Constructing minimum-cost solder paths

van Hoeij, P.W.P.

Award date:
2009

TU/e Technische Universiteit Eindhoven University of Technology

PRODRIVE

Constructing Minimum-Cost Solder Paths

Paul van Hoeij

25 August 2009

In Cooperation with Prodrive B.V.

Master's Thesis

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science

Supervisor: dr. Alexander Wolff
Referees: dr. Mark de Berg
dr. ir. Cor Hurkens
Company contact: Roy Reuser

Abstract

In this thesis, we investigated the problem of soldering components on printed circuit boards. Currently, soldering machines are configured per product by hand. The goal is to automate this process to save time and reduce mistakes. We divide the problem into two sub problems; finding solder actions and finding solder paths.

Multiple pins can be soldered by one solder action. Finding these solder actions can be seen as covering a set of points in the plane with hippodromes, that is, cigar-shaped objects. We give a heuristic approach to obtain a possible hippodrome cover. Unfortunately, the covers found do not result in high-quality solder joints and are therefore not practical.

A solder path is a sequence of solder actions performed by the machine. We search for solder paths that solder all pins and that maximize the throughput of boards through the machine. This problem can be seen as a traveling salesman problem with extra side constrains. We used an integer-programming approach to solve this problem to optimality given a set of solder actions per solder module.

Contents

Chapter 1

Introduction

In the manufacturing of electronic devices many steps are automated; from placing the components on the printed circuit board to functional testing of the device. Unfortunately, before the production machine can operate automatically, human interaction is needed. Operators must first check and adapt the input for these machines in order to let them operate smoothly. Automating these processes without loss of quality is the ultimate aim of production companies. Automation does not only save time but also ensures a constant quality level of the processes.

A well-studied example is the problem of efficiently drilling holes in a printed circuit board [Rei94]. These holes are needed for placing larger components, for example transistors and conductors, onto a board and connecting them with copper layers of the board. The input of the drilling machine is a sequence of positions that must be drilled in that order. To make drilling efficient the aim is to find a sequence that minimizes the total processing time. Humans can provide an input that is close to optimal [MO96] but there are also exact algorithms to get an optimal solution for the drilling problem such as the algorithms used in the state-of-the-art TSP solver Concorde [Coo05].

In this thesis we are interested in generating the input for another machine in the manufacturing process of electronic devices, the *selected wave soldering* (SWS) machine. This machine solders components that are placed through the drilled holes in the printed circuit board. Compared to other machines in the production, this SWS machine is relatively slow and needs a lot of human interaction to work well. A detailed description of the machine is given below. Our goal is to automate the setup process of the SWS machine as far as possible without loss of *quality*.

1.1 Selected Wave Soldering Machine

The selected wave soldering machine solders the so-called *through-hole components* of a product. The though-hole components are mounted on the board by pins. These pins are inserted in drilled holes of the board. The SWS machine solders the pins on the side of the board opposite to where the component is placed. The SWS machine consists of three modules; a

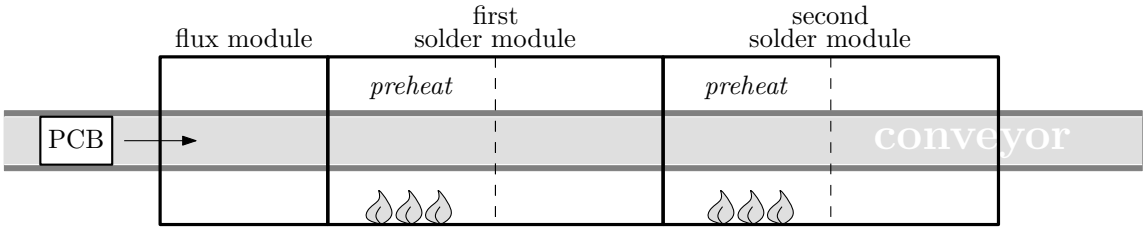


Figure 1.1: A Selected Wave Soldering (SWS) machine with three units; a flux unit and two soldering units.

flux module and two soldering modules (Figure 1.1).

- The *flux module* makes sure that every pin that is soldered is sprayed with a flux mixture. The flux mixture is a chemical substance that increases the quality of soldering. It removes and prevents oxidation on the surface and improves the wetting characteristics of tin.

  The spray nozzle can move in x- and y-direction. The spraying can be done with different velocities and intensities. The amount of flux on a particular location on the board is a function of the velocity and intensity.

- The *soldering modules* first preheat the board until a certain temperature is reached. This is important step in order to get good soldering joints. After preheating, the components are soldered. The solder units use a soldering nozzle that creates a tin fountain. This fountain is located below the board. It solders everything it comes in contact with, which makes it possible to solder multiple pins simultaneously. To get high-quality soldering joints, the pins must be within a certain distance with respect to the center of the nozzle.

  The soldering nozzle can move in x-, y- and z-direction. The z-direction is used to move the soldering nozzle toward and away from the board. By moving the nozzle toward the board the fountain touches the board at a certain distance. This distance is a function of the shape of the fountain and the z-position of the nozzle.

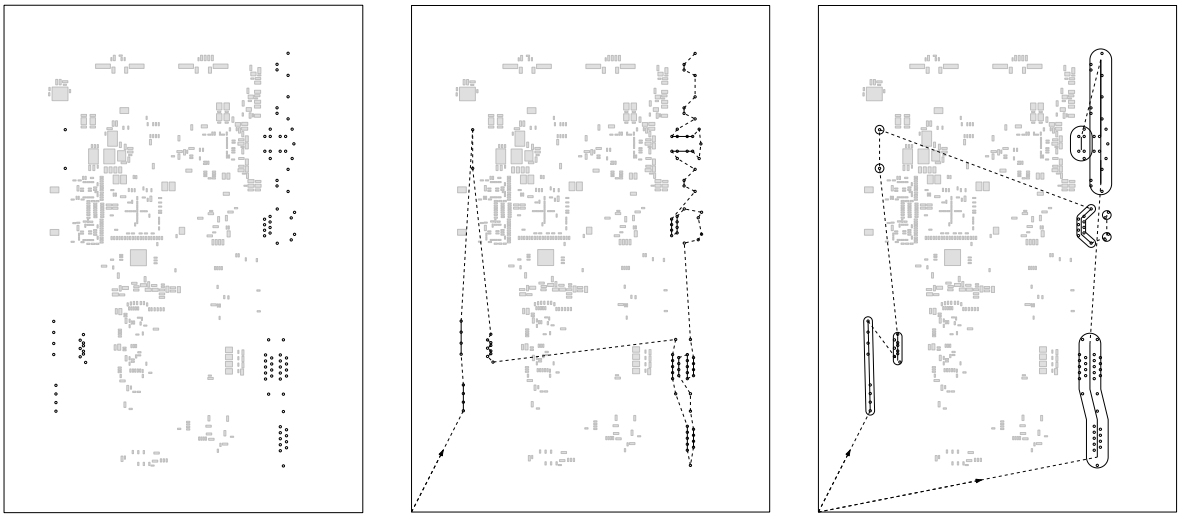
  To prevent the board from bending during the soldering a support bar can be added. This bar is placed on the solder side thus the solder nozzle must avoid the support-bar area.

  Both solder modules operate simultaneously and can have different nozzles. The order of visiting the modules is predefined by the conveyor that moves the board through the machine.

1.2 Problem Statement

Given the description of the machine we have to construct three paths to solder all components, namely a flux path and two solder paths. In order to construct these paths we need information about the board that is soldered. This information includes the following:

- The set of *pins* that need to be soldered. Every pin has an unique point location on the board. The minimal distance between two pins is assumed to be 2.5 mm.
- A set of axis-parallel rectangular *keep-out areas*. These areas correspond to components on the solder side, the support bar and other areas where the machine may not solder.
- The set of soldering *nozzle* (sizes) that can be used by the machine. The region that touches the board is assumed to be disk shaped. We call this region the solder area. In practice the shape of the region depends on the movement of the nozzle.
- The *start position* of the nozzle in the modules. In this thesis we use as start position located on the lower left corner of the board.



(a) (b) (c)

Figure 1.2: Example of product 6001-0710-1001 (a) with the three paths; flux path (b) and two solder paths (c).

Figure 1.2 shows an example of a typical printed circuit board. The pins are represented by small circles and keep-out areas by gray rectangles. With the information shown in Figure 1.2a the operator defines three paths; a flux path (Figure 1.2b) and two soldering paths (Figure 1.2c).

Flux Path

The *flux path* is used to spay a *flux mixture* on every pin that needs to be soldered. The purpose of this flux mixture is to increase the quality of soldering. It is important that the pin and the inside of the hole is totally covered with the flux mixture. In many cases it is enough to spray exactly on the center of the pins. There are, however, cases when this is not sufficient, for example, in the case of rectangular shaped pins, every side of the pin must be sprayed. We do not consider these cases because we are not provided with this information. The velocity of the flux nozzle is about 20 mm/s while fluxing and 400 mm/s

without fluxing. All velocities mentioned are according the specifications of the SWS machine used by Prodrive.

In Figure 1.2b the flux path is shown for a board (Figure 1.2a). This path starts at the begin position which is here located at the lower left corner of the board. The dashed lines represent the nozzle movements. At every pin position on these dashed lines some flux mixture is sprayed. The solid lines represent paths that are totally sprayed. It is common that the start and end position of the lines are located at the center of a pin.

Solder Paths

A *solder path* is a path that solders pins. Such a path consists of three types of actions; movement actions, track-solder actions and point-solder actions.

- A *movement action* moves the nozzle from one location to another location without the fountain touching the board. The velocity of a movement action is about 200 mm/s for movement parallel to the x-y plane and about 100 mm/s for movement in z-direction. These velocities must be adaptable in the algorithm to ensure the solder quality and the lifespan of the SWS machine into account.
- A *track-solder action* moves the solder nozzle while the board is touched by the fountain. Since the fountain touches the board every pin that lies on the path is soldered. In order to start with soldering the nozzle has to move toward the board. Stopping with soldering is done by moving the nozzle away from the board. The velocity while soldering is about 10 mm/s.
- The last type of action is the *point-solder action*. This action solders a disk-shaped region with the same diameter as the nozzle. It moves the nozzle toward the board and waits for a particular time. After waiting, the nozzle moves away from the board. This action is almost the same as the track-solder action except the nozzle waits instead of moving to another location.

The example in Figure 1.2c shows two solder paths starting at the lower left corner of the board. One path uses a small nozzle and the other a larger nozzle. The smaller nozzle moves from the start position to its first track-solder action. This movement is represented by a dashed line. The hippodrome areas (cigar shapes) are track-solder actions. The hippodrome shape is caused when the nozzle touches the board. The first track-solder action by the smaller nozzle solders eight pins. It is possible that a track-solder action has bends, like the first action of the path with the larger nozzle. The firth and fourth solder action performed by the smaller nozzle are point-solder actions, see the upper left corner of the board. Only one pin is soldered per action.

To obtain a solution that maximizes the throughput of boards through the machine we have to balance the flux and solder paths such that the processing time of the path with the largest processing time is minimized. This is like the bottleneck principle. The module that requires the longest processing time to complete its task is the bottleneck. The modules in front of this module are blocked since they cannot deliver the board and the modules after the

bottleneck must wait for a board to be finished by the bottleneck module. So, minimizing the processing time of the module that is the bottleneck increases the throughput of boards through the machine. For balancing we only take the soldering paths into account because the flux process is much faster (twice as fast) and does not change the z-coordinate in order to flux the pins. In the example of Figure 1.2c, the operator tried to balance the paths by soldering the middle right pins with the path with the smaller nozzle.

The operator carefully chose paths that are good with respect to both solder quality and processing time, but is the operator's solution optimal? This question is difficult to answer by looking at the visual representation of the board. There are two main reasons. First, the problem does not only use Euclidean distances but a function of the movement velocities and distances. In some studies researchers have shown that humans solve traveling salesman problem instances, similar to our problem instances, near optimally as long as points lie in the Euclidean plane [SP08] which is not the case in our problem due to the different velocities. Second, the paths must be balanced in order to obtain the fastest overall processing time. This makes it even harder for humans to find an optimal solution.

To simplify the model we use some assumptions. Most of them have already been mentioned in this section. The complete list of assumption is given here:

- Keep-out areas are axis-parallel rectangles.
- The solder area of the fountain is a disk-shaped region.
- Only *one* nozzle is given per solder module.
- Flux is sprayed on the center of a pin.
- Keep-out areas are only avoided by the soldering modules.
- Only the solder paths are balanced.

With this detailed description we can define the problem more formally.

Problem (Minimum-Cost Flux and Solder Paths)**.** *Given a set $\mathcal{S}$ with points in the plane and a set $\mathcal{K}$ with axis-parallel rectangular keep-out areas, partition the set $\mathcal{S}$ into two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$. Given a set R of available nozzle sizes construct three paths p_0, p_1 and p_2 such that p_0 sprays flux on all points in $\mathcal{S}_1 \cup \mathcal{S}_2$, p_1 solders with a nozzle $r_1 \in R$ all points in $\mathcal{S}_1$ and p_2 solders with nozzle $r_2 \in R$ all points in $\mathcal{S}_2$. Both paths p_1 and p_2 must avoid all keep-out areas in $\mathcal{K}$. Let $|p|$ be the processing time of path p. The goal is to minimize the processing time of the path with the largest processing time*

$$\textit{minimizing } \max_u \{|p_u|\}.$$

1.3 Contributions

In this thesis we are only interested in solving the Minimum-Cost Flux and Solder Paths without looking at the flux path, since the flux unit is very fast in comparison to the solder units. In Chapter 6 under future research, we present a possible solution to obtain a flux path.

We divide the problem into two subproblems; obtaining solder action for both solder modules (Chapter 3) and constructing two solder paths that minimize the processing time (Chapter 4).

The process of soldering component with a tin fountain is a difficult process. It involves many variables and therefore it is difficult to predict the solder result of an solder action. Despite the difficulties, we show a way of obtaining solder actions that fulfill a minimal requirement for soldering pins. In Section 3.4, we show an heuristic algorithm that can be seen as a first approach toward finding good soldering action. The algorithm covers the set of pins with nozzle-size disks. The disks represent point-solder actions and connecting two or more disks by a path we obtain track-solder actions. Unfortunately, the solutions of this heuristic has properties that are undesirable in practice (Section 3.5).

After obtaining the solder actions per solder module we find an optimal solder path for all solder modules. These paths solder all pins and minimize the total process time of the machine. We used a well-known mathematical model of traveling salesman problem (Section 2.3) to derive the properties needed for solder paths. We describe two models; the single solder unit model (Section 4.4) and the multiple solder unit model (Section 4.5).

In the final part of this thesis we show some experiments on real products (Chapter 5) and we discuss related problems that need more research in Chapter 6.

Chapter 2

Preliminaries

In this chapter we introduce some notions we use throughout the thesis. We start with linear programming (Section 2.1) and integer programming (Section 2.2). These techniques are used to solve the problem of finding solder paths. We constructed our mathematical model by looking at a similar problem, the traveling salesman problem. We give a short introduction into the traveling salesman problem and discuss various formulations (Section 2.3).

2.1 Linear Programming

Linear programming (LP) is a technique to find an optimal solution given a mathematical model. In this mathematical model only linear equations are allowed. A model of this kind has the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{such that} & Ax \leqslant b, \end{array}$$

where $c^T x$ is the objective function that we want to minimize. The function uses the vector x that are the variables (or unknowns) and c as coefficient vector. The coefficient matrix A and the vector b encode the constraints that must hold for x. The conjunction of these constraints defines a convex polytope that represents the feasible solution space.

Figure 2.1a gives an example of an LP instance with five constraints and two variables. These constraints define the (gray) convex polytope representing the solution space. The objective function is shown by the dashed line and the arrow. The arrow indicates the direction that minimizes the objective function. Moving the dashed line in the direction of the arrow yields the optimal solution at the boundary of the solution space, indicated by the square.

There are a number of ways of solving these type of mathematical models: the simplex method [Dan51], the ellipsoid method [Kha79] and the interior point method [Kar84]. The simplex method, in short, solves an LP by starting at an arbitrary vertex on the boundary of the convex polytope. It then walks to an adjacent vertex where the value of the objective function is better. This continues, avoiding cycles, until the vertex of the optimal solution is found. In Figure 2.2 the steps of the simplex algorithm are shown for an example. The simplex method performs well in practice. It has a polynomial smoothness complexity in the

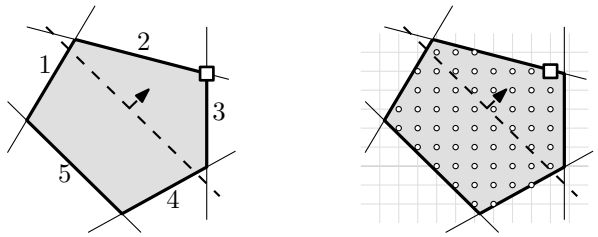


Figure 2.1: In (a) an LP instance is shown with five constraints that define the gray convex polytope representing is the solution space. The dashed line represents the objective function. The optimal solution is indicated by the square. The IP instance (b) uses the same constraint and objective function. Only the integral solutions are feasible. The optimal solution is indicated by the square.

number of variables [ST04]. The other algorithms run in polynomial time; therefore LP has polynomial complexity.

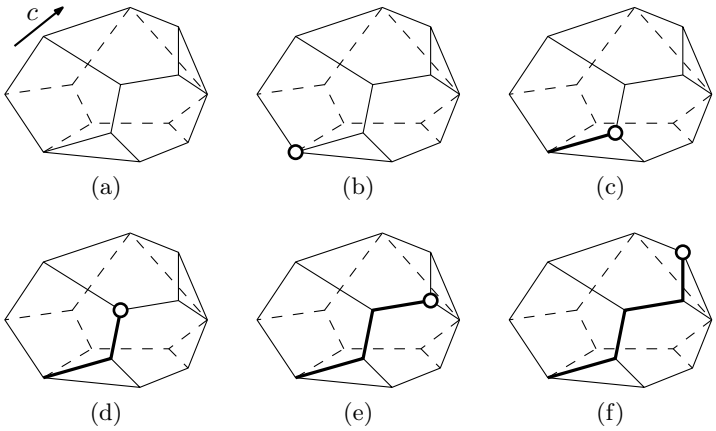


Figure 2.2: Example of a walk around the boundary by the simplex algorithm. The convex polytope of the linear relaxation of the IP instance with the objective function is shown in (a) where c is the objective function. The algorithm starts at an arbitrary vertex (b). In the next steps adjacent vertices of the current vertex are chosen with the best costs (c)-(f). The vertex with no better adjacent vertex is the optimal solution (f).

2.2 Integer Programming

In contrast to LP, integer programming (IP) uses only integer variables. With this restriction the problem becomes harder to solve. The $\mathcal{NP}$-hardness of IP is obvious since IP generalizes, say, the $\mathcal{NP}$-hard problem vertex cover. In 0-1 IP, a solution can be guessed (with non-zero probability), so 0-1 IP is $\mathcal{NP}$-complete.

In LP, the simplex algorithm finds the optimal solution by walking over the boundary to the

vertex that has no better adjacent vertices. In IP, the same trick cannot be applied because not all values on the boundary are feasible solutions due to the integrality restriction, see Figure 2.1b where the dots are integral solutions.

There are a number of techniques to solve an IP problem. We briefly sketch some techniques for solving an IP; the branch-and-bound method, cutting-plane algorithms and the branch-and-cut method. For more details, see the book *Integer Programming* by Wolsey [Wol98].

2.2.1 Branch-and-Bound

Branch-and-bound is a divide-and-conquer method for solving IP models. The method creates a tree with nodes corresponding to subproblems of the original problem. In the branching step, the algorithm splits the IP problem into multiple smaller IP problems. In the bounding step, the algorithm decides whether to prune or to continue traversing a branch of the tree.

