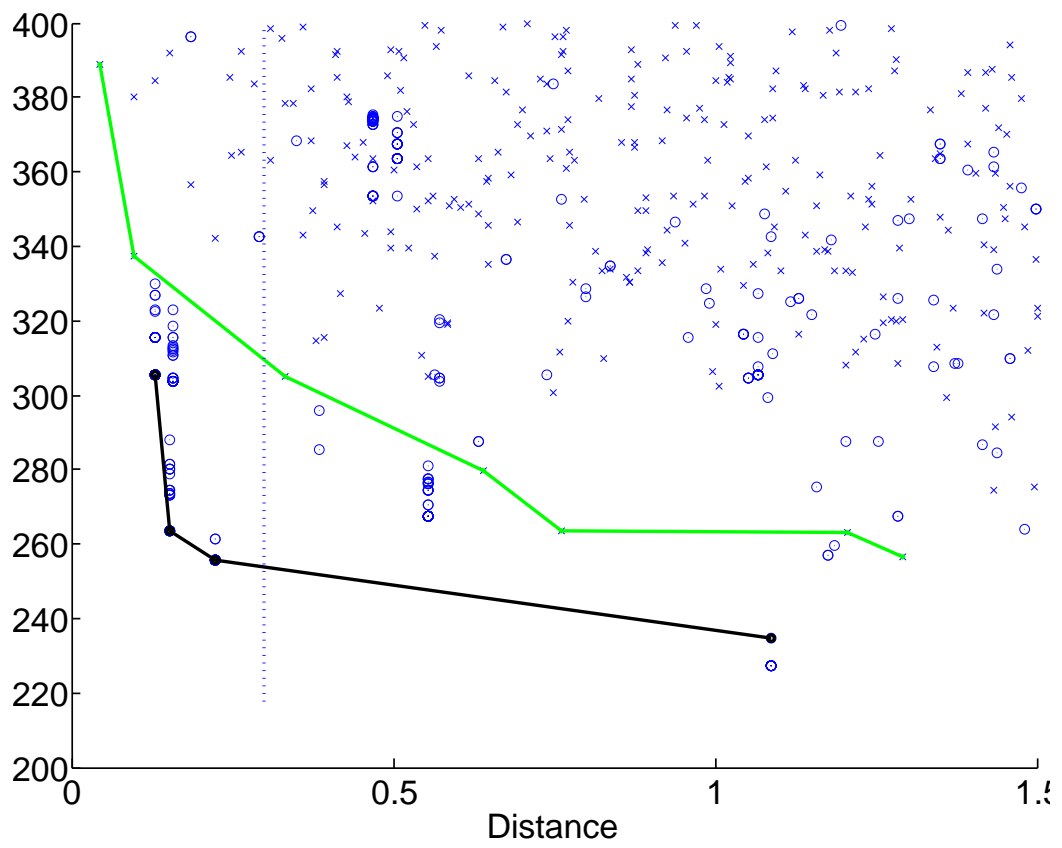


Figure 4.11: Scatter Search vs. Local Search

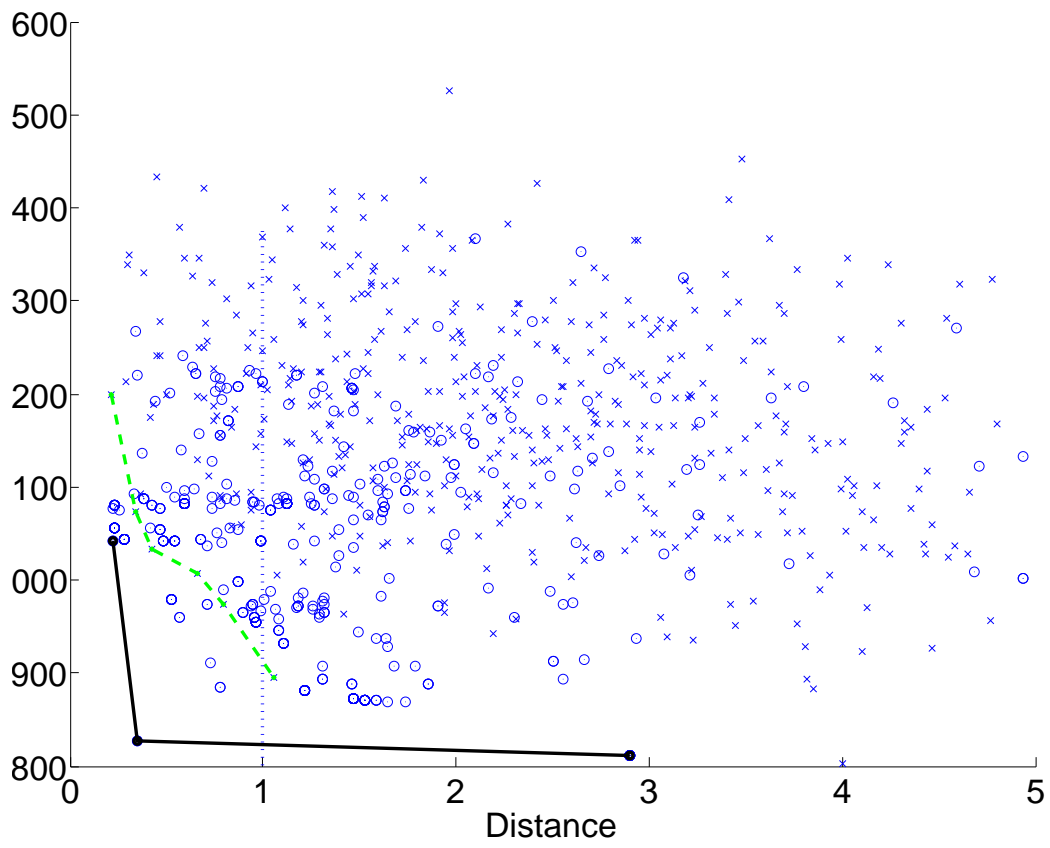


Figure 4.12: Scatter Search vs. Local Search

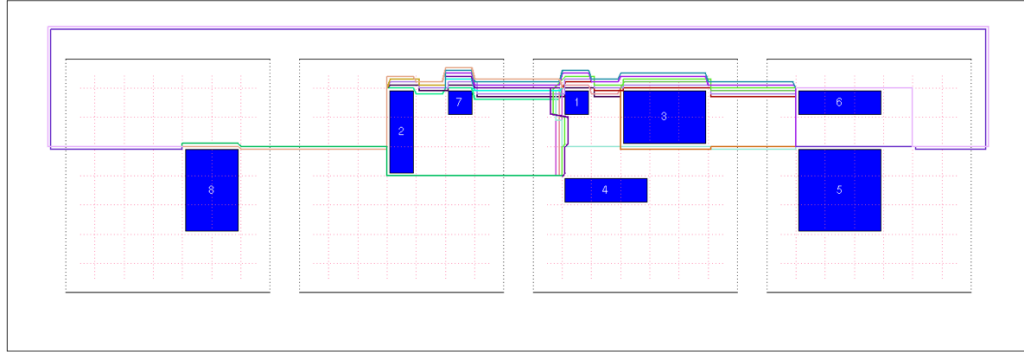


Figure 4.13: Box Placement

Table 4.1: Data Set I

Population	Reference Set	Best Set	Iterations	Wire Weight	Time(sec)
20	6	3	15	1000	2679
20	6	3	25	944.5	3482
30	8	4	15	827.5	5351
50	10	5	15	815.5	6292

Table 4.2: Data Set II

Population	Reference Set	Best Set	Iterations	Wire Weight	Time (sec)
10	4	2	10	346	108
10	4	2	20	307	187
10	4	2	30	294	201
20	4	2	10	324	228
20	4	2	20	310	488
20	4	2	30	263	554
30	6	3	10	310	420
30	6	3	20	284	745
30	6	3	30	260.5	1025
30	8	4	50	251	1793

strated that the algorithm does better than local search with random starts on several data sets. Although with a collection of fewer than seven boxes, Scatter Search and Graph Combining shows no advantage over local search and actually did worse. From our results it can be seen that our reference set improves at each iteration. These encouraging results not only attest to the appropriateness of Scatter Search for this problem, but also imply that the Graph Combining aspect of the algorithm succeeds at ensuring that the "good" qualities of solutions are passed on to successive solutions. It is also clear from our results that our solutions make sense, that is, the output from the algorithm easily translates into an actual box placement. Overall Scatter Search and Graph Combining has shown great promise as a method of solving the avionics electrical box placement problem.