For large IP problems the branch-and-bound algorithm is unpractical. A more common approach to tackle larger problems is to tighten the solution space by strengthening the constraints or by adding (stronger) cutting planes.

2.2.2 Cutting-Plane Algorithms

A cutting-plane algorithm iteratively adds multiple cutting planes to the model to tighten the solution space.

Definition 1 (Cutting plane)**.** *A cutting plane is an inequality that cuts off part of the feasible solutions space and preserves the optimal solution.*

The first cutting-plane algorithm was presented by Dantzig et al. [DFJ54]. They used this technique to solve a 49-city TSP by using sub-tour elimination constraints, see Sections 2.3.1–2.3.3, and comb inequalities as cutting planes. The big challenge of a cutting-plane algorithm is to find a cutting plane that cuts off the non-integral solution obtained from the LP relaxation. Finding such cutting planes is known as the Separation Problem.

Problem (Separation Problem)**.** *Find a cutting plane that cuts off a non-integral optimal solution obtained by a linear relaxation of an integer linear program. The cutting plane separates the non-integral optimum from the feasible solution space.*

Solving the separation problem is sometimes as hard as the original problem itself. Some problems such as the traveling salesman problem are heavily investigated by researchers to find a polynomial separation algorithm, sometimes with success. In his doctoral thesis Hong [Hon72] presented an exact polynomial separation algorithm to find sub-tour cuts for TSP by using min-flow min-cut computations.

2.2.3 Branch-and-Cut

More advanced methods are created in order to solve faster and large instances. One of these methods is the branch-and-cut method. These algorithms combine the branch-and-bound

structure with cutting planes. In the branching step, not only the problem is split but also the solution space is tighten by using cutting planes. This results in a tree with fewer nodes and branches. Since the tree is smaller, the optimal solution is found faster.

2.3 Formulations for the Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is one of the most investigated problems in computer science. The problem is defined as follows:

Problem (Traveling Salesman Problem)**.** *Given a set of n cities and the distances between each pair of cities, find a tour of minimal total length visiting each city exactly once.*

This problem arises in logistics, robotics, manufacturing and many other areas. Some TSP-like problems are school bus routing [SSS06], newspaper delivery [ACDR02] and inspection problems. With the introduction of robotics in manufacturing of electronic devices new TSP-like problems arise such as the drilling problem. The problem is to drill holes in a printed circuit board and to minimize the processing time. The drilling problem is similar to the TSP because the cities can be seen as the holes that need to be drilled and the distances between the cities can be seen as the time needed for moving the drill head.

In this section we give the three most well-known IP formulations of the (asymmetric) TSP, namely sub-tour formulation (Section 2.3.1), sequential formulation (Section 2.3.2) and flow formulation (Section 2.3.3). Before describing these formulations in detail, we give some preliminaries.

The formulations are based on a complete directed graph $G = (V, E)$, where V is the set of vertices corresponding to the cities and $E \subseteq \{(u, v) \mid u, v \in V, u \neq v\}$ is the set of edges representing the roads between the cities. In the given IP formulations zero-one variables are used to represent the edges in E. We use x_{ij} to denote the variable that corresponds to the edge (i, j) that starts at vertex i and ends at vertex j.

$$x_{ij} = \begin{cases} 1 & \text{if edge } (i,j) \text{ is in the tour} \\ 0 & \text{otherwise} \end{cases}$$

The objective function of these formulations is to minimize the total cost of all edges in a tour. The cost of edge (i, j) is given by c_{ij}. The objective function is thus to

$$\text{minimize} \sum_{\substack{i,j \in V \\ i \neq j}} c_{ij} x_{ij}. \tag{2.1}$$

All formulations consist of two types of constraints; assignment constraints and sub-tour elimination constraints. The *assignment constraints* are used to assign two edges per vertex, one incoming and one outgoing. This is based on the definition of a Hamiltonian cycle in graph theory.

Definition 2 (Hamiltonian Cycle)**.** *A Hamiltonian cycle is a cycle that traverses every vertex exactly once.*

For short we use the term *tour* to mean a Hamiltonian cycle. From the definition of the tour it is trivial to see that if we have a tour then every vertex has a degree of two due to the fact that the tour must go through each vertex exactly once.

$$\sum_{j \in V \setminus \{i\}} x_{ji} = 1 \qquad \text{for } i \in V \tag{2.2}$$

$$\sum_{j \in V \setminus \{i\}} x_{ij} = 1 \qquad \text{for } i \in V \tag{2.3}$$

Unfortunately, if we use only assignment constraints, the solution may contain several so-called *sub tours*. By adding extra constraints, we can exclude these sub tours from the solution space. These extra constraints are called *sub-tour elimination constraints* (SECs). In the next sections we describe three types of SEC formulations.

2.3.1 Sub-tour Formulation

The sub-tour formulation, also known as *clique packing constraints*, is the simplest formulation. It was introduced by Dantzig et al. [DFJ54] and it follows a graph theoretical approach by restricting the number of edges in a set of vertices. The number of edges in a tour is equal to the number of vertices in the tour. So, for any non-empty proper subset S of V, the number of edges in the sub graph induced by S is at most $|S| - 1$.

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leqslant |S| - 1 \qquad \text{for } S \subset V \text{ and } S \neq \emptyset \tag{2.4}$$

Theorem 1. *Minimizing the function* (2.1) *subject to the assignment constraints* (2.2) *and* (2.3) *and the sub-tour elimination constraints* (2.4) *results in one tour visiting all vertices exactly once.*

Proof. "⇐" Let t be a tour visiting all vertices that satisfies constraints (2.2) and (2.3). Without loss of generality the order of visiting the vertices in t is $v_1, v_2, \ldots, v_n$. By taking any non-empty proper subset S of the vertices in t, the number of edges in the subgraph of t induced by S is always less than $|S|$. The only set which results in $|S|$ edges is the set V, but by the definition of proper subset this cannot be the case.

"⇒" Suppose there are $k \geqslant 2$ sub tours $t_1, t_2, \ldots, t_k$ that together visit all vertices and satisfy constraints (2.2) and (2.3). Take for S the proper subset of V that includes all vertices of t_1 and none of the other sub tours. The subgraph induced by S contains $|S|$ edges, which violates constraint (2.4). □

The constraint (2.4) can be rewriting in an equivalent form. Let S be a non-empty proper subset of V. To obtain one tour, all subsets S must have at least one edge to $V \setminus S$. The correctness of this constraint can be proven with the same principle as the previous constraint.

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geqslant 1 \qquad \text{for } S \subset V \text{ and } S \neq \emptyset \tag{2.5}$$

Both (2.4) and (2.5) use the definition of proper non-empty subsets. There are $2^n - 2$ non-empty proper subsets of V and therefore $O(2^n)$ constraints. There are no new variables introduced in these constraints, so the sub-tour formulation uses $O(n^2)$ zero-one variables.

2.3.2 Sequential Formulation

The sequential formulation described by Miller et al. [MTZ60] is very efficient with respect to the number of constraints and variables. It involves $n^2 - 3n + 2$ additional constraints and $n - 1$ additional variables. This formulation uses the order of the vertices in the tour to eliminate sub tours. Every vertex i gets a variable u_i that represents the position of that vertex in the tour. If the tour is $1, 2, \ldots, n$ then $u_1 = 1, u_2 = 2, \ldots, u_n = n$.

$$u_i - u_j + nx_{ij} \leqslant n - 1 \qquad \text{for } (i,j) \in V^2 \text{ with } i \notin \{1\} \text{ and } j \notin \{1, i\} \tag{2.6}$$

Theorem 2. *Minimizing the function* (2.1) *subject to the assignment constraints* (2.2) *and* (2.3) *and the sequential sub-tour elimination constraints* (2.6) *results in one tour visiting all vertices exactly once.*

Proof. "⇐" Suppose we have a tour that consists of vertices $v_1, v_2, \ldots, v_n$ and let π be a permutation that determines the order of the vertices in the tour. For every $i \in \{1, 2, \ldots, n-1\}$, we define that $v_{\pi(i+1)}$ is the successor of $v_{\pi(i)}$, and $v_{\pi(1)}$ is the successor of $v_{\pi(n)}$. Let u_i be $\pi^{-1}(i)$, that is, the i-th vertex in the tour. Take an $l \in \{1, 2, \ldots, n-1\}$ such that $\pi(l) = i$ and $\pi(l+1) = j$, then it follows that

$$u_i - u_j + nx_{ij} \leqslant n - 1.$$

This is equivalent to

$$u_{\pi(l)} - u_{\pi(l+1)} + nx_{ij} \leqslant n - 1.$$

Due to the definition of π^{-1} it follows that

$$\pi^{-1}(\pi(l)) - \pi^{-1}(\pi(l+1)) + nx_{ij} \leqslant n - 1.$$

This simplifies to

$$l - (l+1) + nx_{ij} \leqslant n - 1,$$

which in turn means that

$$nx_{ij} - 1 \leqslant n - 1.$$

So, by assigning the variables u_i and u_j with the the i-th and j-th vertex in the tour, respectively, the constraint (2.6) is satisfied independently of the value of x_{ij}.

"⇒" Without loss of generality, assume there is a sub tour $2, 3, \ldots, k, 2$ with $2 \leqslant k \leqslant n$. Then it follows that

$$\sum_{i=2}^{k-1} [u_i - u_{i+1}] + (u_k - u_2) \leqslant -(k+1).$$

This is rewritable to

$$(u_2 - u_k) + (u_k - u_2) \leqslant -(k+1),$$

which simplifies to

$$1 \leqslant -k.$$

This proves that constraint (2.6) is violated because $k \geqslant 2$. □

2.3.3 Flow Formulation

A new formulation was introduced by Gavish and Graves [GG78]. This formulation uses as basis the network flow problem to eliminate sub tours. In this formulation, for each edge (i, j) a new continuous variable y_{ij} is introduced that indicates the amount of flow through edge (i, j). The idea is that each vertex, except vertex 1, consumes one unit of flow. This formulation uses $n^2 - 2n + 1$ additional constraints and $n^2 - n$ additional variables.

$$\sum_{j \in V \setminus \{i\}} y_{ji} - \sum_{j \in V \setminus \{1,i\}} y_{ij} = 1 \qquad \text{for } i \in V \setminus \{1\} \tag{2.7}$$

Any edge that has flow is forced to be in the tour.

$$0 \leqslant y_{ij} \leqslant (n-1)x_{ij} \qquad \text{for } (i,j) \in V^2 \text{ with } j \notin \{1, i\} \tag{2.8}$$

Together, these constraints force every tour to include the first vertex. This vertex is the only one that can produce flow because it is not constrained by (2.7). So, all vertices must be connected by edges with the first vertex which results in one tour by constraints (2.2) and (2.3).

Theorem 3. *Minimizing the function* (2.1) *subject to the assignment constraints* (2.2) *and* (2.3) *and the flow sub-tour elimination constraints* (2.7) *and* (2.8) *results in one tour visiting all vertices exactly once.*

Proof. "⇐" Let $t = 1, 2, \ldots, n, 1$ be a tour visiting all vertices. By giving the variables $y_{12}, y_{23}, \ldots, y_{n1}$ the values $n-1, n-2, \ldots, 0$, respectively, the flow constraints (2.7) and (2.8) are satisfied.

"⇒" Let $k \geqslant 1$ and let $t = \{w_1, w_2, \ldots, w_k, w_1\}$ be a sub tour that does not contain vertex 1. Let $f = y_{w_1 w_2}$. Then from constraint (2.7) it follows that $y_{w_2 w_3} = f - 1$ and $y_{w_k w_1} = f - (k-1)$. Applying these values results in:

$$\sum_{\substack{j=1 \\ i \neq j}}^{n} y_{jw_1} - \sum_{\substack{j=2 \\ i \neq j}}^{n} y_{w_1 j} = y_{w_k w_1} - y_{w_1 w_2} = (f - (k-1)) - f = 1 - k$$

which contradicts (2.7) because $k \geqslant 1$. □

2.3.4 Experiments

We implemented two of the three formulations given in the previous sections. We used the free solver lp_solve [EN09] and the commercial solver CPLEX [ILO09]. We did not implement the sub-tour formulation due to the exponential number of constraints.

All test instances have been extracted from real products. The sizes of the point sets varies from 26 to 1091 points. In practice, the average number of points is about 200 with a few outliers of up to 1000 points. In Figures 2.3 and 2.4, graphical representations of two instances and their solutions are given. The distance metric used for solving these instances is the Euclidean metric, also known as the L^2 metric. The results are shown in the corresponding columns of Table 2.1.

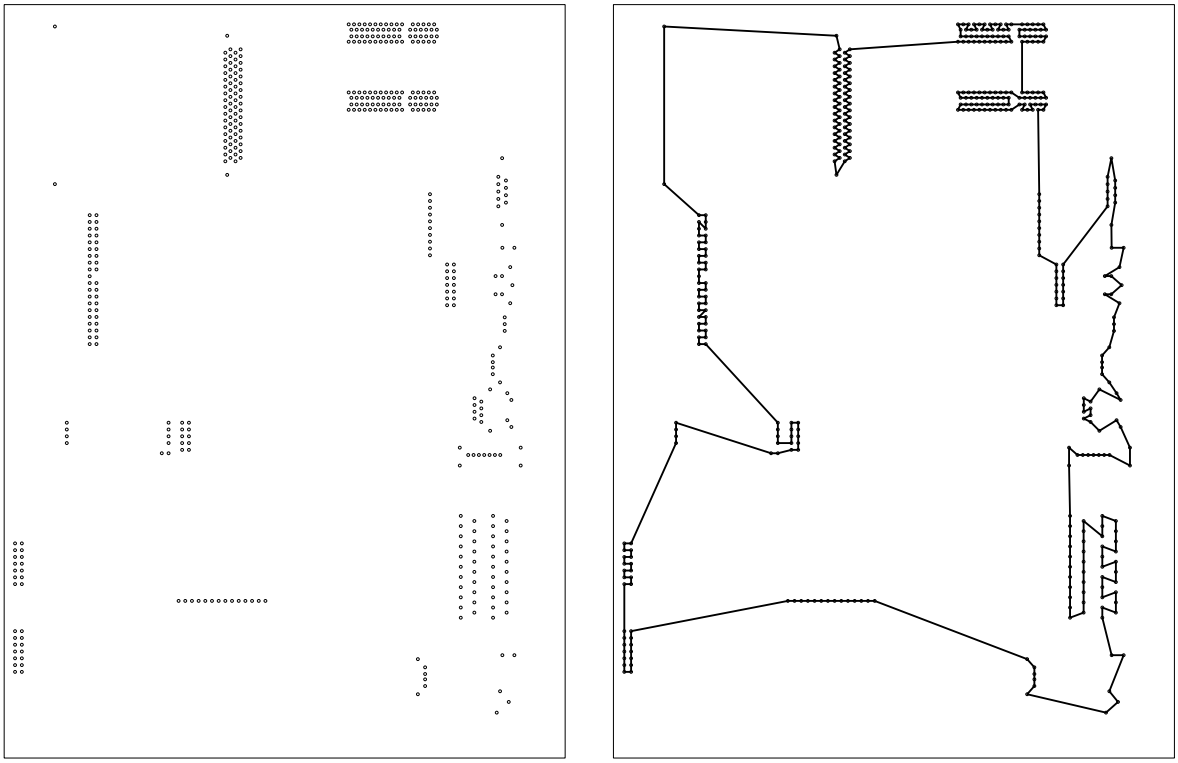
Problem instance	Points	lp_solve[1]		CPLEX[2]		Concorde[3]
		Sequential	Flow	Sequential	Flow	
6001-0200-6700	26	-	6.75s	3.39s	0.55s	0.03s
6001-0702-5701	46	-	-	-	28.92s	0.08s
6107-0702-2900	186	-	-	-	-	4.81s
6265-0800-5500	270	-	-	-	-	1.17s
6159-0300-1903	386	-	-	-	-	44.81s
6180-0600-3901	432	-	-	-	-	12.80s
6143-0400-1200	1091	-	-	-	-	1021.24s

Table 2.1: test results

For most of the test instances no optimal solution is found by the formulations withing a time frame of 8 hours. In Table 2.1, the unsolved instances are represented by empty cells. Our results are disappointing because solving instances with more than 100 points are impractical with these formulations. This even holds for CPLEX, a state-of-the-art commercial IP solver. Due to this observation we tried a different approach to solve these instances. We used the state-of-the-art TSP solver Concorde [Coo05].

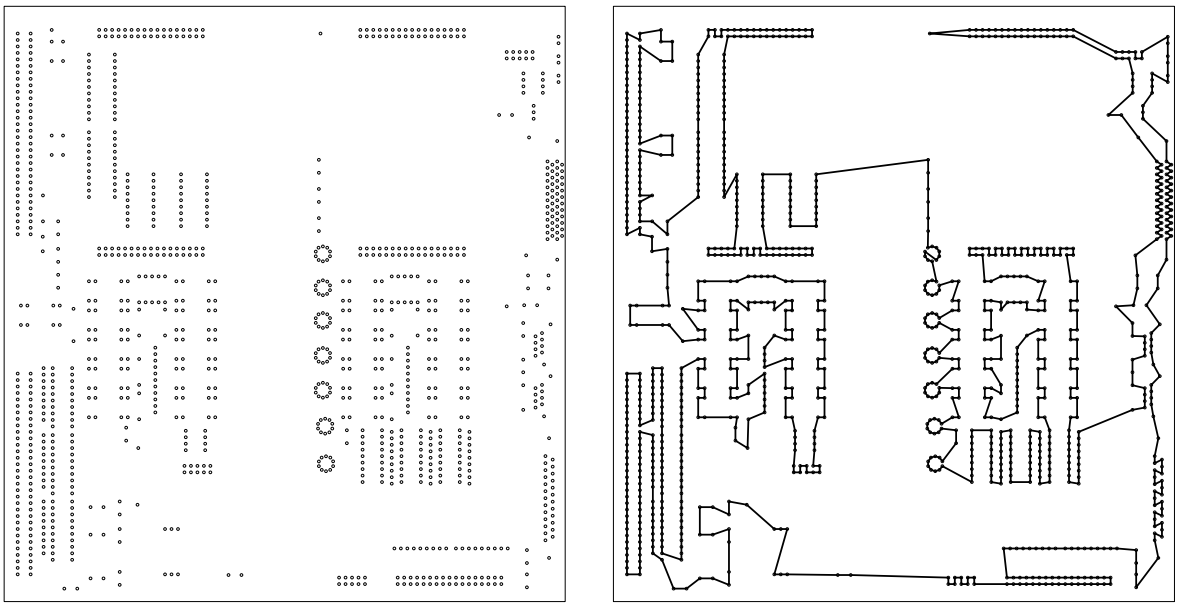
Concorde is a symmetric TSP solver that has solved 107 of the 110 TSPlib instances [Rei08] to optimality. The Concorde uses a cutting-plane algorithm that is described by Dantzig et al. [DFJ54]. The technique first uses an LP relaxation of the TSP formulation by using the assignment constraints and allowing the edge variables to obtain values between zero and one. It iteratively solves the LP and adds cutting planes to the model to improve the solution until all edge variables have a zero-one values. These cutting planes are efficiently found by a number of algorithms. The results we obtained by Concorde are shown in the last column of Table 2.1. The solutions of the two instances, presented in Figure 2.3 and 2.4, are obtained by Concorde.

[1] lp_solve version 5.5.0.14 used on an Intel Pentium D 3Ghz - 1GB RAM
[2] CPLEX version 10 used on a AuthenticAMD 2x Dual Core Opteron 270 - 16GB RAM
[3] Concorde 1.1 used on an Intel Pentium D 3Ghz - 1GB RAM



(a) (b)

Figure 2.3: Test instance 6180-0600-3901 with 432 pins (a) and the solution (b).



(a) (b)

Figure 2.4: Test instance 6143-0400-1200 with 1091 pins (a) and the solution (b).

Chapter 3

Solder Actions

Soldering with a tin fountain is a complex process which makes finding good solder actions with respect to solder quality difficult (Section 3.2). First, we introduce what solder actions are and their corresponding solder areas (Section 3.1). With the definition of the solder actions and areas, we give a (simplified) formal description of the problem of finding solder actions (Section 3.3) and present an algorithm that obtains possible solder actions (Section 3.4). This algorithm can be seen as a first step toward finding good solder actions.

3.1 Solder and Avoid Areas

A solder area is an area that is soldered by the tin fountain when it touches the board. We assume that all pins inside this area are correctly soldered. We introduce the avoid area to make sure that exterior of this area is definitely not soldered. Both areas depend on the nozzle used and the distance d between the nozzle and the board. In Figure 3.1, two instances show that by moving the nozzle away from the board the solder area gets smaller and by moving to the board the solder area gets bigger. The size of the solder and avoid area is a parabolic function of d and the nozzle radius r.

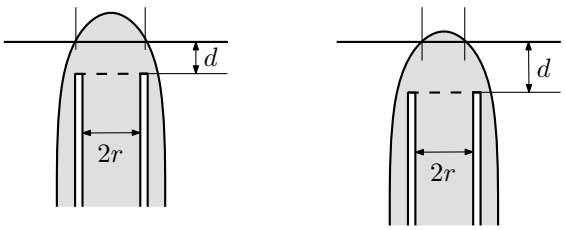
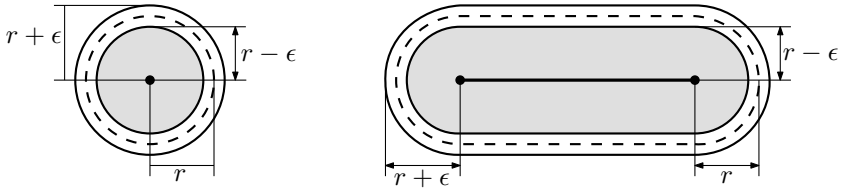


Figure 3.1: Schematic cross section of a nozzle with a tin fountain.

To avoid extra complexity in our model, we simplify the physical model by using only two z-coordinates. The z-coordinate indicates whether the machine is soldering or not. The



(a) (b)

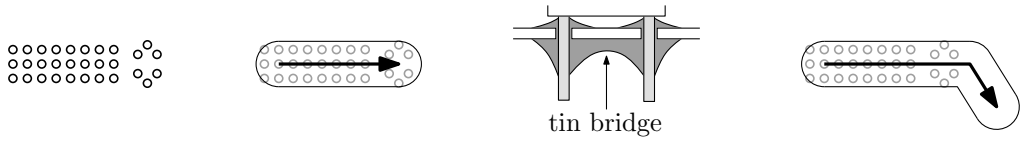
Figure 3.2: The solder and avoid areas of point-solder action (a) and track-solder action (b) with a nozzle of radius r.

disadvantage of the two state approach is that we are restricted to a single diameter for the solder area.

The machine can perform two soldering action; point- and track-solder actions. The point-solder action only moves the nozzle toward and away from the board. The area that is soldered is a disk-shape area with a radius of $r - \epsilon$, where ϵ is a small distance to make sure that the area is correctly soldered. To avoid contact with keep-out areas we introduce another area, the avoid area. This is a disk-shaped area with a radius of $r + \epsilon$, where ϵ is a small distance to make sure that no keep-out area is soldered. In Figure 3.2a, the area is shown for the point-solder action. The track-solder action starts soldering at the start position and moves to the end position. At the end position the nozzle stops with soldering by moving the nozzle away from the board. All points between the start and end position are soldered. The solder and avoid areas are hippodromes, see Figure 3.2b.

3.2 Why is Soldering Difficult?

The process of soldering with a tin fountain is a complex process. The physical behavior is only predictable by a very complex model. The complexity makes predicting the result as hard as predicting the weather. To show how difficult the soldering process is we look at an example given in Figure 3.3a. The example shows a footprint of a component with a high pin density. Soldering the pins by the solder action (Figure 3.3b), results in a short circuit. Due to the little space between the pins, the tin experience enough adhesion force to create



(a) (b) (c) (d)

Figure 3.3: An example to show how difficult it is to find a good solder action. We show a footprint of a component (a) where the pins are close to each other. By using solder action (b), a tin bridge is created between two pins (c), resulting in a short circuit. Using instead a solder action (d) does not result in a short circuit.

a tin bridge between pins, see Figure 3.3c. This bridge connects both pins, resulting in a short circuit. To avoid this behavior, operators define a solder action that moves away from the pins at certain angle, see Figure 3.3d. To predict this behavior many variables must be known, like the pin lengths, pin density, pad sizes, flux mixture composition, tin composition, tin temperature, board and component temperature and many more variables.

Despite the difficulties of finding good solder actions we simplified the problem by only looking at solder areas that have the minimal requirement of touching the pins.

3.3 Formal Description

We describe a simplified model that is used to find solder actions. The outcome of these solder actions may not solder all pins correctly, like the example given in the Section 3.2.

We denote the set of solder units by U and the set of pins by $\mathcal{S}$. Per solder unit $u \in U$, we search for a set of solder actions that solder all pins. It is not always possible to solder all pins due to the keep-out areas present on the board. We assume that for every pin $s \in \mathcal{S}$ there is at least one unit that can solder s. The solder actions can be seen as hippodromes and therefore we search for a set $\mathcal{H}$ of hippodromes that cover all pins in $\mathcal{S}$. These hippodromes may not overlap the keep-out areas in $\mathcal{K}$. We want a set $\mathcal{H}$ that minimize a function of the number of hippodromes and the total length of the hippodromes. Let n be the number of hippodromes and let l be the total length of the hippodromes. We want to minimize $n \cdot c^{\text{dip}} + l \cdot c^{\text{solder}}$, where c^{dip} is the cost for moving the nozzle toward and away from the board and c^{solder} is the cost for moving the nozzle while soldering. In Section 4.1, we explain more solder times which can be used in the objective function. We use here a simplified cost model.

With the start and end points of the hippodromes we construct a graph $G_u = (V_u, E_u)$ where every start and end point of a hippodrome is represented by a vertex in V_u. Let v and w be the start and end vertex of a hippodrome then the edges (v, w) and (w, v) are in the graph. The hippodromes that are restricted to be performed in one direction have one directed edge in the graph. The graphs constructed are used as input for the solder path algorithm.

3.4 Disk Cover Method

We investigated the possibility to automatically find the solder actions by use of a minimum disk cover algorithm. The idea behind this is that a disk is like a point-solder action which can be change to a track-solder action by connecting multiple disks. Connecting multiple disks is left out because the solder path algorithm automatically decides which point-solder actions are transformed to track-solder actions.

Problem (Minimum Disk Cover)**.** *Given a set $\mathcal{S}$ with point in the plane and a radius r, cover the set $\mathcal{S}$ with a set $\mathcal{D}$ of disks of radius r such that the number of disks is minimized.*

Since the minimum disk cover problem is $\mathcal{NP}$-hard [CCJ90], we know that we cannot solve the problem in polynomial time, unless $\mathcal{P} = \mathcal{NP}$. With this in mind we focus on a heuristic approach to obtain a reasonable solution. Gulczynski et al. [GHP06] discussed a number of

such heuristic approaches; the tiling method [DYC07], the Steiner-zone method and other heuristics. They presented some numerical results where the Steiner-zone method preformed best in six of the seven problem instances.

Tiling Method

Dong et al. [DYC07] presented an approach to tackle the disk covering problem by tiling with hexagon tiles. The first step in the algorithm is to cover the plane with hexagon tiles. The edges of the hexagons have the same length as the radius of the disks. By placing disks at the corners of the hexagons, every point in the plane is covered. Some disks do not cover any point and some points are covered by two disks. These redundant disks are removed, if possible. After removing these disks the algorithm checks if two disks can be merged. After merging two disks the algorithm checks again for possible merge actions and stops if no merge action is possible. Dong et al. only presented an approach and did not give a complexity bound of the algorithm.

More theoretical results are achieved by Călinescu et al. [CMWZ01]. They presented a polynomial-time disk cover algorithm with an approximation factor of 108. This algorithm uses triangles to cover the plane. Narayanappa and Vojtěchovský improved [NV06] the approximation factor to 72 by using hexagon tiling. They also proved that no better bound can be achieved using the algorithm described by Călinescu et al.

Steiner-Zone Method

The Steiner-zone method is a greedy algorithm that uses the overlapping regions, so-called Steiner zones, as possible locations for the disks. Let $D(p, r)$ be a disk centered at p with a radius of r. A Steiner zone of k points is defined by a non-empty set $D(1, r) \cap D(2, r) \cap \cdots \cap D(k, r)$. The degree of such Steiner zone is k, the number of disks that overlap each other, see Figure 3.4. First the algorithm place a disk centered at any point inside the Steiner zones with the highest degree. Next, it update all Steiner zones by removing the points that are covered by the new disk. Iteratively, the algorithm places and updates the Steiner zones until all points are covered.

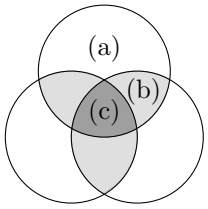


Figure 3.4: Three disks with their Steiner zones of degree (a) one, (b) two and (c) three.

Franceschetti et al. [FCB01] showed that the approximation factor of a greedy disk cover algorithm, such as the Steiner-zone method, is $O(\ln n)$ where n is the number of points in the input.

Improvement of the Steiner-Zone Method

We improved the Steiner-zone method to avoid soldering keep-out areas. We remove the parts of the Steiner zones where no disks can be placed. These areas overlap with the Minkowski sum of the keep-out areas and a nozzle-size disk. The remaining areas are used for placing the disks. A result of the Steiner-zone method avoiding keep-out areas is given in Figure 3.5.

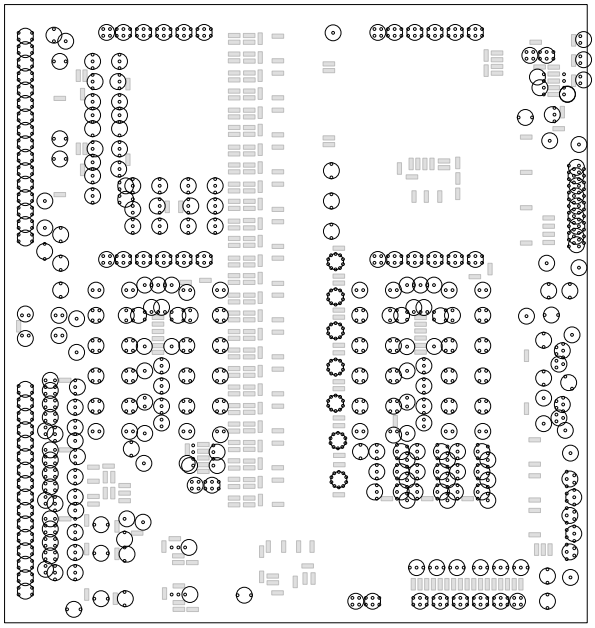


Figure 3.5: Improvement of the Steiner-zone method to avoiding keep-out areas. Note that some of the pins are not covered due to near-by keep-out areas.

3.5 Results

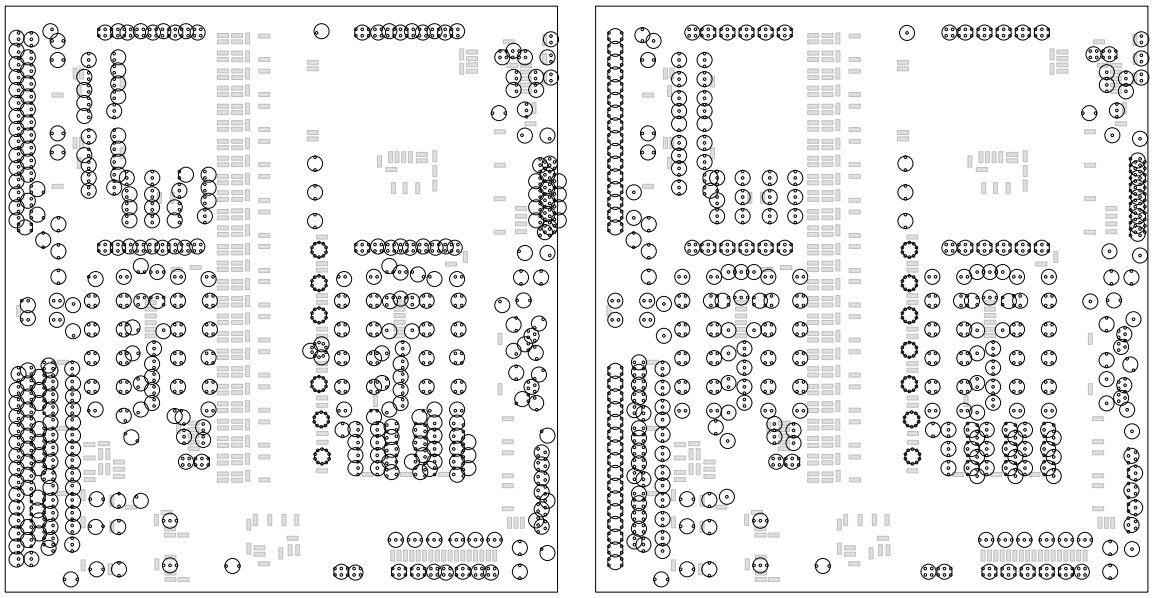
Looking at the results obtained from both algorithms (see Figure 3.6), we can conclude that the Steiner-zone method performed best on the given problem instance. Take, for example, the upper left corner of the board where the tiling method uses two rows and the Steiner-zone method only one.

Unfortunately, there are places where the Steiner-zone method produces a messy cover, see Figure 3.7a. Connecting these disks may result in unwanted zig-zag patterns and more track-action than needed (Figure 3.7b). These zig-zag patterns are unwanted because they decrease the soldering quality.

The disk cover algorithm only tries to minimize the number of disk used to cover the pins. In our problem we want more than that. We also want to minimize the total length of the hippodromes.

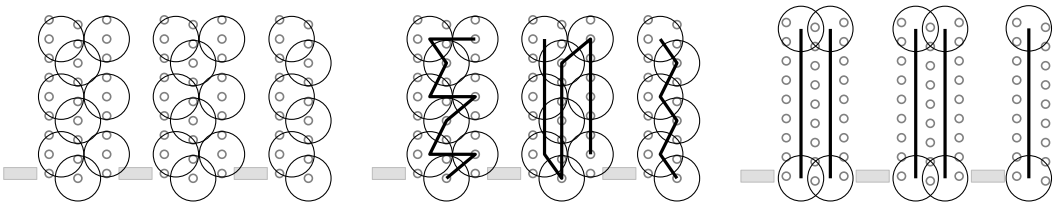
We introduced in this chapter a approach to tackle the problem of finding solder actions. Unfortunately, due to the complexity of the problem it is almost impossible to find always

good solder actions. The algorithm presented is a first step toward finding nearly good solder actions.



(a) (b)

Figure 3.6: Result of the minimum disk cover algorithms; (a) tiling method result with 402 disks and (b) Steiner-zone method result with 331 disks.



(a) (b) (c)

Figure 3.7: In this example we show a result of the Steiner-zone method (a). We connected the disks with lines (b). Note that these lines are just guesses of what the solder path algorithm could produce. An operator defines the solder actions by straight lines (c).

Chapter 4

Solder Paths

In the previous chapter we gave an algorithm that produces graphs that represent the solder actions. In this chapter we use these graphs to find multiple solder paths consisting of solder actions of the given graphs. The union of the solder actions used in these paths solder all components present on the board. We use an IP approach to find multiple solder paths that minimizes the total processing time of the machine.

We start by describing the exact cost model of the problem (Section 4.1). The cost model leads to a formal description of the problem (Section 4.2). Next, we construct a graph containing the information of all solder actions found in the previous chapter (Section 4.3). The graph is used as basis for the two mathematical models; the single solder unit model (Section 4.4) and the multiple solder unit model (Section 4.5). We conclude this chapter with some model improvements to get a faster and better result (Section 4.6).

4.1 Cost Model

It is important that the costs in our model correctly reflect the real costs. By costs we mean the time needed to perform an action, like moving the nozzle from a to b. We can express the processing time of an action by seven different types of costs, see Figure 4.1.

Below we describe the seven types of costs in more detail.

c^{shift} Time needed for the movement of the nozzle in z-direction. This cost is proportional to the distance. In our model we use Euclidean distances.

c_{ij}^{move} Time needed for moving the nozzle from position i to j, while not soldering.

c_{ij}^{solder} Time needed for moving the nozzle from position i to j, while soldering. Moving while soldering is about 20 times slower then moving without soldering.

c_i^{dip} Time needed to wait at position i in order to get a good quality soldering joint.

c_i^{begin} Time needed to wait at position i before starting with soldering. More time is needed to ensure soldering quality if the component has a high-temperature absorption coefficient.

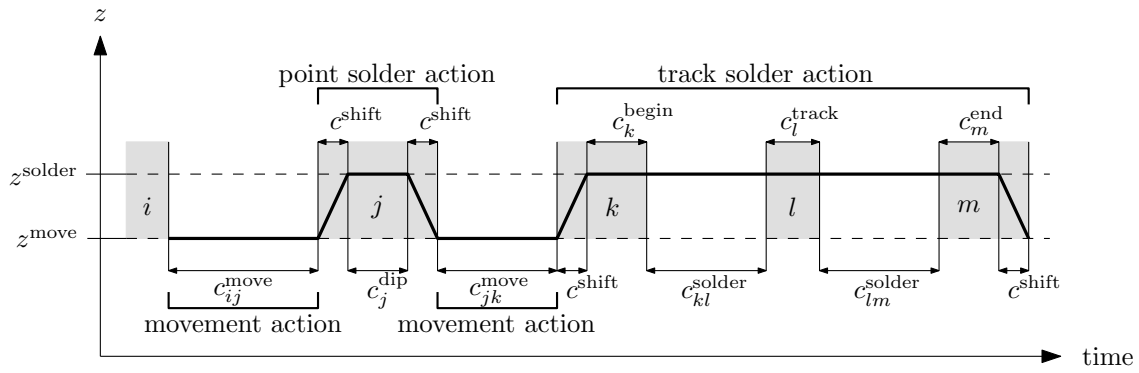


Figure 4.1: Processing times of the movement and soldering actions.

c_i^{end} Time needed to wait at position i after stopping with soldering. It is important to stop correctly after a track-solder action. It is possible that a short circuit is created when a track soldering action is stopped fast. Using extra time to heat and withdraw the tin it makes more likely to avoid short circuits.

c_i^{track} Time needed to wait at position i during a track-solder action. The nozzle simply waits before going to the next position, again in order to ensure the soldering quality.

As Figure 4.1 shows, we assign the z-movement costs to the soldering actions instead of the movement actions. The reason is that the nozzle does not touch the board at the start. So, we do not need an z-movement before moving to the first solder action.

4.2 Formal Problem Definition

Let U be the set of solder units, and let m be the number of solder units present in the machine. Per unit we have a directed graph $G_u = (V_u, E_u)$ that represents the solder and movement actions of unit u. The vertices in V_u represent locations on the board where an action may take place. There is a special vertex that is called *start vertex*; it represents the start location of the nozzle. The set E_u consists of two types of edges; *solder edges* and *movement edges*. The solder edges represent track-solder actions and the *movement edges* represent movement actions. For both type of edges if $(v, w) \in E_u$ then an action starts at v and stops at w. We use E to denote all edges in the graphs $G_1, G_2, \ldots, G_m$ and V all the vertices in the graphs. Let $\sigma(e)$ be the hippodrome-shaped area "along" e with nozzle-size radius. Let $\sigma(v)$ be a nozzle-size disk centered at v.