4.6 Future Work

Possible future work may include a variety of approaches and adjustments. First and foremost, more testing of the algorithm must be done, including testing on larger data sets (more boxes and more connections). The Graph Combining aspect of the algorithm only combines two solutions to create a new solution, future work may be to have the algorithm "mate" more than two solutions (3-5) to produce a new solution. More fine tuning of the evolution of the penalty parameter value used in Team 1's local search may also produce better results. Also, the algorithm currently makes use of a 'greedy' phase 2 box placement algorithm and it may be beneficial to improve upon this process. Additionally it may be interesting to implement a TABU or Simulated Annealing based algorithm and compare results across the board.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The algorithms produced in this Mathematics Clinic provide a promising first step toward solving the Electrical Box Placement and Wiring Problem. These algorithms were implemented specifically to tackle a baseline model involving a 4 panel vehicle and relatively simple constraints; however, they can easily be adapted to other vehicle topologies with more complex constraints. Indeed, throughout the project, we kept potential variations of the problem in mind, with an eye toward generalizing our methods. In particular, we restricted our investigation to black-box methods, which allow for extremely complex function evaluations (perhaps involving simulations). Additionally, we used a penalized objective function for the local search method, which can easily be extended to handle arbitrarily complex constraints.

Preliminary testing of the algorithms is encouraging. The local search algorithm has been studied in detail on a small number of test cases. These tests indicate that the algorithm is working as it should, improving upon the starting point, and producing locally optimal solutions in all cases. Additionally, this algorithm has been exercised extensively in the testing done by both Teams 2 and 3.

Test results reported by Team 2 indicate that the starting points produced by their heuristic result in significantly better solutions than those produced by randomly generated starting points.

The most convincing results are reported by Team 3, which developed the Scatter Search and Graph Combining Algorithm. Their test results demon-

strate that their method is significantly better than running local search from a large number of randomly generated starting points. They recognized that comparing their solution to a *single* randomly generated starting point is hardly a fair comparison. This is because the scatter search method calls the local search algorithm many times, from many different starting points. To provide a fair comparison, they counted the number of times their algorithm called the local search method, and then called the local search algorithm using the same number of randomly generated starting points. The solutions generated by scatter search are dramatically superior to the best solution produced by using randomly generated starting points.

One issue that merits further discussion deals with how to balance the conflicting goals of minimizing wire mass and satisfying the center of mass constraint. Recall that the local search algorithm minimizes a *penalized* objective function, and that the user of the algorithm must specify the penalty parameter. An important question is how to choose the penalty parameter. If the parameter is too small, then the resulting solution may be far from any feasible solution (i.e., the center of mass constraint will be violated by a large amount, and there may not be any nearby solutions that satisfy the center of mass constraint). If the parameter is too large, then the algorithm may place too much emphasis on the center of mass constraint, making it very difficult to reduce the wire mass.

Team 3 implemented a simple and effective strategy for managing the penalty parameter. Namely, their algorithm initially uses a small value of the penalty parameter and progressively increases the penalty at each iteration. This strategy ensures that eventually, feasible solutions will be generated, while at the same time allowing flexibility in the early iterations to search for improvements in wire mass. By itself, this approach would not necessarily be successful. However, when used as part of a scatter search algorithm (which maintains a population of solutions), and when the selection of “best” solutions is based on selecting Pareto optimal solutions, this proves to be a very effective approach.

5.2 Avenues for Further Study

5.2.1 Local Search

There are several issues that could benefit from further study:

Neighborhood Structures. The particular neighborhood structure used in the local search algorithm has one significant disadvantage: there is an extremely large number of local minima. This is because only one box can be moved at a time, and those moves are limited to horizontal or vertical moves. This is illustrated by the following two observations:

1. If a given solution is balanced (i.e., the center of mass is near the central axis), then any horizontal move of a single box will move the center of mass away from the central axis. Thus, given a balanced configuration, *any* move will likely result in an unbalanced solution. In fact, this is one reason the local search algorithm optimizes a penalized objective function rather than searching only over the feasible solutions. But even at that, if the center of mass penalty is large, then essentially every balanced solution will be a local optimum.
2. If all the boxes are placed in a single row, then any vertical move of a box will increase the wire mass while not affecting the distance between the center of mass and the central axis. Thus, every solution that places all the boxes on a single row will be locally optimal. This observation is the reason why Team 2's algorithm explicitly places the boxes on multiple rows before calling the local search algorithm.

To overcome this limitation, it is worth considering richer neighborhood structures in which more types of moves are allowed. Team 1 has proposed several ideas for expanded neighborhood structures that could result in significant improvements. One idea is to allow diagonal moves of boxes. This simple change would make it possible to move boxes off of a single-row solution, since a diagonal move could be made that keeps the wire mass the same. However, this would result in doubling the number of neighbors each solution has.