In the graph the solder actions are embedded by using vertices and edges. We denote the set of soldering actions in graph G by $A(G)$. A solder action $a \in A(G)$ is a non-empty sequence of vertices and edges. We use $V(a)$ to denote the set of vertices in a and $E(a)$ to denote the set of edges induced by a. Let $(v_1, v_2, \cdots, v_k)$ be such a sequence then (v_i, v_{i+1}) and (v_{i+1}, v_i) are solder edges for all $i = 1, 2, \ldots, k-1$. We denote the set with the first and last vertex of action a by $\varphi(a)$. A special case is when the $\varphi(a)$ is a singleton, representing a point-solder action. The actions with a larger sequence are track-solder actions. Given a vertex v we denote the action that includes this vertex as $\alpha(v)$. We use A to denote the union of all

solder actions in the graphs $G_1, G_2, \ldots, G_m$. Let $\sigma(a)$ be the area soldered by action a, that is,

$$\sigma(a) = \bigcup_{v \in V(a)} \sigma(v) \cup \bigcup_{e \in E(a)} \sigma(e).$$

We assume that for every pin at least one graph must exist that has a solder action that solders that pin, otherwise that pin cannot be soldered by the machine.

We define a path to be a sequence of soldering actions present in G_u. The actions can be used in normal or reversed order. Between every two neighboring actions in the path a movement or soldering edge is present. These edges connect the last vertex of the current action to the first vertex of the next action. We denote the set of all solder edges in a path p by $S(p)$ and the set of movement edges in p by $M(p)$. We use $V(p)$ to denote the set of all vertices in p and $E(p) = S(p) \cup M(p)$ to denote the set of all edges of p. Let $\sigma(p)$ be the area soldered by path p, that is,

$$\sigma(p) = \bigcup_{v \in V(p)} \sigma(v) \cup \bigcup_{e \in E(p)} \sigma(e) \supseteq \bigcup_{a \in p} \sigma(a).$$

A path p can be seen as a sequence of vertices. In order to get the correct cost function we need to recognize four different types of vertices in p. The vertex types depend on the edge types used to connect the vertices of the actions in p.

- $B(p)$ is the set of vertices in p where the edge type changes from moving to soldering including the begin vertex of p if it has an adjacent soldering edge.
- $F(p)$ is the set of vertices in p where the edge type changes from soldering to moving including the end vertex of p if it has an adjacent soldering edge.
- $D(p)$ is the set of vertices in p that are connected to only movement edges including the begin and end vertex of p if they have an adjacent movement edge.
- $T(p)$ is the set of vertices in p that are connected to only soldering edges.

Now we can give a formal definition of the Minimum-Cost Solder Paths Problem.

Problem (Minimum-Cost Solder Paths)**.** *Given graphs $G_1, G_2, \ldots, G_m$ with the solder and movement actions per unit, a set $\mathcal{S}$ with pin locations, a set $\mathcal{K}$ of keep-out areas and the costs:*

- c_{ij}^{move} *per solder edge* $(i,j) \in E$
- c_{ij}^{solder} *per movement edge* $(i,j) \in E$
- c_i^{dip}, c_i^{begin}, c_i^{end}, c_i^{track} *per vertex* $i \in V$,

find for every solder unit $1 \leqslant u \leqslant m$, *a path* p_u *in* G_u *such that:*

(i) path p_u *starts with the start vertex of* G_u,

(ii) no keep-out area is soldered in path p_u, that is,

$$\mathcal{K} \cap \bigcup_{u \in U} \sigma(p_u) = \emptyset$$

(iii) the support bar is avoided, that is, the movement action of the solder nozzle is performed at a large distance from the board and the vertical projection of the support bar on the board is treated as a keep-out area,

(iv) every pin is soldered by at least one unit, that is,

$$\mathcal{S} \subseteq \bigcup_{u \in U} \sigma(p_u)$$

(v) the throughput of boards through the machine is maximized, that is,

$$\max_{u \in U} \{PathCost(p_u)\},$$

is minimized, where $PathCost(p)$ is the time needed to process all actions in p, that is,

$$\begin{aligned} PathCost(p) = & \sum_{(i,j) \in S(p)} c_{ij}^{\text{solder}} + \sum_{(i,j) \in M(p)} c_{ij}^{\text{move}} + \\ & \sum_{i \in B(p)} c_i^{\text{begin}} + \sum_{i \in F(p)} c_i^{\text{end}} + \sum_{i \in D(p)} c_i^{\text{dip}} + \sum_{i \in T(p)} c_i^{\text{track}} + \\ & (|B(p)| + |F(p)| + 2|D(p)|) \cdot c^{\text{shift}} \end{aligned}$$

4.3 Graph Construction

We have m graphs $G_1, G_2, \ldots, G_m$ with solder actions obtained in the first part of the algorithm (Chapter 3). We combine these graphs into a new graph, the *model graph*, as detailed below. With the model graph and a modified TSP formulation we want to find the best sequence of solder actions.

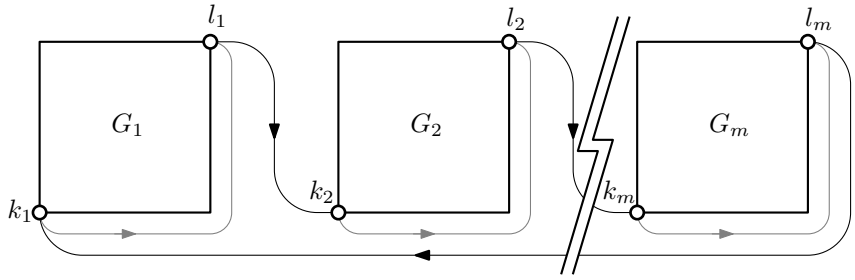


Figure 4.2: Graph construction

To create the model graph, we first add for each solder unit $u \in U$ a start vertex k_u and stop vertex l_u, see Figure 4.2. The start vertex k_u represents the start position of the nozzle of u.

We use the origin of the board as start position. The stop vertices are used to connect all m graphs. After a tour is obtained, the stop vertices are removed. This results in m paths that consist of solder actions of the units.

Per solder unit u, we have to connect the solder actions in all possible ways by adding edges to G_u. We use two kind of edges; movement edges and solder edges. The movement edges represent a movement action of the machine where the tin fountain does not touch the board. The solder edges are the same but then the tin fountain touches the board.

Every action in the G_u is connected to all other actions in the graph. We use only the first and last vertex of an action. Every pair of these vertices is connected by two directed movement and solder edges one of each type in each direction. We add both edge types to the G_u since we cannot compute which edge is taken. The choice of edge type depends on the edges used before and after the edge, see Section 4.6.4 for reducing the number of edges. There are no edges added between vertices of the same action. The costs of the new edges are corresponding to the edge types and the distance between the vertices.

To avoid keep-out areas some of the solder edges may not be added or must be replaced by detour solder edges. Not adding these edges makes sure that they never occur in the solution of the model. The detour edges make sure that the keep-out areas are avoided and therefore not soldered. For the support bar area we cannot use solder edges with a detour. So, all solder edges crossing the support bar area must be removed. The only actions allowed to cross the support bar are movement actions. These actions must be performed at a higher distance from the board to avoid hitting the support bar. We add to the corresponding movement edges an extra cost representing the cost of the extra z-movement.

We connect the start vertex by outgoing movement edges to every first and last vertex of an action. These edges have the cost of moving. Equally, we add incoming movement edges to the stop vertex. These movement edges have zero cost because the machine can move board and nozzle simultaneously. If the machine has a settable starting position of the nozzle then this can be included in the model by using zero cost for the edges that are adjacent to the start vertex. We add these edges to each graph G_1, G_2, ..., G_m.

The algorithm ExpandSolderUnitGraph, see Algorithm 1, constructs the model graph according to the specification described in this section. Note that we require the action of the start and end vertex of a graph to be not equal, hence $\alpha(b) \neq \alpha(e)$.

Next, we have to connect all graphs to obtain one graph that can be used by the modified TSP formulation to obtain a tour. We use the stop vertices of the units to connect to the start vertex of the next unit. For the last unit we connect the stop vertex to the start vertex of the first unit. These edges are directed edges and therefore the resulting tour always starts at the first unit. Observe that the stop vertices are always in the tour. We can remove these vertices and the adjacent edges to obtain m separate paths.

Algorithm ConstructModelGraph creates the graph for m soldering units.

Algorithm 1: ExpandSolderUnitGraph(G, k, l)

input : Graph G and start vertex k and stop vertex l
output: Graph G with added movement and solder edges

Add movement edge (k, l) to G with cost 0
for *each vertex* $v \in G \setminus \{k, l\}$ **do**
  if $v \in \varphi(\alpha(v))$ **then**
    Add movement edge (k, v) to G with corresponding move cost
    Add movement edge (v, l) to G with cost 0
for *each pair of vertices* $\{v, w\} \subseteq G \setminus \{k, l\}$ **do**
  if $\alpha(v) \neq \alpha(w)$ **and** $v \in \varphi(\alpha(v))$ **and** $w \in \varphi(\alpha(w))$ **then**
    if $\sigma((v, w)) \cap K = \emptyset$ **then**
      Add solder edges (v, w) and (w, v) to G with corresponding solder cost
    if (v, w) *crosses the support bar* **then**
      Add movement edge (v, w) and (w, v) to G with corresponding move cost and extra z-movement cost
    else
      Add movement edge (v, w) and (w, v) to G with corresponding move cost
return G

Algorithm 2: ConstructModelGraph(G_1, G_2, ..., G_m)

input : Graphs $G_1, G_2 \ldots, G_m$ with solder actions, where m is the number of units.
output: The model graph G

for $1 \leqslant i \leqslant m$ **do**
  Add start vertex k_i and stop vertex l_i to G_i
  ExpandSolderUnitGraph(G_i, k_i, l_i)

for $1 \leqslant i \leqslant m$ **do**
  Add edge $(l_i, k_{(i \bmod m)+1})$ with cost 0

4.4 Single Solder Unit Model

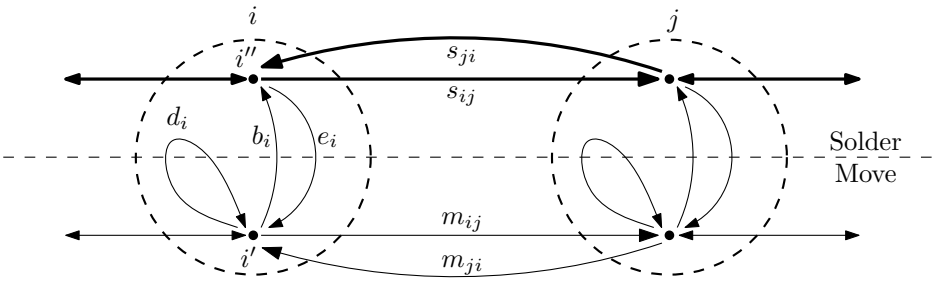


Figure 4.3: Graph construction for a realistic cost model.

In the single unit model we use the model graph that is constructed in the previous section. The vertices of the model graph are expanded using *internal* edges and vertices. We refer to

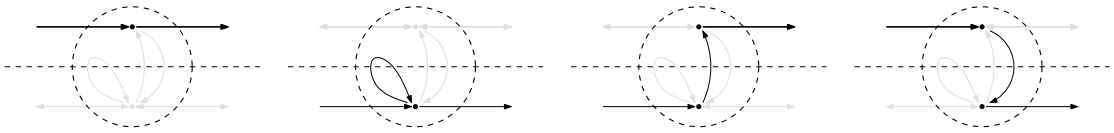
these expanded vertices as super vertices. The internal edges represent the costs for starting and stopping with soldering and for point soldering. In Figure 4.3 the graph construction is given for two super vertices i and j. Super vertices i and j are illustrated by a dashed circle that surrounds the internal edges and vertices. The inner vertex i' is called the *move vertex* of i because only movement edges are connected to this vertex. The inner vertex i'' is called the *solder vertex* of i. Edge d_i represents the cost of the point-soldering action at super vertex i. Edges b_i and e_i carry the costs for starting and stopping with soldering for a track soldering action at super vertex i, respectively.

Every edge in the expanded model graph has a corresponding zero-one variable in the IP model. The variables have the same name as the corresponding edge. We use the same approach as the standard TSP formulation, that is, if the edge b_i is used then the variable b_i is one.

If we construct the expanded model graph like this, we have to avoid some combination of edges. We force that only one of the internal edges b_i, e_i and d_i can be used. This can be achieved by introducing the constraint

$$d_i + b_i + e_i \leqslant 1 \qquad \text{for } i \in V. \tag{4.1}$$

Further constraints are needed to model the solder behavior correctly. There are four ways in which a super vertex i can be visited, see Figure 4.4. The first way (a) is by only using



(a) solder edges (b) movement edges (c) start with soldering (d) stop with soldering

Figure 4.4: Four ways of using the internal edges of a super vertex.

soldering edges. The move vertex and the internal edges are not used. The second way (b) is by using only movement edges and the point-soldering edge. The solder vertex is not used. The third (c) and fourth way (d) use the start- and stop-soldering edge, respectively. In these cases one movement edge and one soldering edge must be used. These four ways can be forced by the following constraints.

$$\sum_{j \in V \setminus \{i\}} [m_{ij} + s_{ij}] = 1 \qquad \text{for } i \in V \tag{4.2}$$

$$\sum_{j \in V \setminus \{i\}} [m_{ji} + s_{ji}] = 1 \qquad \text{for } i \in V \tag{4.3}$$

$$e_i + \sum_{j \in V \setminus \{i\}} s_{ij} = 1 - d_i \qquad \text{for } i \in V \tag{4.4}$$

$$b_i + \sum_{j \in V \setminus \{i\}} s_{ji} = 1 - d_i \qquad \text{for } i \in V \tag{4.5}$$

These four set of constraints replace the assignment constraints of the standard TSP formulation. These constraint force that every vertex has a degree of two, see Section 2.3.

Constraints (4.2) and (4.3) are the same as in the standard TSP formulation with respect to the super vertices. Constraints (4.4) and (4.5) are used to restrict the degree of solder vertex of i. The left-hand sides of these constraints restrict the solder vertex to have a degree of two, one outgoing (4.4) and one incoming (4.5). The right-hand sides make it possible that the solder vertex is not used, having a degree of zero. This is only possible if edge d_i is used.

Lemma 1. *Given constraints* (4.1)–(4.5)*, the internal edges of a super vertex i are uniquely determined by the use of movement and solder edges incident to i.*

Proof. By definition, the solder edges are only connected to the solder vertex of i and the movement edges only to the move vertex of i. Given (4.2) and (4.3), the number of incoming and outgoing edges, with respect to vertex i, is one. So, the solder vertex and the move vertex of i have each at most one incoming and one outgoing edge. This results in the four cases illustrated in Figure 4.4.

(a) *Two solder edges* (Figure 4.4a)
The super vertex has one incoming and one outgoing solder edge. The constraints (4.4) and (4.5) are only satisfied if edges b_i, e_i and d_i are not used.

(b) *Two movement edges* (Figure 4.4b)
No solder edges are used and therefore (4.4) and (4.5) can only be satisfied by using b_i and e_i or by using d_i. It is not allowed to use both b_i and e_i by (4.1). Thus, d_i must be used to satisfy the constraints.

(c) *One incoming movement edge and one outgoing solder edge* (Figure 4.4c)
If one outgoing solder edge is used, the edges e_i and d_i cannot be used according to (4.4); by (4.5) the edge b_i must be used.

(d) *One incoming solder edge and one outgoing movement edge* (Figure 4.4d)
According to (4.4), the edges b_i and d_i cannot be used and by (4.5) edge e_i must be used.

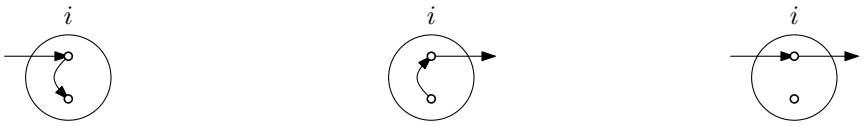
This case analysis completes the proof of the lemma. □

Some solder edges represent track-solder actions that solder pins. The next constraint force one of the solder edges between two super vertices to be used. It can happen that only one direction is present in the model. Then the other direction must be removed from the constraint or forced to be zero, that is,

$$s_{ij} + s_{ji} = 1 \qquad \text{for } a \in A \text{ and } (i,j) \in E(a). \tag{4.6}$$

Note that the constraint is only applicable for the single unit model.

Every super vertex has internal edges that directly correspond to the cost of the soldering actions. Edge d_i has cost c_i^{dip}, edge b_i has cost c_i^{begin} and edge e_i has cost c_i^{end}. For the solder edge and the movement edge from i to j, the costs are c_{ij}^{solder} and c_{ij}^{move}, respectively. The cost c_i^{track} does not directly correspond to an edge in the model. To realize the cost, we add



(a) extra cost 0 (b) extra cost 0 (c) extra cost c_i^{track}

Figure 4.5: Three cases visiting a super vertex i by using a soldering edge.

$c_i^{\text{track}}/2$ to the cost of every soldering edge connected to super vertex i and by subtracting $c_i^{\text{track}}/2$ from the internal edge b_i and edge e_i. This results in the following objective function.

$$\begin{aligned}\text{minimize} \sum_{\substack{i\in V\\ j\in V\setminus\{i\}}} \left[\left(c_{ij}^{\text{solder}} + \frac{c_i^{\text{track}} + c_j^{\text{track}}}{2}\right) s_{ij} + c_{ij}^{\text{move}} m_{ij}\right] + \\ \sum_{i\in V} \left[\left(c_i^{\text{dip}} + 2\cdot c^{\text{shift}}\right) d_i + \left(c_i^{\text{begin}} + c^{\text{shift}} - \frac{c_i^{\text{track}}}{2}\right) b_i + \right. \\ \left. \left(c_i^{\text{end}} + c^{\text{shift}} - \frac{c_i^{\text{track}}}{2}\right) e_i\right] \qquad (4.7)\end{aligned}$$

The stop vertices present in the IP model must have zero cost. Therefore these vertices are not included in the objective function.

Observation 1. *The costs in the objective function are corresponding the costs given in Section 4.1.*

Lemma 2. *A super vertex i having two soldering edges results in a cost of extra c_i^{track} in the objective function* (4.7)*.*

Proof. The cost c_i^{track} is used in the objective function (4.7) by b_i, e_i and the soldering edges. A super vertex i can be visited in four different ways. The cost c_i^{track} is used in three of these four cases, see Figure 4.5. All these cases have at least one soldering edge.

(a) *one incoming solder edge*
The incoming solder edge adds $c_i^{\text{track}}/2$ and the edge e_i subtracts $c_i^{\text{track}}/2$ to the objective function, which makes the extra cost zero.

(b) *one outgoing solder edge*
This case is equivalent to the first case. No extra cost is added to the objective function.

(c) *two solder edges*
Two solder edges induce an extra cost of c_i^{track}.

These cases prove the lemma. □

We want to obtain a solution of the IP model that corresponds to a single tour. In order to achieve this we need the sub-tour elimination constraints (SECs) that are described in Section 2.3. To complete the model, we use the flow SECs (2.7) and (2.8) where the super

vertices represent the vertices in the TSP formulation. The variable x_{ij} is replaced by $m_{ij}+s_{ij}$, see constraint (4.8). The other SEC versions can also be used for the IP model.

$$0 \leqslant y_{ij} \leqslant (n-1)\,[m_{ij} + s_{ij}] \qquad \text{for } (i,j) \in V^2 \text{ with } j \notin \{1, i\} \tag{4.8}$$

Lemma 3. *Any solution of the IP model results in a single tour of super vertices.*

Proof. According to (4.2) and (4.3), the super vertices are forced to have degree two which is equivalent to the vertices in the TSP problem. With the SECs (2.7) and (2.8) the subtours of the super vertices are eliminated making the solution of the IP a single tour of super vertices. □

Correctness

We prove that the given IP model solves the Minimum-Cost Solder Problem with *one unit*. We refer to the *solder model* as the model in reality.

Theorem 4. *Any solution of the model described by constraints* (2.7), (4.1)–(4.8) *that minimizes the objective function* (4.7) *is a solution of the* Minimum-Cost Solder Problem *with one unit.*

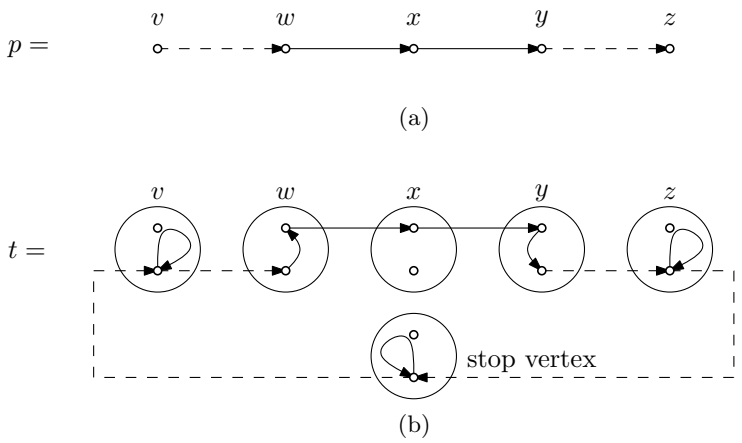


Figure 4.6: In (a) p is a correct path in the solder model. It uses all different actions possible. In the IP model the path p is transformed into a tour t (b).

Proof. "⇒" First we prove that a path in the solder model can be transformed into a correct tour in the IP model with the same cost.

Let p be a path in the solder unit with the vertex sequence (v, w, x, y, z) and let $D(p) = \{v, z\}$, $B(p) = \{w\}$, $F(p) = \{y\}$, $T(p) = \{x\}$. This implies that $M(p) = \{(v, w), (y, z)\}$ and $S(p) = \{(w, x), (x, y)\}$. In Figure 4.6 the path p is shown in the solder model and a tour t in the IP model corresponding to path p. Tour t has six super vertices, five of the original problem and one stop vertex. By the graph construction the stop vertex has only movement edges and all edges of this super vertex have zero cost. For every edges in $S(p)$ a solder

edge is present in tour t between two corresponding super vertices. Equally, every edge in $M(p)$ correspond to a movement edge in t. Due to the use of movement and solder edges, the internal edges in the IP model are forced to be used (Lemma 1). The super vertices v and z have two movement edges connected, which results in edges d_v and d_z being used. Super vertices w and y have both a solder and movement edge. This forces edges b_w and e_y to be used because of the incoming movement edge of w and the incoming solder edge of y. We transformed the path p into a correct representation in the IP model by tour t. Tour t results in the following cost according to (4.7)

$$\left(c_{wx}^{\text{solder}} + \frac{c_w^{\text{track}} + c_x^{\text{track}}}{2}\right) + \left(c_{xy}^{\text{solder}} + \frac{c_x^{\text{track}} + c_y^{\text{track}}}{2}\right) + c_{vw}^{\text{move}} + c_{yz}^{\text{move}} + \left(c_v^{\text{dip}} + 2 \cdot c^{\text{shift}}\right) + \left(c_z^{\text{dip}} + 2 \cdot c^{\text{shift}}\right) + \left(c_w^{\text{begin}} + c^{\text{shift}} - \frac{c_w^{\text{track}}}{2}\right) + \left(c_y^{\text{end}} + c^{\text{shift}} - \frac{c_y^{\text{track}}}{2}\right).$$

This simplifies to

$$c_{wx}^{\text{solder}} + c_{xy}^{\text{solder}} + c_{vw}^{\text{move}} + c_{yz}^{\text{move}} + c_w^{\text{begin}} + c_y^{\text{end}} + c_v^{\text{dip}} + c_z^{\text{dip}} + c_x^{\text{track}} + 6 \cdot c^{\text{shift}},$$

which is equal to

$$PathCost(p).$$

"$\Leftarrow$" According to Lemma 3, solving the IP model results in a tour t of super vertices. Let p' be the path of super vertices that is obtained after removing the stop vertex and its adjacent edges from t. First, we transform p' to a path p in the solder model and then we show that p is the optimal solution of the Minimum-Cost Solder Path Problem.

We use as vertices in p the super vertices used by p'. The sequence of vertices in p is defined by the solder and movement edges of path p'. Let $S(p) = \{(i,j) \mid (i,j) \in E(p') \text{ and } s_{ij} = 1\}$ and let $M(p) = \{(i,j) \mid (i,j) \in E(p') \text{ and } m_{ij} = 1\}$. The path p' has four types of super vertices that are determined by the edges b_i, e_i and d_i for every $i \in V(p')$, see Lemma 1. Let $B(p)$, $F(p)$ and $D(p)$ be the set of super vertices i with $b_i = 1$, $e_i = 1$ and $d_i = 1$, respectively. Let $T(p)$ be the set of super vertices that are incident to two adjacent soldering edges ($b_i = e_i = d_i = 0$) on p.

Now we have to prove that p is a correct path and that p is an optimal solution of the solder model. Note that there is only one path in the solder model because we only use one unit. This implies that path p must have the following properties according the formal description (Section 4.2):

(i) *Path p must start with the starting vertex of the unit.*
By the construction of the model graph, we know that the stop vertex is connected by a directed edge to the start vertex of the unit. Removing the stop vertex makes the start vertex of the unit also the start vertex of p.

(ii) *Keep-out areas are not soldered.*
With the assumption that the solder actions produced in Chapter 3 do not overlap keep-out areas and by avoiding solder edges in the graph that overlap keep-out areas we know that no point of $\sigma(p)$ overlaps a keep-out area.

(iii) *Support bar is avoided.*
By construction of the model graph, the solder edges do not cross the support bar area and the movement edges have an extra cost to represent the movement of the nozzle at a larger distance to the board. Therefore we know that the support bar is avoided.

(iv) *Every pin must be soldered.*
Lemma 3 shows that the solution of the IP is a tour of super vertices. These super vertices are the vertices of p and therefore the corresponding solder actions are performed. The solder edges corresponding to the track-solder actions are present in the path due to constraint (4.6). By using all possible actions and with the assumption that every pin is soldered by at least one of the actions available, we know that all pins are soldered.

(v) *Throughput of boards is maximized.*
We have one path p in the soldering model. By the definition of the soldering model the cost of this path must be minimized.

The set of super vertices with two soldering edges is $T(p)$. According to Lemma 2, we can state that

$$\sum_{i \in T(p)} c_i^{\text{track}} = \sum_{\substack{i \in V(p') \\ j \in V(p') \setminus \{i\}}} \frac{c_i^{\text{track}} + c_j^{\text{track}}}{2} s_{ij} + \sum_{i \in V(p')} \left[-\frac{c_i^{\text{track}}}{2} b_i - \frac{c_i^{\text{track}}}{2} e_i \right]. \tag{4.9}$$

We show that the minimizing function of the IP model is equal to the objective of the solder model.

$$\begin{aligned} &\sum_{\substack{i \in V(p') \\ j \in V(p') \setminus \{i\}}} \left[\left(c_{ij}^{\text{solder}} + \frac{c_i^{\text{track}} + c_j^{\text{track}}}{2} \right) s_{ij} + c_{ij}^{\text{move}} m_{ij} \right] + \\ &\sum_{i \in V(p')} \left[\left(c_i^{\text{dip}} + 2 \cdot c^{\text{shift}} \right) d_i + \left(c_i^{\text{begin}} + c^{\text{shift}} - \frac{c_i^{\text{track}}}{2} \right) b_i + \right. \\ &\qquad \left. \left(c_i^{\text{end}} + c^{\text{shift}} - \frac{c_i^{\text{track}}}{2} \right) e_i \right]. \end{aligned}$$

Rewriting this formula yields

$$\sum_{\substack{i\in V(p')\\ j\in V(p')\setminus\{i\}}} c_{ij}^{\text{solder}} s_{ij} + \sum_{\substack{i\in V(p')\\ j\in V(p')\setminus\{i\}}} c_{ij}^{\text{move}} m_{ij} +$$
$$\sum_{i\in V(p')} \left[c_i^{\text{begin}} b_i + c_i^{\text{end}} e_i + c_i^{\text{dip}} d_i \right] +$$
$$\sum_{i\in V(p')} \left[c^{\text{shift}} b_i + c^{\text{shift}} e_i + 2 \cdot c^{\text{shift}} d_i \right] +$$
$$\sum_{\substack{i\in V(p')\\ j\in V(p')\setminus\{i\}}} \frac{c_i^{\text{track}} + c_j^{\text{track}}}{2} s_{ij} + \sum_{i\in V(p')} \left[-\frac{c_i^{\text{track}}}{2} b_i - \frac{c_i^{\text{track}}}{2} e_i \right]$$

By the transformation of p' we know that the sets $S(p)$, $M(p)$, $B(p)$, $F(p)$ and $D(p)$ correspond to the edges and vertices used in p'. Using these sets and (4.9) we can simplify the formula to

$$\sum_{(i,j)\in S(p)} c_{ij}^{\text{solder}} + \sum_{(i,j)\in M(p)} c_{ij}^{\text{move}} +$$
$$\sum_{i\in B(p)} c_i^{\text{begin}} + \sum_{i\in F(p)} c_i^{\text{end}} + \sum_{i\in D(p)} c_i^{\text{dip}} + \sum_{i\in T(p)} c_i^{\text{track}} +$$
$$(|B(p)| + |F(p)| + 2|D(p)|) \cdot c^{\text{shift}},$$

which is equal to

$$PathCost(p).$$

The path obtained by solving the IP model has the four properties of the solder model. By these properties the correctness of the IP model is proven. □

4.5 Multiple Solder Unit Model

In this section we describe the multiple solder unit model. This model consists of three types of constraints. The first type of constraints are the *detour constraints* (Section 4.5.1). With these constraints we allow vertices to be passed. Because we do not want that all vertices are, passed we introduce *overlap constraint* (Section 4.5.2) to force vertices to be used. In the model we have multiple units. To obtain a solution where the throughput of boards through the machine is maximized we need *balancing constraints* (Section 4.5.3). These constraints force the cost of the bottleneck unit to be minimized.

4.5.1 Detour Constraints

The model has multiple units and therefore also multiple actions soldering the same pins. Since these actions are represented by different vertices, only one of them has to be used and

the others can be passed. In the previous model we used the classical TSP approach where every vertex is visited. For the new model we have to adapt our set of constraints to allow vertices to be passed.

To be able to pass super vertices we have two options; a super vertex has degree 0 or 2. Degree 2 is needed to visit a super vertex and degree 0 is needed to pass a super vertex. We transform the constraints of the previous model by forcing the in-degree and out-degree to be equal. With another constraint we force the in-degree to be at most 1. This results in the two possible degrees.

$$\sum_{j \in V \setminus \{i\}} [m_{ij} + s_{ij}] = \sum_{j \in V \setminus \{i\}} [m_{ji} + s_{ji}] \qquad \text{for } i \in V \tag{4.10}$$

$$\sum_{j \in V \setminus \{i\}} [m_{ji} + s_{ji}] \leqslant 1 \qquad \text{for } i \in V \tag{4.11}$$

For the internal solder vertex we use the same approach to make sure that it has degree two or zero.

$$b_i + \sum_{j \in V \setminus \{i\}} s_{ji} = e_i + \sum_{j \in V \setminus \{i\}} s_{ij} \qquad \text{for } i \in V \tag{4.12}$$

Lemma 4. *The internal solder vertex has at most one incoming edge by constraints* (4.1), (4.10), (4.11) *and* (4.12).

Proof. We have to prove for an arbitrary super vertex i that constraints (4.1), (4.10) and (4.11) force that

$$b_i + \sum_{j \in V \setminus \{i\}} s_{ji} \leqslant 1.$$

By (4.10) and (4.11) the number of solder edges connected to a super vertex is at most two. There are three ways of connecting them to the solder vertex of i.

(a) *two solder edges are used*
By using two solder edges the constraint (4.12) implies that $b_i = e_i$. According to (4.1), b_i and e_i cannot be used both. This results in $b_i + \sum_{j \in V \setminus \{i\}} s_{ji} = 1$.

(b) *one solder edge is used*
In this case we must use b_i or e_i. By (4.12) we must have an incoming solder edge if e_i is used or an outgoing solder edge if b_i is used. In both cases $b_i + \sum_{j \in V \setminus \{i\}} s_{ji} = 1$.

(c) *no solder edges are used*
This case is equal to (a) because (4.12) implies that $b_i = e_i$ and according to (4.1), b_i and e_i cannot be used both. This results in $b_i + \sum_{j \in V \setminus \{i\}} s_{ji} = 0$.

In all these cases it follows that the internal solder vertex has at most one incoming edge. □

By Theorem 4, no further constraints are needed to force the solder vertex to have at most one incoming edge.

We know from (4.10) and (4.11) that the internal move vertex has a maximum degree of 2. If the maximum degree is used then we want d_i to be used.

$$\sum_{j \in V \setminus \{i\}} [m_{ij} + m_{ji}] \leqslant 1 + d_i \qquad \text{for } i \in V \tag{4.13}$$

Lemma 5. *Let i be a super vertex. Assuming that d_i occurs with a positive coefficient in the objective function, it holds that, the edge d_i is used iff there are two movement edges adjacent to i.*

Proof. For all cases where the number of movement edges is at most 1, the edge d_i is not used due to (4.13) and due to the assumption that d_i is minimized by the objective function. The case where there are two movement edges d_i is used to satisfy (4.13). □

Lemma 6. *Given constraints* (4.1), (4.10)–(4.13)*, the internal edges of a super vertex i are uniquely determined by the use of movement and solder edges incident to super vertex i.*

Proof. Constraint (4.11) forces that there are two main cases; a super vertex i is visited or not. The incoming edges determine the visiting condition.

(i) *one incoming edge*
Constraint (4.10) implies that the super vertex must have as many incoming as outgoing edges, which results in the following four cases.

  (a) *two solder edges* (Figure 4.4a)
  By (4.12) it holds that $b_i = e_i$, which by (4.1) implies that both edges are not used. Equally d_i is not used according to Lemma 5.

  (b) *two movement edges* (Figure 4.4b)
  According to Lemma 5, d_i is used, and due to (4.1), b_i and e_i are not used.

  (c) *one incoming movement edge and one outgoing solder edge* (Figure 4.4c)
  By (4.12) and the presence of an outgoing solder edge, the edge b_i is forced to be used. Thus edges d_i and e_i are not used according to (4.1).

  (d) *one incoming solder edge and one outgoing movement edge* (Figure 4.4d)
  Due to (4.12), edge e_i is used; edges d_i and b_i are not used according to (4.1).

(ii) *no incoming edges*
Constraint (4.10) forces that the number of incoming edges and outgoing edges are equal. Hence, no outgoing edges are allowed. By (4.12) it holds that $b_i = e_i$. Therefore and due to (4.1) edges b_i and e_i are not used. According to Lemma 5, edge d_i is also not used.

Together, these cases prove the lemma. □

The SECs must also be adapted to skip a super vertex. We show only for the flow SECs how to adapt them. There are two types of flow constraints. Constraints that force one unit of flow to be consumed per (super) vertex (2.7) and constraints that make sure that only edges in the tour can have flow (2.8). The consume constraints must be changed to make sure that one unit is consumed if and only if a super vertex is used in the tour, otherwise no unit of flow can be consumed. We achieve this by

$$\sum_{j \in V \setminus \{i\}} y_{ji} - \sum_{j \in V \setminus \{i\}} y_{ij} = \sum_{j \in V \setminus \{i\}} [m_{ji} + s_{ji}] \qquad \text{for } i \in V. \tag{4.14}$$

Lemma 7. *Given* (2.8) *and* (4.14)*, sub-tours are eliminated and super vertices can be skipped.*

Proof. By (4.11) the number of incoming edges of a super vertex i is 0 or 1. For the case that one incoming edge is used, the flow constraint is equal to the normal flow sub-tour elimination constraint (2.7). For the other case, the right-hand side of the constraint equals 0. This ensures that no flow can be consumed at super vertex i. This proves that constraints (2.8) and (4.14) eliminate sub-tours and allow super vertices to be skipped. □

Solving the model with only the constraints given in this section makes no sense because all super vertices would be skipped. In the next section we introduce overlap constraints that force super vertices to be used.

4.5.2 Overlap Constraints

In this section we introduce overlap constraints that force pins to be soldered by one of the units. The overlap constraints force at least one of the overlapping actions to be used.

Let a *group* be a set of solder actions that solder a particular region. Let Γ be a set of all groups. A group g is called *minimal* if no other group exists that solders a subregion of g. Let $M \subseteq \Gamma$ be the set of all minimal groups. We use the set M_g to be the set of minimal groups that solder a subregion of group g. If g is minimal then $M_g = \{g\}$. The solder region of a group is determined by the solder actions inside the group. For a group $g \in \Gamma$, let the set of solder actions be $A_g \subseteq A$, where A is the set of all actions. By definition, the set A_g is non-empty. The number of point- and track-solder actions in A_g is denoted by $|A_g|$.

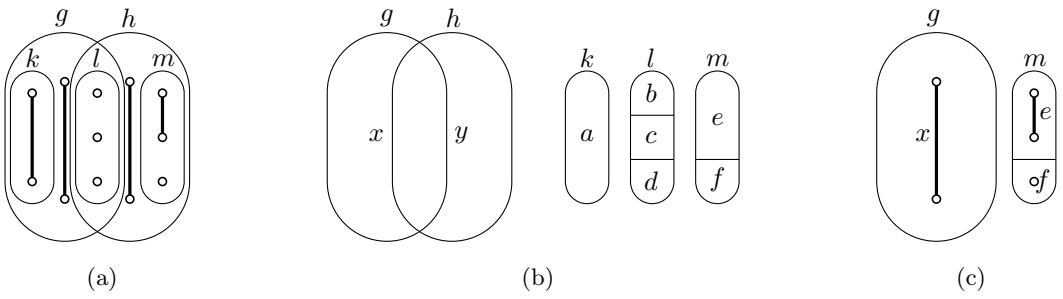


Figure 4.7: Group examples

Figure 4.7a shows an example of groups and actions that overlap each other. Two non-minimal groups g and h belong to the solder unit with the larger nozzle and k, l and m are (minimal) groups of the solder unit with the smaller nozzle. Each group consists of actions. Each action is represented by a region and labeled by a letter in $\{a, \ldots, f\}$, see Figure 4.7b. By soldering g the minimal groups k and l are soldered. Equally, by soldering h the minimal groups l and m are soldered. A possible solution for soldering all minimal groups is given in Figure 4.7c.

The definition of a minimal group is that the corresponding region is not split into subregions. Thus, if group $g \in M_g$ then M_g is a singleton. This assumption is practical because we know that a bigger nozzle can be replaced by a smaller nozzle. Unfortunately, an operator may decide that a region must be soldered by a bigger nozzle, like in the example in Figure 4.8, where the dashed region must be soldered by g. The dashed region makes g a minimal group because g *must* be soldered. The problem can be solved by removing all minimal groups in M_g except g itself, which results in $M_g = \{g\}$.

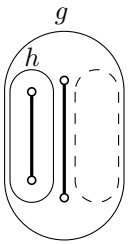


Figure 4.8: An example of group g that is also a minimal group of itself. Group g must be used to solder the dashed area.