Another possible change would be to allow two boxes to be moved simultaneously. In this way, moves could be made without significantly altering the center of mass. However, this strategy would dramatically increase the number of neighbors each solution has. Therefore, some care would be needed to ensure the neighborhood space could still be searched efficiently. As an example, we might limit the allowable two-box moves to be exchanges between adjacent boxes.

Other possible changes include allowing boxes to move from one panel to another, and allowing boxes to be rotated.

5.2.2 Heuristics

Additional Testing. More testing needs to be done to evaluate how effective the heuristic method is. The preliminary test results demonstrate that the method produces better results than a single run of local search with a random starting point. However, one could argue that randomly generated starting points are not likely to result in good solutions; so beating a random start is not too hard to achieve. A more informative test would be to compare the results generated from the heuristic starting points to the best results generated by a large number of random starting points. Another reasonable test would be to compare the results of the heuristic method to the results generated by the scatter search method.

Methods for generating circular lists. The idea of constructing a circular list of boxes is intuitively appealing and potentially quite powerful. Team 2 developed one particular strategy for creating this circular list, but it is worth exploring other methods for generating the circular lists:

1. **Min-cut.** A min-cut method would partition the boxes into several subsets in order to minimize the total mass of wires that go from one subset to a different subset. After the subsets are formed, the boxes in each subset could then be ordered into a list, and the lists could be concatenated to form the final circular list. This method is interesting not only because it could potentially do a better job of grouping highly interconnected subsets of boxes, but also because the Min-cut problem is a very well-studied problem, so it would be possible to take advantage of existing algorithms for solving this problem.
2. **Clustering.** Clustering methods, like the min-cut method described above, partition items into subsets, where the items in each subset are more similar to each other than they are to items in other subsets. For our purposes, we could define a similarity measure for boxes based on the interconnection matrix. There is a very large literature on clustering methods, so it would be very easy to explore a variety of methods.
3. **Exchanges.** After the circular list has been generated by one of the methods described above, it might be worth implementing a local search strategy over the space of circular lists. For example, we might create a new starting solution by exchanging two adjacent boxes in the

circular list and run the local search algorithm based on the new list. If the resulting solution is worse than that generated by the first list, it would be rejected, and we would then try a different exchange. If the new solution is better than the original solution, we would accept the change, and then try exchanging boxes in the new list. This process would terminate when a circular list is found that cannot be improved by any such exchange.

5.2.3 Scatter Search and Graph Combining

Some of the key components of the scatter search method are dependent on the particular topology of the vehicle in our baseline model. In particular, the construction of the distance matrices assumes that there are four panels on the vehicle being designed. However, it would be reasonably straightforward to generalize the method to larger numbers of panels.

That said, there are two possible modifications that should be explored, which might prove better-suited for vehicles with large numbers of panels, or with cylindrical topologies:

1. For purposes of maintaining diversity within the reference set, it may be better to define the distance between two solutions not just by counting the number of entries in the distance matrices that are different, but by summing up the absolute values of the differences between the entries.
2. For purposes of combining two solutions, there will likely be far fewer entries in the distance matrices that are identical. So the similarity graph described in Section 4.3.2 would have far fewer arcs. It may be worth exploring modifications to these graph in which arcs would be drawn between two nodes if the corresponding entries in the distances matrices are sufficiently close.

Glossary

NP-hard: The problem class P consists of all problems for which it is possible to find a algorithm that will solve any instance of the problem in polynomial time. The problem class NP consists of all problems where it is possible find a polynomial time algorithm that will verify whether or not a solution is correct. It is widely believed that $P \neq NP$. That is, it is believed that there exist problems in NP that are not in P . However, this has never been proven.

A problem is said to be NP-hard if it cannot be solved in polynomial time unless $P = NP$.

Neighborhood Structure: A neighborhood structure is a relationship between items in a search space that specifies which items are neighbors.

Metaheuristic: A metaheuristic is a high-level algorithmic strategy that guides other heuristics (such as local search) in searching for a globally optimal solution.

Polynomial Time: An algorithm is said to run in polynomial time if its worst-case running time is bounded above by a polynomial function of the problem size.

Bibliography

- [1] E. W. Dijkstra. A note on two problems in connection with graphs. *Numer. Math.*, 1:269–271, 1959.
- [2] Fred Glover et al. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000.
- [3] Fred Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.
- [4] B. Jackson and J. Norgard. A stochastic optimization tool for determining spacecraft avionics box placement. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 5, pages 5–2373–5–2382 vol.5, 2002.
- [5] Darrell Whitley. A genetic algorithm tutorial.