For each group $g \in \Gamma$, we introduce a zero-one variable z_g that indicates whether group g is soldered, that is,

$$z_g = \begin{cases} 1 & \text{Region of } g \text{ is soldered by group } g \text{ or by a non-minimal group } h \text{ such that } g \in M_h \\ 0 & \text{Region of } g \text{ is not soldered} \end{cases}$$

For each minimal group g, we introduce a zero-one variable z'_g to indicate whether g is soldered by a non-minimal (bigger) group. If $z'_g = 1$, a non-minimal group solders g.

We first make sure that all minimal groups are soldered, possibly as part of non-minimal groups.

$$z_g = 1 \qquad \text{for } g \in M \tag{4.15}$$

Of course, we must ensure that the values of the variables z_g and z'_g get the correct value. We force z_g to be 1 if group g is soldered by a non-minimal group or if soldered by the actions of g itself.

$$\sum_{\substack{i \in V(a) \\ j \in V \setminus V(a)}} [m_{ji} + s_{ji}] \geqslant z_g - z'_g \qquad \text{for } g \in M \text{ and } a \in A_g \tag{4.16}$$

The constraint ensures that all actions inside a group are used if the right-hand side of the constraint equals 1. The right-hand side consists of two variables; z_g indicating that the group g is soldered and z'_g indicating that group g can be passed because of another non-minimal group that solders all actions in group g.

Combining the two constraint (4.15) and (4.16), we make sure that all minimal groups are soldered.

$$\sum_{\substack{i \in V(a) \\ j \in V \setminus V(a)}} [m_{ji} + s_{ji}] \geqslant 1 - z'_g \qquad \text{for } g \in M \text{ and } a \in A_g \tag{4.17}$$

For the non-minimal groups we use the same trick to force z_g to be 1 if group g is soldered.

$$\sum_{\substack{i \in V(a) \\ j \in V \setminus V(a)}} [m_{ji} + s_{ji}] \geqslant z_g \qquad \text{for } g \in \Gamma \setminus M \text{ and } a \in A_g \tag{4.18}$$

With the correct values of z_g we restrict the value of z'_g to have a correct value with respect to the overlapping regions.

$$z'_h \leqslant z_g \qquad \text{for } g \in \Gamma \setminus M \text{ and } h \in M_g \tag{4.19}$$

The value of z'_h is allowed to become 1 if a non-minimal group g that contains the minimal group h is soldered. This allows the IP to skip the actions of group h.

Finally, we need to make sure that if an action has an incoming edge then all the solder edges inside that action must be used.

$$(d-1) \sum_{\substack{i \in V(a) \\ j \in V \setminus V(a)}} [m_{ji} + s_{ji}] = \sum_{(i,j) \in E(a)} s_{ij} \qquad \text{for } a \in A \tag{4.20}$$

where d is the number of vertices in action a.

Lemma 8. *All soldering edges are used if one of the vertices of the action has an incoming edge.*

Proof. An action that has an incoming edge to any vertex of that action is forced by (4.20) to have $d-1$ soldering edges, where d is the number of vertices in the action. By the definition of a path we know that at least $d-1$ edges are needed to connect d vertices. According to this definition all the soldering edges are used of an action. Suppose that an action has more then one incoming edge. Then constraint (4.20) is violated since the number of edges in an action is already $d-1$. □

Lemma 9. *All pins are soldered due to constraints* (4.17)–(4.20)*.*

Proof. By the definition of a minimal group all pins are soldered if all minimal groups are soldered, that is,

$$S \subseteq \bigcup_{g \in M} \sigma(g),$$

where $\sigma(g) = \bigcup_{a \in A_g} \sigma(a)$.

Constraint (4.17) ensures that every minimal group is soldered by the actions of that group or z'_g must be one. Due to Lemma 8 we know that all actions are used if the actions have incoming edges, indicating the pins in $\sigma(g)$ are soldered. If z'_g equals one then an overlapping group h solders the same pins $\sigma(g) \subseteq \sigma(h)$. According to (4.19), z'_g may only be one iff the overlapping group is soldered. The overlapping group is a non-minimal group by definition. Using (4.18) makes sure that all actions of the non-minimal groups have incoming edges if indicated that the group is soldered. Due to Lemma 8 we know that all pins of this group are soldered.

All actions soldering the pins of the minimal groups are used and therefore included in the solution tour, that is,

$$S \subseteq \bigcup_{g \in M} \sigma(g) = \bigcup_{u \in U} \bigcup_{a \in p_u} \sigma(a) \subseteq \bigcup_{u \in U} \sigma(p_u).$$

□

4.5.3 Balance Constraints

The main objective of the Minimum-Cost Solder Paths is to optimize the throughput of boards through the machine. This problem can be reformulated by minimizing the processing time of the solder unit with the longest processing time. We call this unit the bottleneck unit.

We introduce a new variable Z that will take the cost of the largest path. By minimizing this variable the cost of the longest path is minimized.

$$\text{minimize } Z \tag{4.21}$$

To ensure that Z takes the value of the largest path we need additional constraints, namely

$$Cost(u) \leqslant Z, \qquad \text{for } u \in U \tag{4.22}$$

where $Cost(u)$ is the cost of the edges used by unit u, that is,

$$\begin{aligned} Cost(u) = &\sum_{\substack{i \in V_u \\ j \in V_u \setminus \{i\}}} \left[\left(c_{ij}^{\text{solder}} + \frac{c_i^{\text{track}} + c_j^{\text{track}}}{2} \right) s_{ij} + c_{ij}^{\text{move}} m_{ij} \right] + \\ &\sum_{i \in V_u} \left[\left(c_i^{\text{dip}} + 2 \cdot c^{\text{shift}} \right) d_i + \left(c_i^{\text{begin}} + c^{\text{shift}} - \frac{c_i^{\text{track}}}{2} \right) b_i + \right. \\ &\left. \left(c_i^{\text{end}} + c^{\text{shift}} - \frac{c_i^{\text{track}}}{2} \right) e_i \right]. \end{aligned} \tag{4.23}$$

Lemma 10. *The variable Z equals the processing time of the bottleneck unit given the objective function* (4.21) *and constraints* (4.22).

Proof. Due to (4.22), the variable Z must be at least the processing time of the unit with the largest processing time. The variable is minimized by the objective function (4.21) and therefore Z takes the value of the bottleneck unit, that is,

$$Z = \max_{u \in U} Cost(u).$$

□

Lemma 11. *The function $Cost(u)$ is equivalent to $PathCost(p_u)$.*

Proof. Let $B(p_u)$, $F(p_u)$ and $D(p_u)$ be super vertices in V_u where $b_i = 1$, $e_i = 1$ and $d_i = 1$, respectively. The set $T(p_u)$ consists of all vertices with $b_i = e_i = d_i = 0$ and two soldering edges. Let $S(p_u) = \{(i,j) \mid (i,j) \in E(p_u) \text{ and } s_{ij} = 1\}$ and $M(p_u) = \{(i,j) \mid (i,j) \in E(p_u) \text{ and } m_{ij} = 1\}$.

In the same way as in the proof of Lemma 2 we can prove that

$$\sum_{i \in T(p_u)} c_i^{\text{track}} = \sum_{\substack{i \in V(p_u) \\ j \in V(p_u) \setminus \{i\}}} \frac{c_i^{\text{track}} + c_j^{\text{track}}}{2} s_{ij} + \sum_{i \in V(p_u)} \left[-\frac{c_i^{\text{track}}}{2} b_i - \frac{c_i^{\text{track}}}{2} e_i \right]. \tag{4.24}$$

We rewrite the function $Cost(u)$ to

$$\begin{aligned}
&\sum_{\substack{i \in V(p_u) \\ j \in V(p_u) \setminus \{i\}}} c_{ij}^{\text{solder}} s_{ij} + \sum_{\substack{i \in V(p_u) \\ j \in V(p_u) \setminus \{i\}}} c_{ij}^{\text{move}} m_{ij} + \\
&\sum_{i \in V(p_u)} \left[c_i^{\text{begin}} b_i + c_i^{\text{end}} e_i + c_i^{\text{dip}} d_i \right] + \\
&\sum_{i \in V(p_u)} \left[c^{\text{shift}} b_i + c^{\text{shift}} e_i + 2 \cdot c^{\text{shift}} d_i \right] + \\
&\sum_{\substack{i \in V(p_u) \\ j \in V(p_u) \setminus \{i\}}} \frac{c_i^{\text{track}} + c_j^{\text{track}}}{2} s_{ij} + \sum_{i \in V(p_u)} \left[-\frac{c_i^{\text{track}}}{2} b_i - \frac{c_i^{\text{track}}}{2} e_i \right].
\end{aligned}$$

By the definitions of the sets $S(p_u)$, $M(p_u)$, $B(p_u)$, $F(p_u)$, $T(p_u)$ and $D(p_u)$ and using (4.24), we can simplify the formula to

$$\begin{aligned}
&\sum_{(i,j) \in S(p_u)} c_{ij}^{\text{solder}} + \sum_{(i,j) \in M(p_u)} c_{ij}^{\text{move}} + \\
&\sum_{i \in B(p_u)} c_i^{\text{begin}} + \sum_{i \in F(p_u)} c_i^{\text{end}} + \sum_{i \in D(p_u)} c_i^{\text{dip}} + \sum_{i \in T(p_u)} c_i^{\text{track}} + \\
&(|B(p_u)| + |F(p_u)| + 2|D(p_u)|) \cdot c^{\text{shift}},
\end{aligned}$$

which is equal to

$$PathCost(p_u).$$

This proves that $Cost(u) = PathCost(p_u)$. □

4.5.4 The Complete Model

In this section we present the complete model for solving the Minimum-Cost Solder Paths Problem. We analyze the numbers of constraints and variables.

Objective function

$$\text{minimize } Z$$

Balance constraints

$$Cost(u) \leqslant Z \qquad \text{for } u \in U$$

where $Cost(u)$ is given by

$$\begin{aligned} Cost(u) = \sum_{\substack{i \in V_u \\ j \in V_u \setminus \{i\}}} \left[\left(c_{ij}^{\text{solder}} + \frac{c_i^{\text{track}} + c_j^{\text{track}}}{2} \right) s_{ij} + c_{ij}^{\text{move}} m_{ij} \right] + \\ \sum_{i \in V_u} \left[\left(c_i^{\text{dip}} + 2 \cdot c^{\text{shift}} \right) d_i + \left(c_i^{\text{begin}} + c^{\text{shift}} - \frac{c_i^{\text{track}}}{2} \right) b_i + \right. \\ \left. \left(c_i^{\text{end}} + c^{\text{shift}} - \frac{c_i^{\text{track}}}{2} \right) e_i \right]. \end{aligned}$$

Expanded vertex model constraints

$$d_i + b_i + e_i \leqslant 1 \qquad \text{for } i \in V$$

$$\sum_{j \in V \setminus \{i\}} [m_{ij} + s_{ij}] = \sum_{j \in V \setminus \{i\}} [m_{ji} + s_{ji}] \qquad \text{for } i \in V$$

$$\sum_{j \in V \setminus \{i\}} [m_{ji} + s_{ji}] \leqslant 1 \qquad \text{for } i \in V$$

$$b_i + \sum_{j \in V \setminus \{i\}} s_{ji} = e_i + \sum_{j \in V \setminus \{i\}} s_{ij} \qquad \text{for } i \in V$$

$$\sum_{j \in V \setminus \{i\}} [m_{ij} + m_{ji}] \leqslant 1 + d_i \qquad \text{for } i \in V$$

Subtour elimination constraints

$$\sum_{j \in V \setminus \{i\}} y_{ji} - \sum_{j \in V \setminus \{i\}} y_{ij} = \sum_{j \in V \setminus \{i\}} [m_{ji} + s_{ji}] \qquad \text{for } i \in V$$

$$0 \leqslant y_{ij} \leqslant (n-1)[m_{ij} + s_{ij}] \qquad \text{for } (i,j) \in V^2 \text{ with } j \notin \{1, i\}$$

Overlap constraints

$$\sum_{\substack{i\in V(a)\\ j\in V\setminus V(a)}} [m_{ji} + s_{ji}] \geqslant 1 - z'_g \qquad \text{for } g \in M \text{ and } a \in A_g$$

$$\sum_{\substack{i\in V(a)\\ j\in V\setminus V(a)}} [m_{ji} + s_{ji}] \geqslant z_g \qquad \text{for } g \in \Gamma \setminus M \text{ and } a \in A_g$$

$$z'_h \leqslant z_g \qquad \text{for } g \in \Gamma \setminus M \text{ and } h \in M_g$$

$$(d-1)\sum_{\substack{i\in V(a)\\ j\in V\setminus V(a)}} [m_{ji} + s_{ji}] = \sum_{(i,j)\in E(a)} s_{ij} \qquad \text{for } a \in A$$

Analysis

Table 4.1 summarizes the number of constraints and variables used in the model. The total number of constraints is $O(|U| + |V|^2 + 2^{|\Gamma|})$ and the total number of variables is $O(|V|^2)$.

	# constraints	**# variables**
Graph	-	$O(\|V\|^2)$
Detour constraints	$O(\|V\|)$	$O(\|V\|)$
Subtour elimination constraints	$O(\|V\|^2)$	$O(\|V\|^2)$
Overlap constraints	$O(2^{\|\Gamma\|} + \|A\|) = O(2^{\|\Gamma\|} + \|V\|)$	$O(\|\Gamma\|) = O(\|V\|)$
Balancing constraints	$O(\|U\|)$	$O(1)$
Total	$O(\|U\| + \|V\|^2 + 2^{\|\Gamma\|})$	$O(\|V\|^2\|)$

Table 4.1: Number of constraint and variables.

Correctness

Theorem 5. *Any solution of the model described by objective function and constraints given in this section is a solution of the* Minimum-Cost Solder Problem *with multiple units.*

Proof. "⇒" As in Theorem 4, we first prove that multiple paths in the solder model can be transformed into a correct tour in the IP model with the same cost.

The solder model has m paths. We can construct as in Theorem 4 a single tour in our IP model that represents these paths by adding stop vertices per path and connecting the stop vertices with the first super vertex of a path of the next path. The last super vertex of the last unit is connected to the first super vertex of the first path. We use the same type of edges between the vertices as in the solder model. We have a correctly transformed the m paths to a possible solution of the IP model. We show that all constraints are satisfied using this transformation, and we show that the processing time of the bottleneck unit in the IP model is equal to the processing time of the bottleneck unit in solder model.

- *Detour constraints*
  In the IP model it is possible that not all super vertices are used due to the detour constraints. Lemma 6 shows that the edges d_i, b_i and e_i are used according the edge types connected to the super vertex i. The solution of the IP model by construction corresponds to a single tour and therefore all the detour constraints are satisfied.

- *overlap constraints*
  By definition, all pins are soldered in the solder model and the transformation of the solder model uses the same point- and track-solder actions. Due to these actions all the minimal groups are soldered and the overlap constraints are satisfied.

- *balancing constraints*
  Due to Lemmas 10 and 11 the processing time of the bottleneck unit of the solder model is equal to the processing time of the bottleneck unit in the IP model, that is,

$$\max_{u \in U} PathCost(p_u) = \max_{u \in U} Cost(u) = Z.$$

"$\Leftarrow$" By Lemma 7 and the overlapping constraint we obtain a single tour t of super vertices. The overlapping constraints force super vertices to be used otherwise the tour would consist of zero super vertices. By removing the m stop vertices we obtain m paths p'_1, p'_2, $\ldots$, p'_m. After transforming p'_u to a path p_u in the solder model for each $u \in U$ we must show that these paths form an optimal solution of the Minimum-Cost Solder Path Problem.

The transformation is done as follows: per unit u we use the super vertices in p'_u as the vertices of p_u. The sequence of vertices in p_u is defined by the soldering and movement edges in p'_u. Let $S(p_u) = \{(i,j) \mid (i,j) \in E(p'_u) \text{ and } s_{ij} = 1\}$ and let $M(p_u) = \{(i,j) \mid (i,j) \in E(p'_u) \text{ and } m_{ij} = 1\}$. A path p'_u consists of four types of super vertices representing the different sets. Let $B(p_u)$, $F(p_u)$, $D(p_u)$ and $T(p_u)$ be the sets of super vertices where $b_i = 1$, $e_i = 1$, $d_i = 1$ and $b_i = e_i = d_i = 0$ with two soldering edges, respectively.

To show that these paths are correct solder paths in the solder model and that the IP model produces an optimal solution, we need to prove the following properties of the path p_u for each unit $u \in U$:

(i) *Path p_u starts with the starting vertex of unit u.*
  By construction of the graph of the IP model the stop vertices are connected to the start vertex of the previous unit. Removing the stop vertices and their adjacent edges makes the start vertex of a unit u the start vertex of p_u.

(ii) *Keep-out areas are not soldered.*
  We assume that the solder actions produced in Chapter 3 do not overlap keep-out areas and we know that the solder edges in the graph avoid keep-out areas, hence we know that no point of $\sigma(p)$ overlaps a keep-out area.

(iii) *The support bar is avoided.*
  By construction of the model graph, the solder edges do not cross the support bar area and the movement edges have an extra cost to represent the movement of the nozzle at a larger distance to the board. Therefore we know that the support bar is avoided.

(iv) *Every pin is soldered.*
Due to Lemma 9 each pin is soldered by some action in one of the paths $p_1, p_2, \ldots, p_m$.

(v) *The throughput of boards is maximized.*
There are five types of vertices present in the IP model. The four types given in Figure 4.4 and the fifth type is a super vertex that is not visited. The sets $B(p_u)$, $F(p_u)$, $D(p_u)$ and $T(p_u)$ represent the four types in Figure 4.4. The vertices of the fifth type use no edges and are therefore not present in the objective function. According to Lemmas 10 and 11, it holds that

$$Z = \max_{u \in U} Cost(u) = \max_{u \in U} PathCost(p_u),$$

and therefore, the cost of the longest path is minimized. This is equivalent to maximizing the throughput of boards through the machine.

All these properties are satisfied by the IP model, which therefore yields a correct optimal solution of the Minimum-Cost Solder Path Problem. □

4.5.5 Additional Soldering Constraints

The model can be extended such that a path is forced to have specific properties.

In practice it is sometimes necessary to restrict a soldering action to be used in one direction. This often occurs by soldering connectors that have plastic pins for extra strength or components that use extra pads to avoid a short circuit caused by soldering. In the original model we always added both directed soldering edges to make sure that both ways can be in the solution. To restrict the model to use only one direction, we can remove one of the directed soldering edge from the graph.

Many products have different release versions. In the different releases minor changes are made. Often some components are removed, added or circuits are changed. The operator uses then the knowledge of the previous release to create new solder paths. Maybe a part of the soldering path must be in the new version as well. We can force a complete sub-path into the solution by adding extra constraints that force every edge of the sub-path to be in the solution.

4.6 Improvements

In this section we improve the IP model explained in the previous sections. We show two types of improvements; aesthetically pleasing and speed improvements. As aesthetic improvement we minimize not only the path of the bottleneck unit but also the paths of the other units. This results in a more aesthetically pleasing result (Section 4.6.1). For increasing the solving speed of the model we show a number of techniques; strengthening the flow constraints (Section 4.6.2), using simple cutting planes (Section 4.6.3) and reducing the number of edges in the model (Section 4.6.4).

4.6.1 Optimizing Smallest Path

The model minimizes the path used by the bottleneck unit. Since only that path is minimized other paths can have crosses and detours. It may not look like an optimal solution but these crosses and detours do not have any effect on the throughput of boards through the machine. To ensure that the result has no detours and crosses, we change the constraints (4.22) by adding a factor of the processing times of the other units. The original problem is still optimized because the bottleneck unit has the biggest factor.

$$Cost(u) + f \cdot \sum_{u' \in U \setminus \{u\}} Cost(u') \leqslant Z \qquad \text{for } u \in U, \tag{4.25}$$

where f is a small percentage, such as 10% or 1%.

The solving speed is decreased when using (4.25) instead of (4.22) but the solutions tend to become more aesthetically pleasing.

4.6.2 Strengthening Flow SECs

We can strengthen the coupling constraints [GP02, page 19] of the flow formulation. We use the knowledge of the graph constructed in Section 4.3.

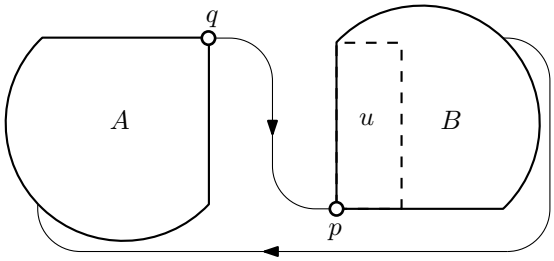


Figure 4.9: Tightening the flow coupling constraints at the edges between two units.

The flow through the edges connecting the units can be tightened by using the number of vertices used in the next units. Let p be the start vertex of unit u and let q be the end vertex of the unit before unit u. We have a set A of all vertices before unit u and the set B of all the vertices of unit u and after u, see Figure 4.9. We can use the following constraint to tighten the flow of the edge (q, p).

$$y_{qp} = \sum_{\substack{i \in B \\ j \in B \setminus \{i\}}} [m_{ji} + s_{ji}] \qquad \text{for } u \in U, \tag{4.26}$$

Observe that the following flow coupling constraints hold.

$$y_{n1} = 0 \tag{4.27}$$

$$y_{kn} = x_{kn} \qquad \text{for } k \in V_m \tag{4.28}$$

4.6.3 Simple Cutting Planes

Cutting planes are used to make the feasible IP solution space smaller and therefore in many cases the IP is solved faster. CPLEX uses already some heuristics to find cutting planes and is therefore faster than other IP solvers that do not use cutting planes such as lp_solve (version 5.5).

The IP model has some properties that can be used for cutting planes. We use a simple cutting plane that takes the number of starting and stopping with soldering edges into account. It is trivial that every solder action must start and stop with soldering.

Theorem 6. *The number of edges starting with soldering edges is equal to the number of edges stopping with soldering, that is,*

$$\sum_{i \in V} b_i = \sum_{i \in V} e_i.$$

Proof. By Lemma 7, we know that the solution is a tour of super vertices and due to the graph construction, the tour starts and ends with a movement edge. Switching from moving to soldering or visa versa, edges b_i and e_i are used. By using a soldering edge or multiple soldering edges the tour switches once from moving to soldering and once from soldering to moving. This occurs multiple times in the tour and resulting in an equal number of starting and stopping with soldering edges. □

We constructed a cutting plane by looking at two super vertices and the edges between them. The solution is a tour of super vertices and therefore one edge is allowed between two super vertices, except for a graph with only two vertices. By the construction of the graph we can assume that we always have a graph with more than two vertices. To tighten the feasible solution space we can add these constraints. Note that the use of more then two edges between two super vertices is already not allowed by the SECs.

Theorem 7. *Only one edge between two super vertices is used, that is,*

$$m_{ij} + s_{ij} + m_{ji} + s_{ji} \leqslant 1.$$

Proof. A solution of the IP model is a tour that has more than two super vertices. By definition only one direction of a soldering edge or movement edge is used between two super vertices. Using a movement edge the two super vertices are visited. By the definition of a tour, super vertices are not allowed to occur multiple times. Therefore the soldering edge between them cannot be in the solution. With the same reasoning we know that a movement edge is not in the tour if a solder edge is used. □

4.6.4 Reducing the Model Size

The solving speed of the model depends on the number of variables and constraints. In this section we reduce the number of edges in the constructed graph that correspond to the number of variables in the model.

We can reduce the model by allowing only one edge type between two vertices. This process is not that trivial because we do not know which edge type is chosen.

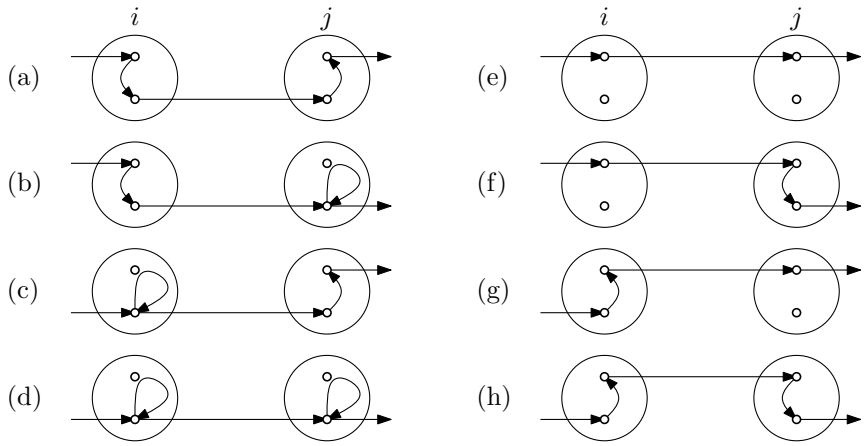


Figure 4.10: Eight ways of connecting two vertices with different edges.

In Figure 4.10, eight ways of connecting two vertices with different edges are shown. These eight instances have different costs depending on the used edge types. Instances (a)–(d) use a movement edge between i and j. The other instances (e)–(h) use a solder edge between i and j. The choice between a movement and solder edge depend a lot on the edge types used to enter i and leave j. We can compute the minimal and maximal values per edge type. With these values we can decide which edge type alway is chosen. Let $c_{\max}^{\text{solder}}$, $c_{\min}^{\text{solder}}$, $c_{\max}^{\text{move}}$ and $c_{\min}^{\text{move}}$ be the maximal and minimal costs of using a solder or movement edge, respectively. With these values we decide to use a movement edge if $c_{\max}^{\text{move}} < c_{\min}^{\text{solder}}$ and a solder edge if $c_{\max}^{\text{solder}} < c_{\min}^{\text{move}}$, otherwise we add them both.

This reduction only decreases the number of edges for instances where the solder actions are far apart. In Chapter 5 we show the difference between the reduced model and the normal model.

Chapter 5

Experiments

In this chapter we show experiments we have done with some products. We experimented on three products; 6001-0710-1001, 6159-0500-1501 and 6180-0900-1000. These three product are chosen because they represent the average, with respect to number of pins, of the products produced by Prodrive. The solder actions used are given in Appendix A. We assume that these solder actions correctly solder all pins.

We used a state-of-the-art IP solver ILOG CPLEX version 11.010 on a 64-bit Quad-Core 2.40Ghz processor with 4GB memory. This server was used by multiple users and therefore not all CPU power and memory resources were used during the experiments. The CPLEX version was restricted to a single thread.

The IP model described in Chapter 4.5 is implemented with different overlap constraints. The implemented overlap constraints exclude some possible solutions from the solution space. This restriction applies only to instances that have at least two non-minimal groups that solder the same minimal group. In our experiments the product 6180-0900-1000 has this kind of overlap. So, there may be a better solution for this product than the one presented here. See Appendix B for the used overlap constraints.

In our experiments we used the improvements *optimizing smallest path* (with $f = 0.1$) and the *simple cutting planes*. Per product instance, we compare the results obtained from the normal model graph and the reduced model graph.

5.1 6001-0710-1001

This product consists of 91 pins and 543 keep-out areas. The paths defined by an operator are given in Figure 5.1a. They used the nozzle of sizes 6 mm and 10 mm. The solution obtained by the IP model is shown in Figure 5.1b. The number of solder actions is 37 (unit 1) and 14 (unit 2), see Appendix A.1.

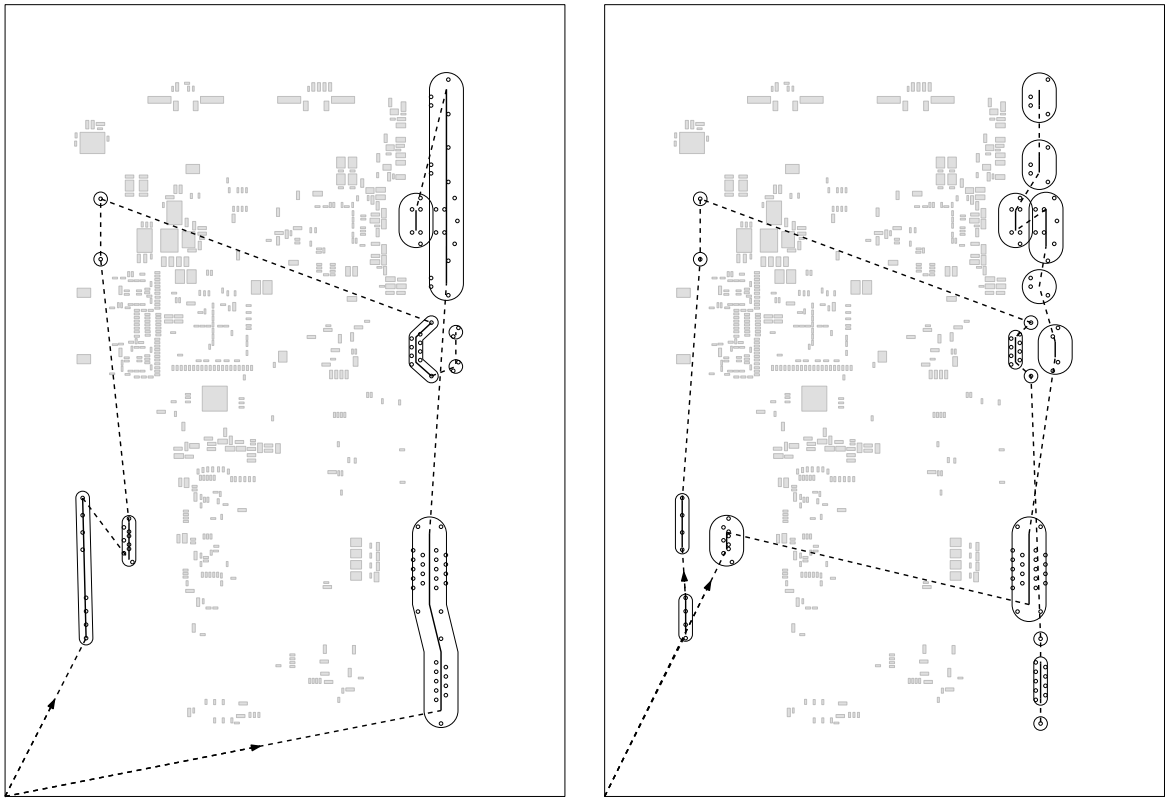
In Table 5.1 the results are given for the same product. The processing time of the bottleneck unit (set in italics) is decreased by 5.46 seconds (19.69%).

It took 1.5 hours to solve the reduced model and 0.5 hours for the normal model. These

Type	*Operator*		*IP solution*	
	unit 1	unit 2	unit 1	unit 2
Soldering time	21.41	26.12	19.25	21.01
Moving time	1.95	1.61	2.68	1.26
Total time	23.36	*27.73*	21.93	*22.27*

Table 5.1: Experimental results concerning product 6001-0710-1001.

solving times are counter intuitive. Possible explanation for this behavior is that CPLEX used different cutting planes for both instances. The cutting planes in CPLEX are found by heuristic algorithms and due to the different models it is possible that the heuristics find better or worse cutting planes.



(a) (b)

Figure 5.1: Product 6001-0710-1001; (a) the paths constructed by an operator and (b) the paths computed using the IP model.

5.2 6159-0500-1501

This product consists of 321 pins and 9 keep-out areas. The paths defined by an operator are given in Figure 5.2a. They used a nozzle of size 10 mm. The solution obtained by the

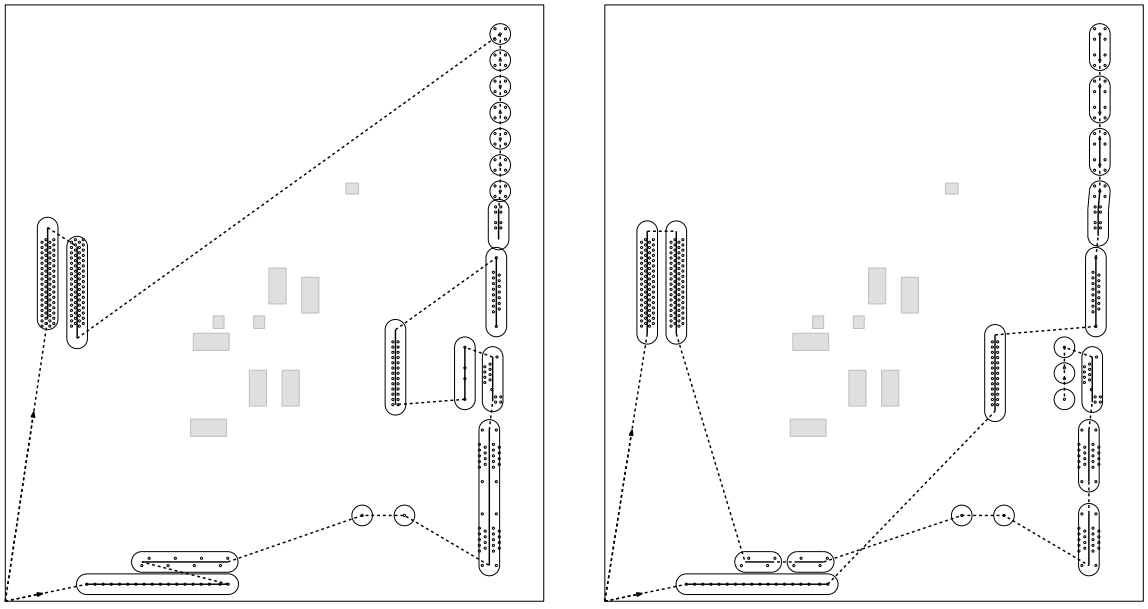
IP model is shown in Figure 5.2b. The number of solder actions for both units is 34, see Appendix A.2.

Type	*Operator*		*IP solution*	
	unit 1	unit 2	unit 1	unit 2
Soldering time	64.67	65.79	54.58	61.98
Moving time	3.29	2.35	3.21	1.70
Total time	67.96	*68.04*	57.79	*63.68*

Table 5.2: Experimental results concerning product 6159-0500-1501.

Table 5.2 shows the result; the processing time of the bottleneck unit is decreased by 4.46 seconds (5.31%).

It took 4 hours to solve the reduced model and 5.25 hours for the normal model. For this model we used about the same solder times as used by the operators. Some of the solder times where unknown due to the use of different actions. The solution of the model has some unwanted patterns. Let's look at the track-solder actions on the upper right part of the board. These track-solder actions are constructed by combining two point-solder actions. The point-solder actions are equally distributed along a straight line, meaning that the distance between two adjacent point-solder actions is the same. We expected that the same edge type is used between the actions because the distances are the same. But this is not the case. After investigating the instance we saw that the costs of the edges of type b_i and e_i were too cheap, which makes it attractive to use these edges instead of a point-solder action or to create one big track-solder action. Thus, wrong solder times result in unwanted patterns.



(a) (b)

Figure 5.2: Product 6159-0500-1501; (a) the paths constructed by an operator and (b) the paths computed by the IP model.

Type	*Operator*		*IP solution*	
	unit 1	unit 2	unit 1	unit 2
Soldering time	113.45	82.97	106.64	107.51
Moving time	3.95	6.37	4.47	3.66
Total time	*117.40*	89.34	111.11	*111.17*

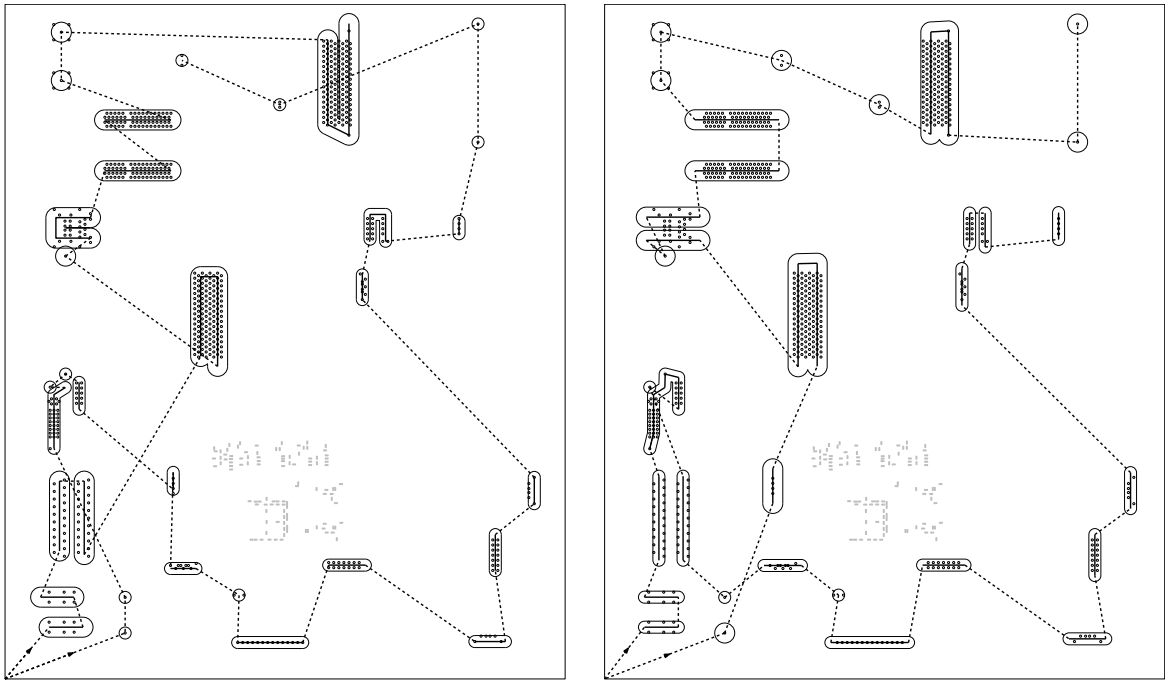
Table 5.3: Experimental results concerning product 6180-0900-1000.

5.3 6180-0900-1000

This product consists of 664 pins and 109 keep-out areas. The current paths defined by an operator are given in Figure 5.3a. They used the nozzle of sizes 6 mm and 10 mm. The solution obtained by the IP model is shown in Figure 5.3b. The number of solder actions is 52 (unit 1) and 37 (unit 2), see Appendix A.3. A nice property of the input is that some pins can only be soldered by the first unit.

In Table 5.3 the results are given for product 6180-0900-1000. The processing time of the bottleneck unit is decreased by 6.23 seconds (5.31%).

It took 2 days to solve the reduced model and 2.3 days for the normal model. As mentioned, we used overlap constraints described in Appendix A and therefore we are not certain that the result shown in Figure 5.3b is indeed an optimal solution.



(a) (b)

Figure 5.3: Product 6180-0900-1000; (a) the paths constructed by an operator and (b) the paths computed by the IP model.

Chapter 6

Conclusion

In this thesis we have investigated ways to automate the process of defining solder paths for the selected wave soldering machine. We divided the problem into two subproblems; obtaining solder actions and finding an optimal solder path containing solder actions.

We have defined the problem of obtaining solder actions given a set of pins. By using a disk cover algorithm we obtained a possible set of solder actions. Unfortunately, these solder actions have unwanted properties. These properties decrease the solder quality. Finding good solder actions is still an open problem, see future research below.

We have given a formal definition for the problem of finding solder paths. We have created two IP models; a single unit model and a multiple solder unit model. We proved that the optimal solution of these models represent an optimal solution for the original problem with respect to the given solder actions.

We experimented on practical problem instances and observed that solving the IP model is a time consuming process. We observed that using correct solder times is very important to obtain a correct solution. For all problem instances the IP model produced better solder paths with respect to the total processing time than the paths defined by the operators. The processing time of the bottleneck units where improved by 5–20%.

Future Research

- *Solder actions*
  We did not find a good algorithm for obtaining solder actions, see Chapter 3. We showed that the disk cover algorithm produced unwanted zig-zag patterns. Are there other approaches to obtaining good solder actions? Maybe a sweep-line algorithm?

- *Flux path*
  We presented in this thesis algorithms for finding solder actions and solder paths but not for finding a flux path. We have an idea to tackle this problem:

  First all solder actions used in the solder paths are ordered. The order of the solder actions is obtained by finding a sequence that minimizes the travel distance between all solder actions. The travel distance between two solder actions a_1 and a_2 is defined as

the minimum Euclidean distance between any end or corner point of a_1 and a_2.

Inside every solder action multiple pins are soldered. A path is constructed that contains all the pins inside a solder action with the minimal travel distance between the pins. For every solder action such a path is created. The start and end position of the path is extracted from the end or corner points used for the minimal distance to obtain the sequence.

In the final step the paths are connected to other paths in the same sequence as the solder actions. This approach is a simple way of obtaining a flux path but how good is it? Are there other (better) ways for obtaining a flux path?

- *Solving speed*
  The disadvantage of our approach for finding solder paths is that it is time consuming. To increase the solving speed, more advanced techniques can be used such as the algorithms used in Concorde.

  Besides the integer-programming approach also heuristics can be used to find a reasonable good solution, like a petal heuristic [RBL96] or the Held-Karp heuristic [HK70, HK71].

- *enhance solder model*
  Our solution makes a number of assumptions, such as the solder area is has a disk shape of one diameter, use a predefined nozzle per solder unit, etc. Most of these assumptions are used to simplify the problem. By simplifying the model possible better solutions can be overlooked. More research is needed to enhance the solder model by dropping some of the assumptions.

Chapter 7

Acknowledgments

I would like express my gratitude to Alexander Wolff for supporting and supervising my graduation project. He provided me with nice ideas and inspirited me in many ways. I would like to thank Cor Hurkens, for the interesting discussions about the problem and for motivating me to look in other directions in order to solve the problem, and Mark de Berg for his support and for giving me the possibility to graduate within the Algorithms Group at TU/e.

I partly worked on my thesis at Prodrive and therefore I would like to thank the directors and the employees of Prodrive for making it possible to graduate within Prodrive, providing me with necessary information and interesting discussions on how to solve the problem. Special thanks to Roy Reuser, for supervising my work on behalf of the company.

My thanks goes out to everyone supporting me with my graduation project, in particular my family, colleagues at TU/e and the people of the Algorithms Group.

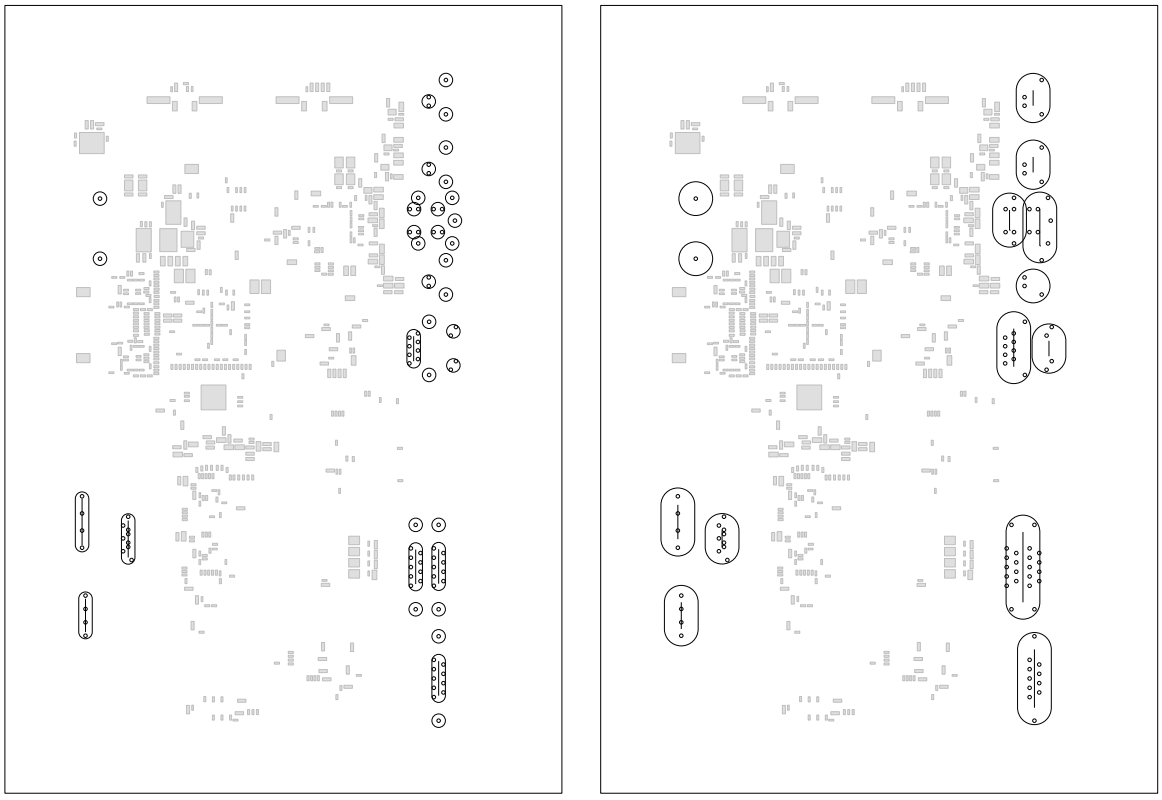
Thanks to you all!

Appendix A

Input Experiments

In this appendix we show the solder actions used for computing the experiments.

A.1 6001-0710-1001



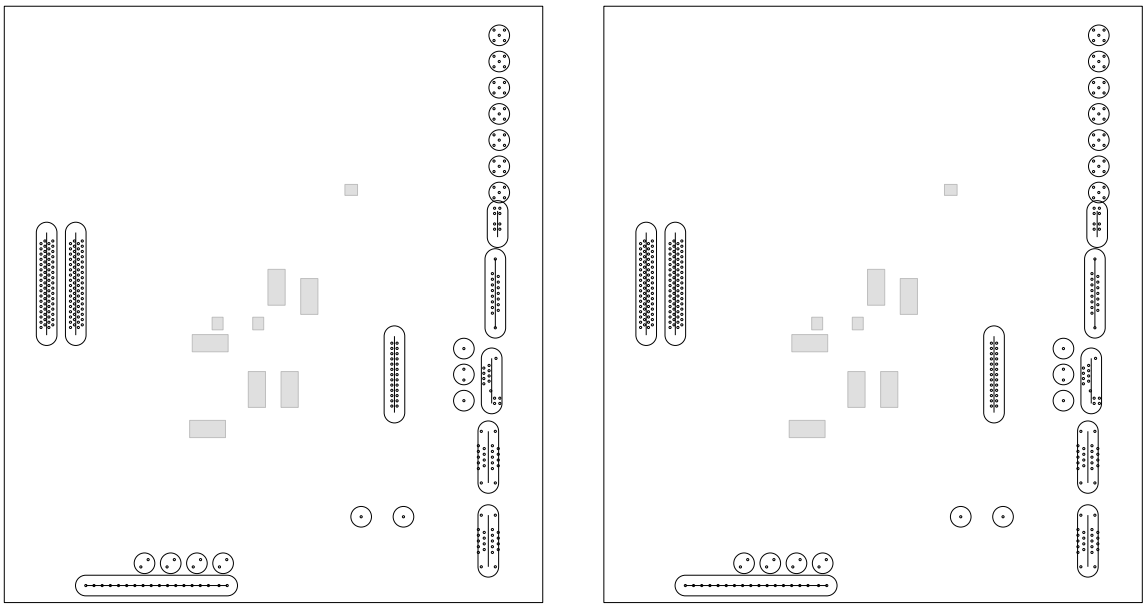
(a) (b)

Figure A.1: The solder actions of 6001-0710-1001 used as input of the algorithm to obtain solder paths.

Info	Value	
Solder speed	5	mm/s
Move speed (x, y)	100	mm/s
Move speed (z)	50	mm/s
z-position (soldering)	3	mm
z-position (movement)	10	mm

Table A.1: Input information

A.2 6159-0500-1501



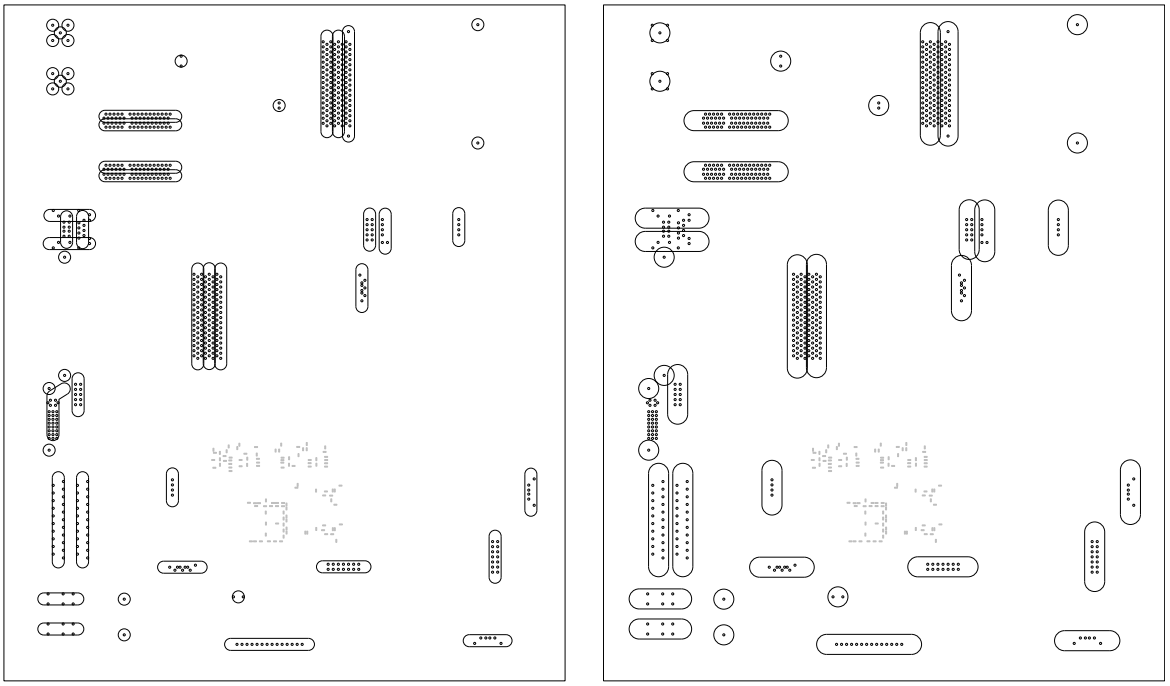
(a) (b)

Figure A.2: The solder actions of 6159-0500-1501 used as input of the algorithm to obtain solder paths.

Info	Value	
Solder speed	5	mm/s
Move speed (x, y)	150	mm/s
Move speed (z)	50	mm/s
z-position (soldering)	3	mm
z-position (movement)	10	mm

Table A.2: Input information

A.3 6180-0900-1000



(a) (b)

Figure A.3: The solder actions of 6180-0900-1000 used as input of the algorithm to obtain solder paths.

Info	Value	
Solder speed	5	mm/s
Move speed (x, y)	150	mm/s
Move speed (z)	50	mm/s
z-position (soldering)	3	mm
z-position (movement)	10	mm

Table A.3: Input information

Appendix B

Alternative Overlap Constraints

In the experiments we used alternative overlap constraints due to the difficulty of implementing the constraints presented in Section 4.5.2.

Before we present the constraints we need to define the set G_g. Let g be a group and let G_g be the set of all groups that overlap g. The set G_g does not only contain minimal groups but also non-minimal groups and g itself.

Constraints

The alternative overlap constraints consist of three types.

- The first type of constraints ensures that a group is used totally or not. By totally, we mean that all actions of that group are used in the solution. If we do not use a group then none of its actions are used in the solution. We achieve this by

$$\sum_{\substack{i \in a \\ j \in V \setminus a}} [m_{ji} + s_{ji}] = \sum_{\substack{i \in b \\ j \in V \setminus b}} [m_{ji} + s_{ji}], \qquad \text{for } b \in A_g \text{ and } g \in \Gamma \tag{B.1}$$

  where a is an arbitrary action of g.

- It is not enough to use only constraint (B.1) because an action may contain solder edges. These edges must also be used in the solution. We ensure this by the same constraint as (4.20) of Section 4.5.2.

- Next, we ensure that one of the groups that overlap each other is used.

$$\sum_{g' \in G_g} \sum_{\substack{i \in a \\ j \in V \setminus a}} [m_{ji} + s_{ji}] = 1, \qquad \text{for } g \in M \tag{B.2}$$

  where a is an arbitrary action of g'.

Difference

The difference between these constraints and the constraints given in Section 4.5.2 is that the set of optimal overlap solutions is different. Let τ^* be the set of all optimal overlap solutions and let τ be the overlap solutions allowed by the constraints in this appendix.

We have the case where the overlapping actions nicely coincide. This results in no difference between the two kind of overlap constraints, see Figure B.1. Both solution sets are the same, that is, $\tau = \tau^*$.

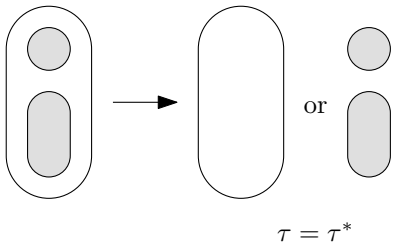


Figure B.1: Solder actions that perfectly coincide each other.

The case where at least two non-minimal groups solder the same minimal group the possible overlap solutions are different, see Figure B.2. We can see that the solutions of the constraints presented here result in a subset of all possible optimal overlap solutions.

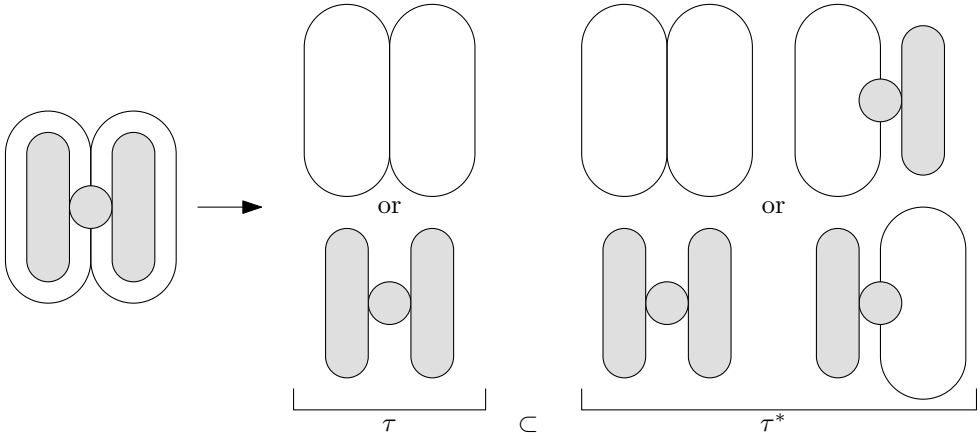


Figure B.2: Non-minimal solder actions that have a minimal-solder action in common.

Bibliography

[ACDR02] D. Applegate, W. Cook, S. Dash, and A. Rohe. Solution of a min-max vehicle routing problem. *INFORMS Journal on Computing*, 14(2):132–143, Spring 2002.

[CCJ90] B.N. Clark, C.J. Colbourn, and D.S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86:165–177, 1990.

[CMWZ01] G. Călinescu, I. Măndoiu, P. Wan, and A. Zelikovsky. Selecting Forwarding Neighbors in Wireless Ad Hoc Networks. In *Proceedings of the 5th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 34–43. ACM, 2001.

[Coo05] W. Cook. The Concorde Traveling Salesman Problem Solver. `http://www.tsp.gatech.edu/concorde.html`, January 2005. Accessed April 2009.

[Dan51] G.B. Dantzig. Maximization of Linear Function of Variables Subject to Linear Inequalities. In T.C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347, 1951.

[DFJ54] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling salesman problem. *Operations Research*, 2(4):393–410, 1954.

[DYC07] J. Dong, N. Yang, and M. Chen. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, volume 37 of *Operations Research/Computer Science Interfaces*, chapter Heuristic Approaches for a TSP Variant: The Automatic Meter Reading Shortest Tour Problem, pages 145–163. Springer US, 2007.

[EN09] K. Eikland and P. Notebaert. Mixed Integer Linear Programming solver. `http://sourceforge.net/projects/lpsolve`, February 2009. Accessed May 2009.

[FCB01] M. Franceschetti, M. Cook, and J. Bruck. A Geometric Theorem for Approximate Disc Covering Algorithms. Technical Report ETR035, California Institute of Technology, 2001.

[GG78] B. Gavish and S.C. Graves. The Travelling Salesman Problem and related problems. Technical report, Operations Research Center, Massachusetts Institute of Technology, 1978. Working paper GR-078-78.

[GHP06] D.J. Gulczynski, J.W. Heath, and C.C. Price. *The Close Enough Traveling Salesman Problem: A Discussion of Several Heuristics*, volume 36 of *Operations Research/Computer Science Interfaces Series*, pages 271–283. Springer US, 2006.

[GP02] G. Gutin and A.P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*, volume 12 of *Combinatorial Optimalization*. Kluwer Academic Publishers, 2002.

[HK70] M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[HK71] M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1:6–25, 1971.

[Hon72] S. Hong. *A Linear Programming Approach for the Traveling Salesman Problem*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, 1972.

[ILO09] ILOG. CPLEX Mathematical Programming Optimizers. `http://www.ilog.com/products/cplex/`, Accessed May 2009.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[Kha79] L. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

[MO96] J.N. MacGregor and T.C. Ormerod. Human performance on the traveling salesman problem. *Perception & Psychophysics*, 58(4):527–539, 1996.

[MTZ60] C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

[NV06] S. Narayanappa and P. Vojtěchovský. An Improved Approximation Factor for the Unit Disk Covering Problem. In *Proceedings of the 18th Canadian Conference on Computational Geometry*, pages 15–18, 2006.

[RBL96] J. Renaud, F.F. Boctor, and G. Laporte. An improved petal heuristic for the vehicle routing problem. *Journal of the Operational Research Society*, 47:329–336, 1996.

[Rei94] G. Reinelt. A Case Study: TSPs in Printed Circuit Board Production. In J. Hartmanis G. Goos, editor, *The Traveling Salesman Problem: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*, chapter 14, pages 187–199. Springer Berlin / Heidelberg, 1994.

[Rei08] G. Reinelt. The Travaling Salesman Problem Library. `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`, August 2008. Accessed April 2009.

[SP08] J. Saalweachter and Z. Pizlo. *Non-Euclidean Traveling Salesman Problem*, volume 21 of *Optimization and Its Applications*, pages 339–358. Springer New York, 2008.

[SSS06] P. Schittekat, M. Sevaux, and K. Sörensen. A Mathematical Formulation for a School Bus Routing Problem. In *Proceedings of the IEEE International Conference on Service Systems and Service Management (ICSSSM'06)*, pages 1552–1557, Troyes, France, 2006.

[ST04] D. Spielman and S. Teng. Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time. *Journal of the ACM*, 51(3):385–463, 2004.

[Wol98] L.A. Wolsey. *Integer Programming.* Wiley-Interscience, 1998.